Software Requirements Specification

for

PARK IN PEACE

Version 1.0 approved

Prepared by Isaac Chun Jun Heng

19 November 2023

Revision History

rev	changes	author
20231119001	*Finalized SRS*	Isaac Chun
20231118002	*Touched up some parts of the SRS*	Isaac Chun
20231118001	*Finished first draft of SRS*	Isaac Chun
20231117001	*Changed User Interface images*	Isaac Chun
20231116002	*Added user interface images and finished some of section 3*	Isaac Chun
20231116001	*Add Section 2*	Isaac Chun
20231114001	*Add Section 1*	Isaac Chun
20231114001	*Created SRS document*	Isaac Chun

Table of Contents

1. Introduction

Purpose

The purpose of this Software Specifications Document (SRS) is to provide possible stakeholders such as developers, testers and project managers an understanding of the full descriptions for **Park in Peace**. It also serves as a document to describe the requirement specifications for **Park in Peace**, to facilitate the development and production process. The document outlines all system features, use case models and descriptions, functional and non-functional requirements. The goal of **Park in Peace** is to assist users in finding nearby carparks to their searched location by providing real time data such as carpark availability and pricing, so as to allow the user to make an informed choice on their selection. The Park in Peace application is built for Android using Kotlin and Jetpack Compose, while using Ktor and a dependency injection framework called Koin for the backend server.

Document Conventions

This section describes all the standards and typographical conventions that will be followed in subsequent parts of the SRS document.

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

All text in the sections are **justified**.

The font and font sizes used are as follows:

Table of Contents	Arial, 11
Heading 2	Times New Roman, Bold, 14
Text	Times New Roman, 12

Intended Audience and Reading Suggestions

This section describes the different types of readers that this document is intended for, and provides suggestions to the sequence of reading this document.

This document is intended for all relevant stakeholders, which includes potential consumers, the development team, the testing team, the project manager, as well as other relevant documentation writers.

The SRS document follows a sequential order, starting with the introduction where it states the purpose and product scope of **Park in Peace**. Then, the description of our product is stated, which is followed by the system features and ending off with our non-functional requirements of **Park in Peace**. The document then ends with an appendix…

All stakeholders mentioned in this section should read the document starting from the introduction, which encompasses → 1.1 Purpose and 1.2 Document Conventions, as well as the attached data dictionary in Appendix A to have a brief introduction to the purpose of **Park in Peace**, as well as to understand the key technical terms which would be used in the following sections.

The **Park in Peace** development team and project managers should read through this document in sequential order to provide themselves with a high level understanding of the product, and the functionalities that would be necessary in developing the product.

The testing team can just read Section 4 (System Features) and Section 5 (Non-Functional Requirements) to understand their testing requirements, and create relevant Black Box Testing to ensure the correctness of the application.

Product Scope

The following section provides a short description of **Park in Peace's** purpose, benefits, objectives and goals.

Every year, the number of Singapore resident households which own a vehicle or motorcycle has been steadily increasing. The convenience that a vehicle brings to an individual as well as his/her family is one of the key reasons why many individuals might want to own a vehicle. The ease of mode of transport is essential in today's fast paced world, where people aim to get between locations as fast as possible to reduce the time wasted due to traveling. Furthermore, as the demand for cars increases, there would be increasing demand for car park lots and the necessary applications that complement it.

Also, there are difficulties in finding car parks through current navigation techniques. Current navigation techniques are too focused on navigation from one place to another, and do not consider the best place to park your vehicle with vacancy at the destination. This leads to inconvenience of searching for nearby carparks, and the possibility of the car park chosen to be full and not accommodating for a user's vehicle type.

As of now, there exist several apps out there that provide car park lot vacancies in real time, but none of which provides the user an interactive map to pan around and present the data in an easy-to-read fashion. Also, they do not provide a means of navigation between the user's location and the specified location, making the user fumble by having to do more work.

Hence, our team intends to bridge all these gaps, by improving on existing market products, through allowing the users to pan an interactive map and search for nearby carparks with ease, presenting the results either on the map, or in a list format. The user would not need to swap between apps just to check for the carpark vacancies as all the information is enclosed in our application, allowing safer driving and better pre-planning before embarking to a location.

Park in Peace's target audiences are any residents in Singapore who want to travel between locations and find a place to park their vehicle to best suit their requirements and specifications.

References

This section includes references that are essential to **Park in Peace**.

- Kotlin Documentation: https://kotlinlang.org/docs/home.html
- Ktor Learn: https://ktor.io/learn/
- Koin: https://insert-koin.io/
- Android Basics Course: https://developer.android.com/courses/android-basics-compose/course
- Android Studio: https://developer.android.com/studio
- LTA Datamall: https://datamall.lta.gov.sg/content/datamall/en/dynamic-data.html
- URA API: https://www.ura.gov.sg/maps/api/#introduction
- Data.gov Carpark Information: https://beta.data.gov.sg/collections/148/view
- OneMap API: https://www.onemap.gov.sg/apidocs/
- MapBox API: https://docs.mapbox.com/api/maps/
- Jetpack Compose: https://developer.android.com/jetpack/compose

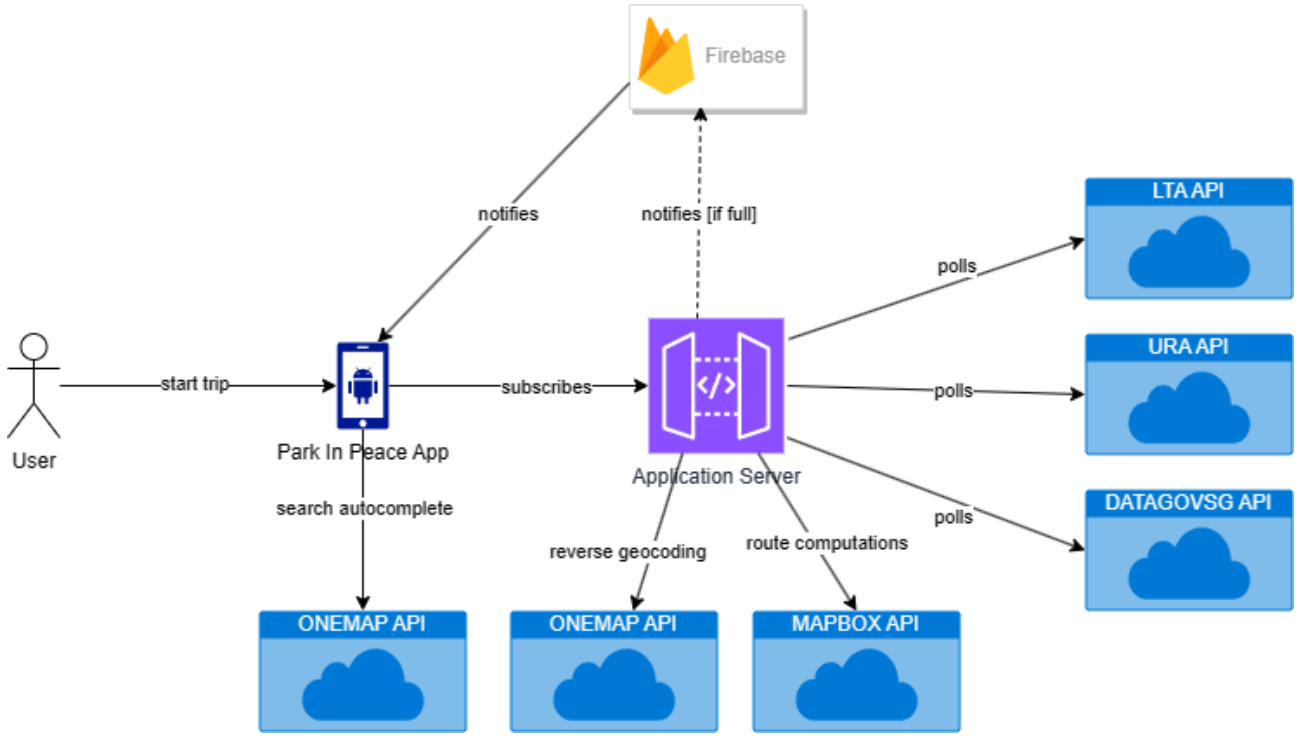
2. Overall Description

Product Perspective

Park in Peace is built for Android, and is a standalone, self contained mobile application which aims to improve existing features and provide new functionalities. The purpose of **Park in Peace** aims to help users find a car park near their specified location, which fits their descriptions, either by lowest price or highest vacancy, or other types of filters. The application automatically does the calculations and abstracts all complexities from the user, allowing the user to have easy accessibility to the features of the application.

Park in Peace connects map rendering APIs such as Mapbox API, and the data provided through an API on car parks are provided to the public by **data.gov.sg**. The Mapbox API hides away all the irrelevant information such as nearby grocery stores, food courts, restaurants and focuses mainly on the roads themselves. From the relevant data fetched from the data APIs, carpark information is

plotted as pins on the map, allowing users to have a visual representation of nearby carparks in the map without needing to swap to another app.



Product Functions

This section summarizes the major functions the product must perform or let the user perform.

The **Park in Peace** recommended flow is to first search for a location, which then finds nearby carparks to that location. The user can then sort this carparks based on some order, and they can then optionally choose to save that location for future use. The following describes the high level features of this process.

1. **Search Location in Singapore**

- Users can search for an address by a valid Singapore postal code
- Users can search for an address by building/street name
- Users can search for address by Singapore mailing address
- Users can search through saved locations such as frequented locations or custom locations such as Home and Work

2. **Find Nearby Carparks**
- System must be able to find nearby carparks near the user input location
- System will automatically conduct API calls to relevant APIs to get nearby carpark details
- System must then plot the nearby carparks on the map and show the data to the user

3. **Sort Nearby Carparks**
- User can choose to sort the results on categories such as vacancy, price, time taken and distance to go to location.
- User can choose to sort the nearby carparks in descending or ascending order.

4. **Save Custom Location**
- Users can input a location and save it as a custom location
- The system saves the data into preferences and reloads it the next time the user starts the app
- Users can edit the names of these locations
- The location must be available to be clicked on for the map to navigate to this saved location easily

User Classes and Characteristics

This section describes the various user classes that we anticipate would use **Park in Peace**. They are differentiated based on the frequency of use, subset of production function used, technical expertise, security or privilege levels. There are 2 categories that classify the main users of our application. They are listed as follows:

1. **Individuals who own a vehicle and drives frequently**

Attribute	**Description**
Frequency of Use	High
Subset of Product Functions Used	All
Technical Expertise	Medium
Security/Privilege Levels	User Mode
Characteristic/Comments	The individuals who fall into this category would be the most frequent users of the application, as they might want to find out the availability and information of a car park before they choose to leave for that destination.
Example	If a particular mall like Westgate has no lots, the user can change his choice to go somewhere else at that

	point of time.

2. **Individuals who want to do pre-planning**

Attribute	**Description**
Frequency of Use	Medium
Subset of Product Functions Used	Searching and finding nearby carparks
Technical Expertise	Low
Security/Privilege Levels	User Mode
Characteristic/Comments	The primary purpose of users in this category would just be users who want to do more research or just play around with the app. Hence, the functionalities that **Park in Peace** provides would be sufficient for those who just want to see which car parks are nearby from anywhere in Singapore and conduct pre planning.
Example	Teachers who are planning a field trip and need some possible options of carparks. They may not need all the information such as availability as the trip might not be on the same day.

Operating Environment

This section will provide the descriptions of the environment in which the software operates on, including the hardware platform, operating system and versions and any other software components or applications that must peacefully co exist.

Product Environment: **Park in Peace** would need to be run on an mobile phone running Android, with minimum SDK version 33 (Android 13)

Development Environment:

Development Environment	**Description**	**Version**
Android Studio	Android Studio is the official integrated development environment for Google's Android	Android Studio Giraffe | 2022.3.1 Patch 4

	operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems.	
Kotlin	Kotlin is a cross-platform, statically typed, general-purpose high-level programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of Kotlin's standard library depends on the Java Class Library, but type inference allows its syntax to be more concise.	1.9.10
Jetpack Compose	Jetpack Compose is Android’s recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.	1.5.1
Ktor	Ktor is an open-source framework built by JetBrains for building asynchronous servers and clients using Kotlin programming language. Ktor is popular because it offers a lightweight and flexible approach to building web applications, with a focus on extensibility and modularization	2.3.5
Koin	Koin is a pragmatic lightweight dependency injection framework for Kotlin developers, developed by Kotzilla and open-source contributors. Koin is a DSL, a light container and a pragmatic API.	3.5.1

Design and Implementation Constraints

This section describes any items or issues that will limit the options available to developers. Such limitations may include hardware limitations such as timing requirements, memory requirements, specific technologies, tools and databases to be used, etc.

Possible Limitations

Service/API	Description
Data.gov.sg	The data.gov.sg API may not be comprehensive, or does not have all the required information of all car parks, such as missing information about pricing rates for some car parks, or may not return the entire list of car parks in Singapore. The API may also be slow and throw exceptions to our server, which may cause application crashes.
Mapbox API	The Mapbox API does not directly support Jetpack Compose, and is generally configured for typical fragment type activities. Creating a map may also take a significant amount of processing power on the mobile device, especially with Jetpack Compose, where UI elements are recomposed when a variable has changed.
OneMap API	Queries to the OneMap API to return locations in Singapore might take longer than expected and throw exceptions if the user's internet connection speed is slow, or if the API fails.

Design Considerations and Standards

Design Consideration/Standards	**Details**
Programming Standards	1. Adhere to the conventions of the Kotlin language. 2. Use proper indentation to improve readability and formatting of code. 3. Embrace immutability by default. 4. Practice layered architecture and avoid bi-directional data flow. 5. Use enums extensively to enforce type safety and input validation. 6. Null safety - nullable values should be checked before use. Values that are nullable should be explicitly marked. 7. Global variables and static are heavily discouraged. Dependency injection should be preferred to promote loose coupling. 8. Composition over inheritance and code to interfaces instead of implementations 9. Boundary classes should be backed by interfaces and adapters to minimise impacts to application code should external APIs change. 10. Assertions should be used to verify invariants and catch logic errors early.
User Interface	1. All user interface design should adhere to UI-related

	requirements, as well as any pre-planned UI mockups. 2. All user interface color schemes should be dynamic, automatically changing to the theme of the user's phone to ensure user's preferences are still met.
File Structure	1. Classes and source files that used by either the client or server should be in their respective folders. ie. client/ or source/ 2. Classes and source files that are used by both should be in the common folder, and added as a dependency. 3. There should be several default folders as follows: a. **models**: source/class files b. **viewmodels**: viewmodel controllers c. **views**: view files that has specific UI functions to be called d. **utils**: any utility files to be used throughout the package

User Documentation

This section lists the currently available user documentation components that exist in the repository.

1. The repository/folder contains a readme that highlights the structure of the project alongside on how to build the project.
2. The repository also contains two files **SECRETS.properties.sample** and **android.ENV.properties.sample** to guide new developers on how to setup their version of the project to start development. This is because we do not want to put API keys that could be made public, which increases the security of our program and allows verified developers to start development.
3. The repository contains a demo video to help users or new developers to understand the flow of the latest working copy, and help with understanding the key features of the application.

Assumptions and Dependencies

This section notes down several assumptions or factors that could affect the requirements stated in this SRS.

1. We assume that the data retrieved from APIs such as OneMap API and Data.gov.sg APIs are all accountable and up-to-date, with minimal error.
2. We assume that the user has a mobile phone that is running Android UI 13, and has a **minimum SDK level of 33** to fully utilise the performance bonuses of our application.
3. We assume that users are connected with a strong internet connection, such that calls to APIs will be completed with minimal delay.

4. The application depends heavily on data.gov.sg APIs to work, even though we have attempted to circumvent that by providing predicted data. However, without the data provided by data.gov.sg APIs, the app would not function as expected.
5. The development of **Park in Peace** assumes that the developer is experienced around gradle and Kotlin, as well as being familiar with Android Studio.

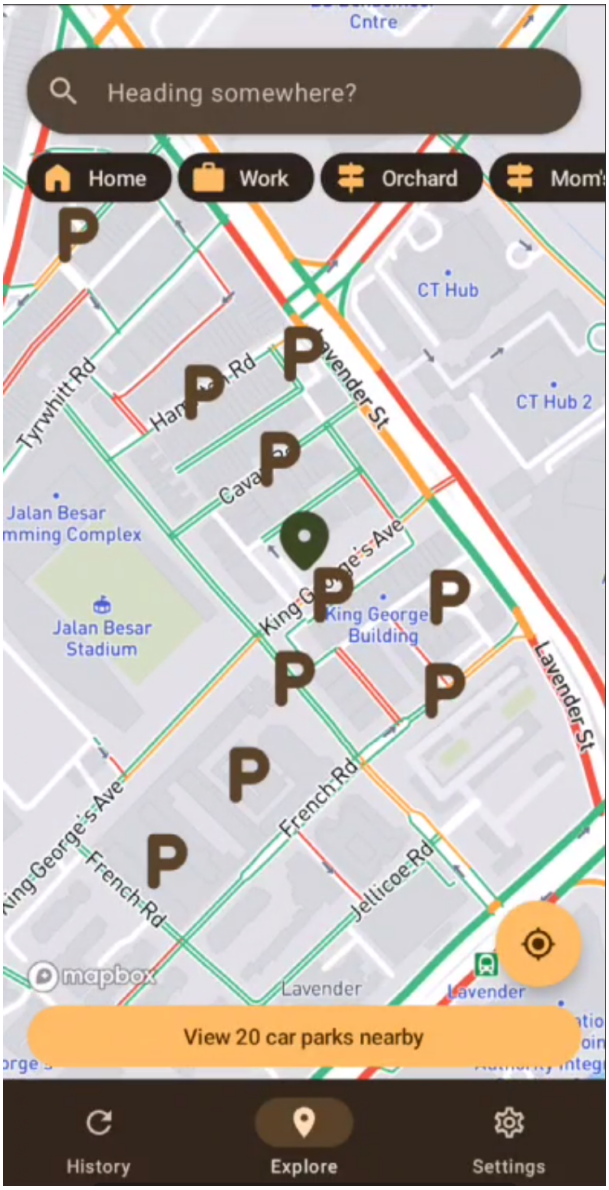
3. External Interface Requirements

User Interfaces

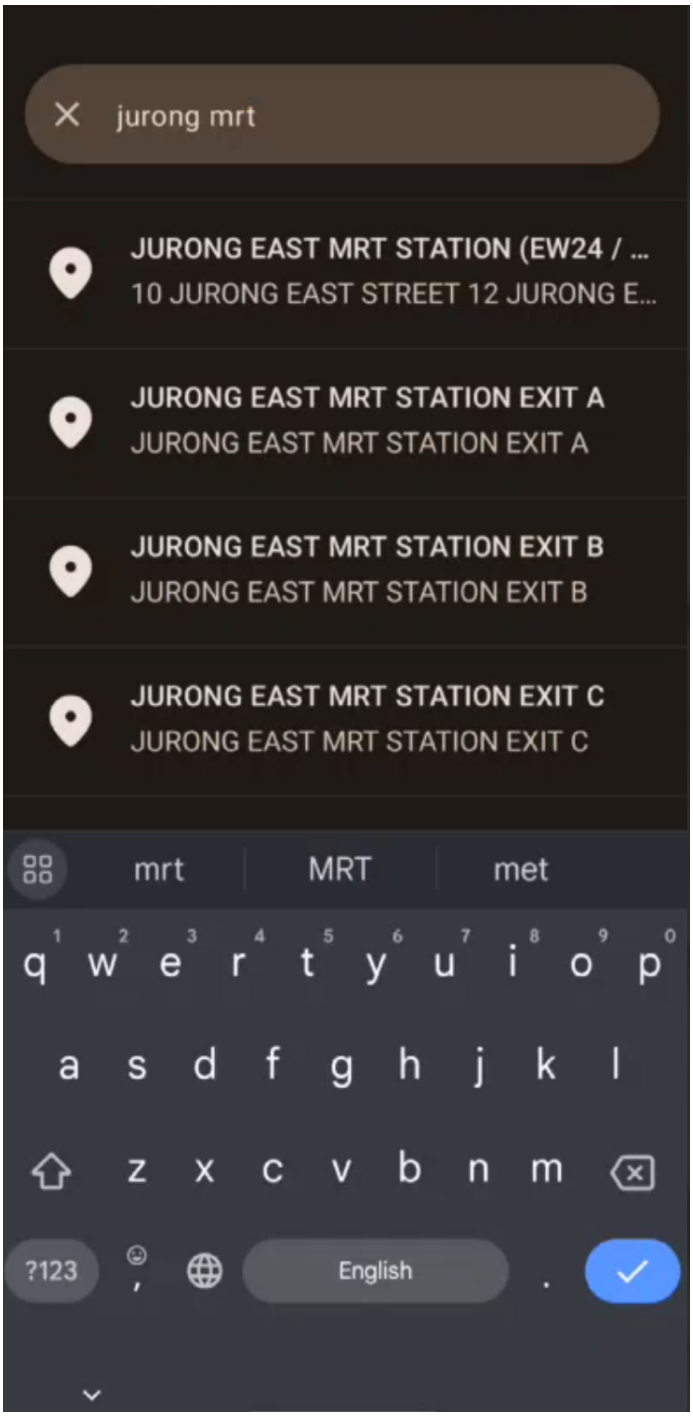
The following section provides an overview of the logical characteristics of each interface, providing sample screen images that depict the GUI standards to be followed, and the standard buttons and functions that will appear. Development of the UI screens should be followed strictly, and any improvements would result in a change in the images in this section.

Details on UI specifications will be located in the subsequent sections under non-functional requirements.

1. **Main View**
    - Users can click on the search bar to search for any location within mainland Singapore.
    - Users can also click on the pins below the search bar to search at their previously saved custom locations
    - Users can click on the view carpark nearby to open up the list of nearby carparks if there are any.
    - Users can click on the re center button to recenter the map back to their position
    - Users can click on any button in the navigation bar to go to any other screen

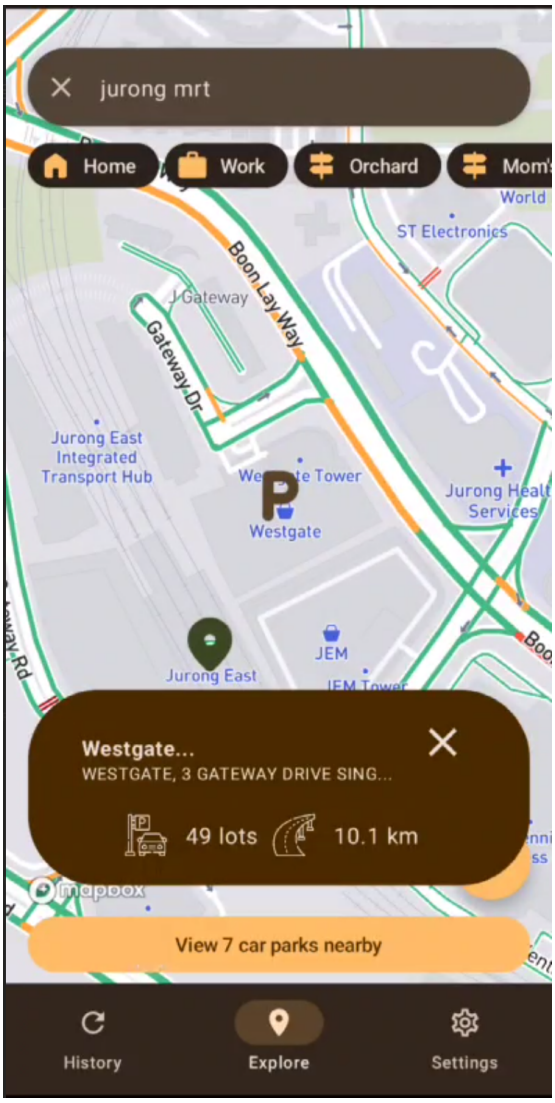

Heading somewhere?
Home
Work
Orchard
CT Hub
CT Hub 2
Tyrwhitt Rd
Jalan Besar
Jalan Besar Stadium
King George's Ave
King George Building
Lavender St
French Rd
Jellicoe Rd
Lavender
mapbox
View 20 car parks nearby
History
Explore
Settings

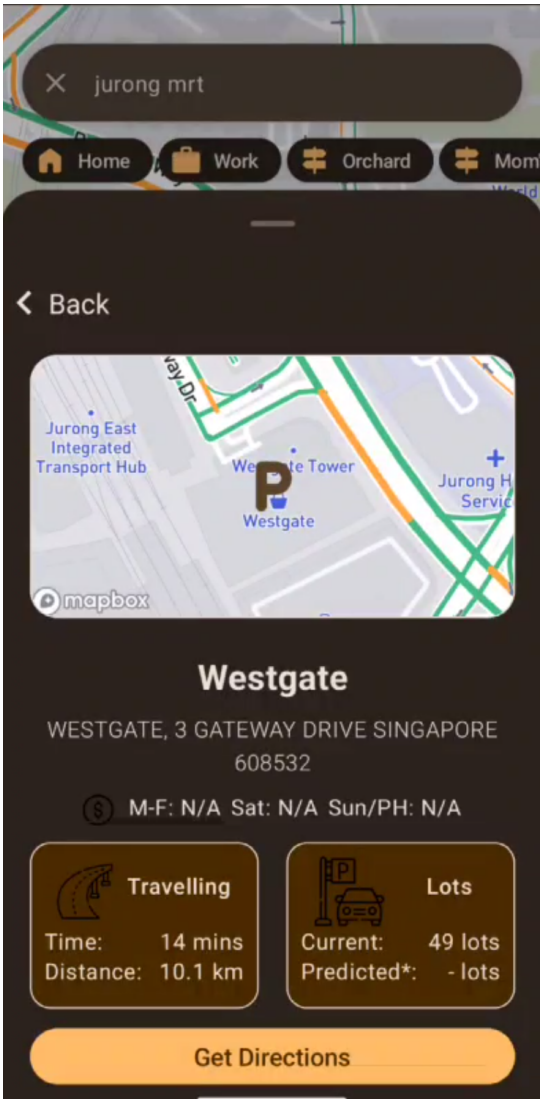
2. **Search Page with Suggested Responses**
    - The results should be shown in a list format that shows the address name and the location stored at that address.
    - The list is scrollable and users can select from any location.



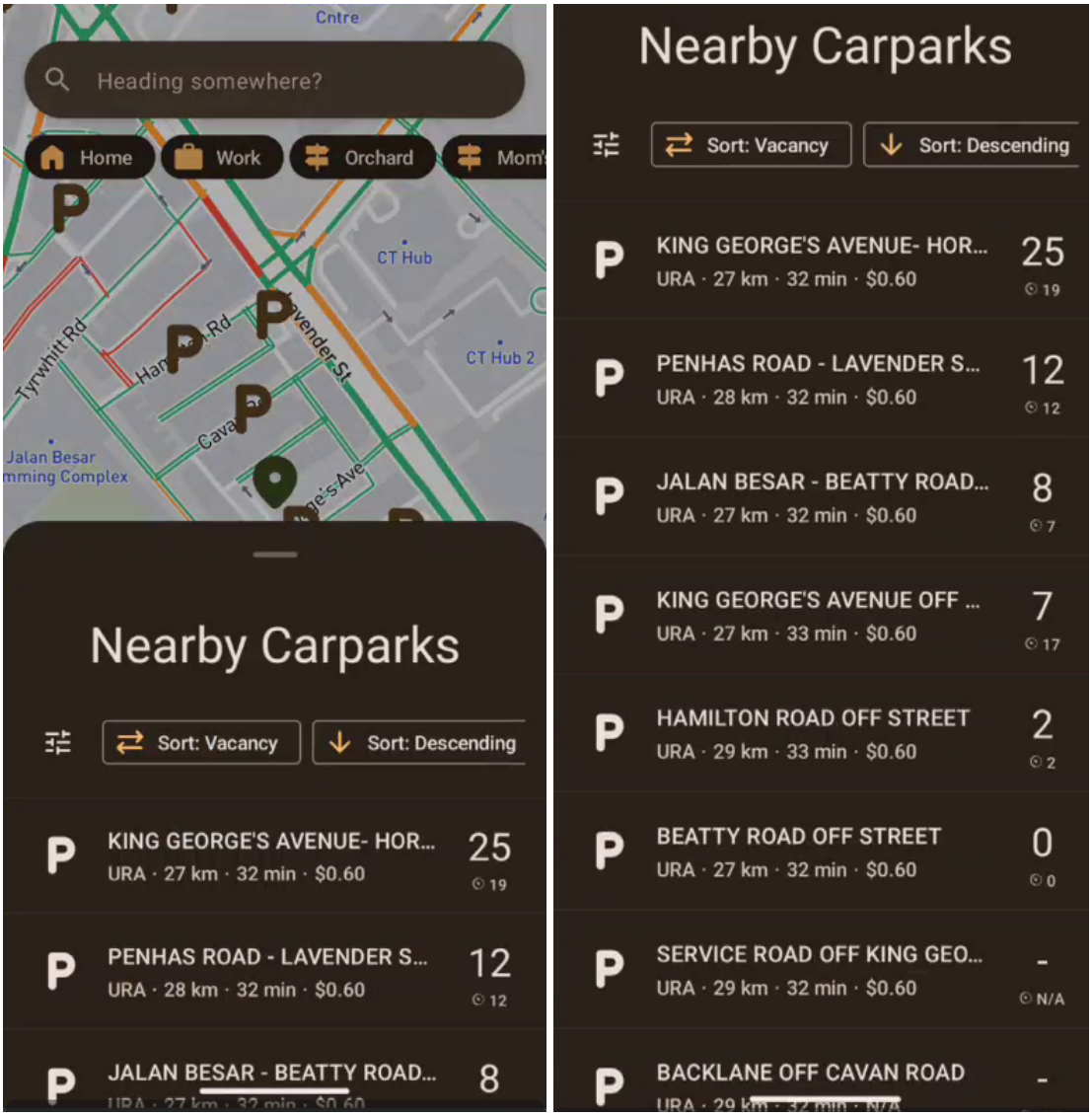
3. Carpark Information Prompt and Extended

- The prompt can be clicked to open up the extended car park information, which shows the full information of the carpark.
- The user can also click on get directions to be redirected to Google Maps



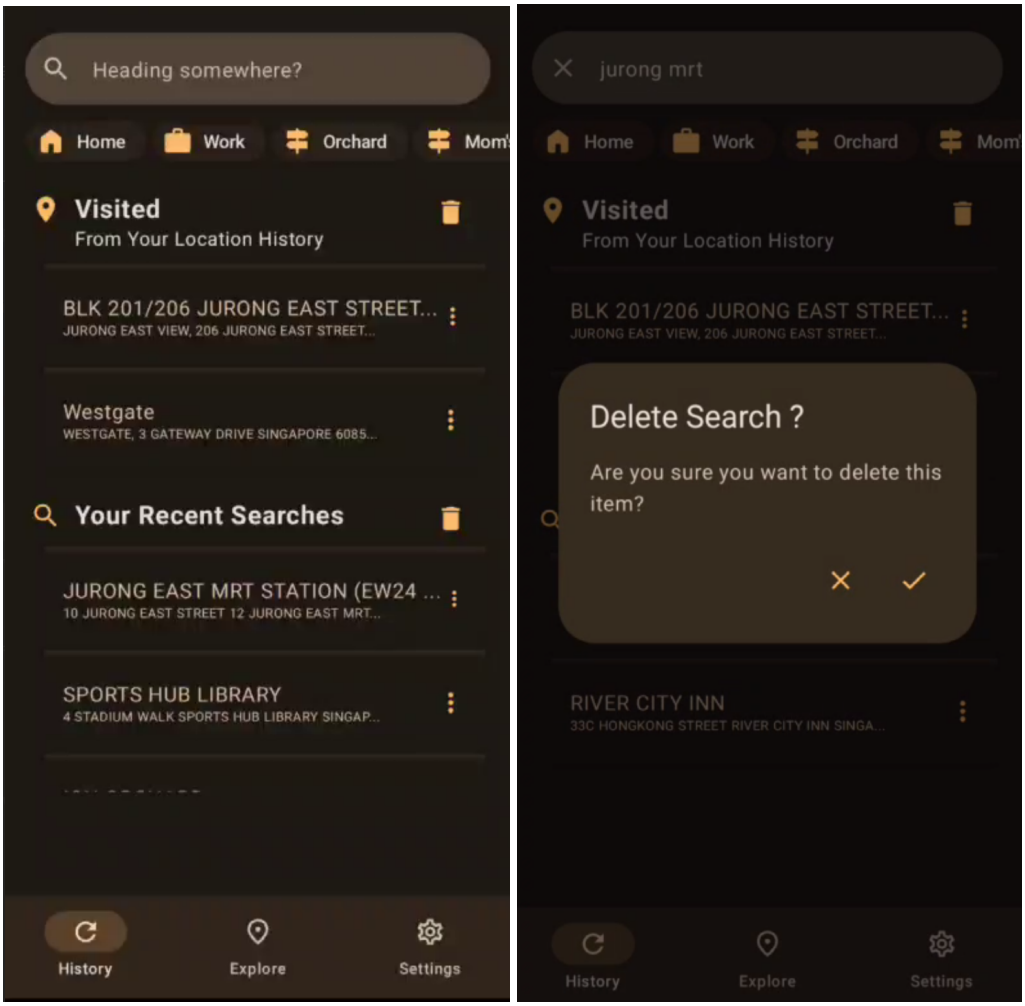


4. **List of Carparks Popup and Extended**
    - The nearby carparks are also shown as a list.
    - Users can click on the sort buttons to either sort by vacancy or other types of sorting methods.
    - User can also choose either descending or ascending order to sort.
    - The small number beneath the lots number is the predicted number of lots for that car park.



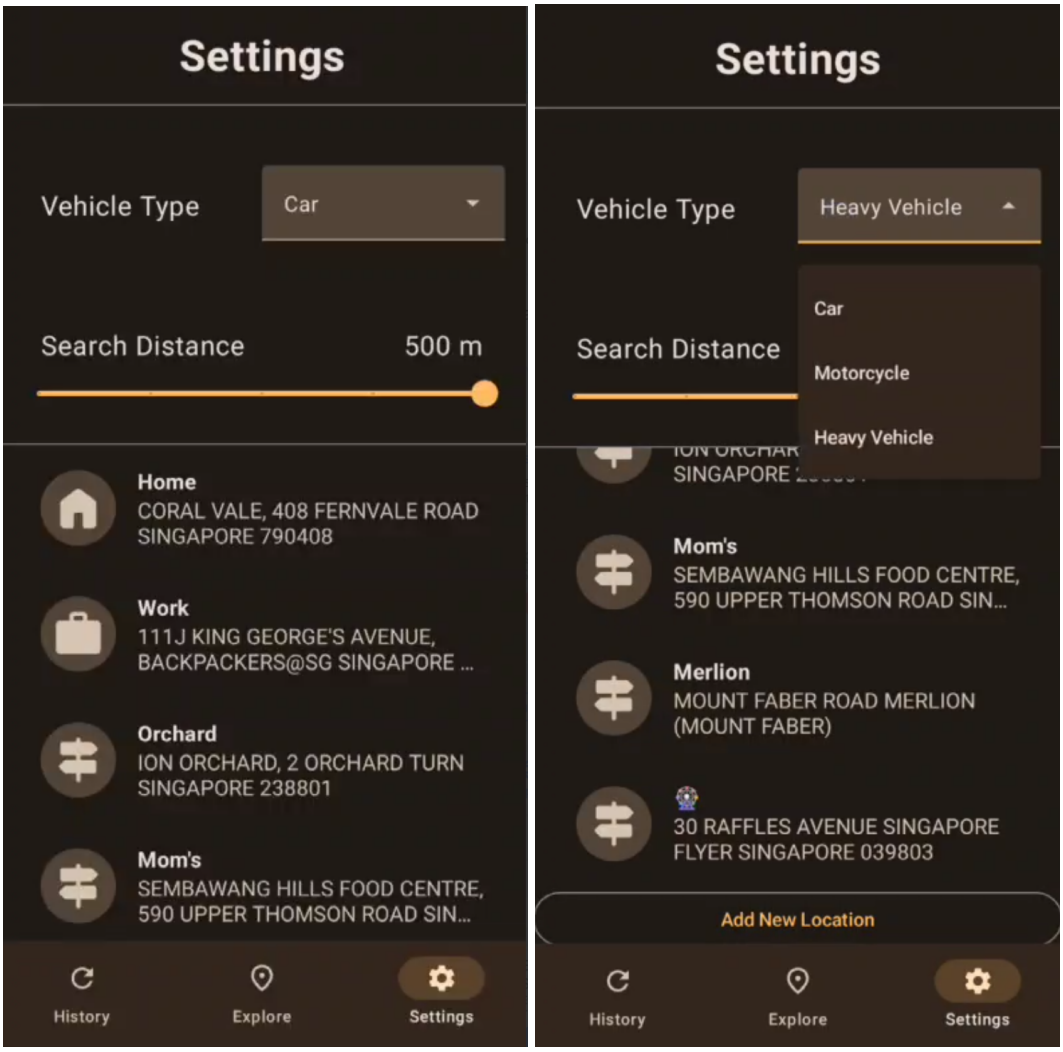
5. History Page

- The previously visited and recently searched locations show up as a list in our history page.
- Users can also click on the delete icons to delete an entry from the history page.
- Users can also click on any visited or recent search entry to search at that location.



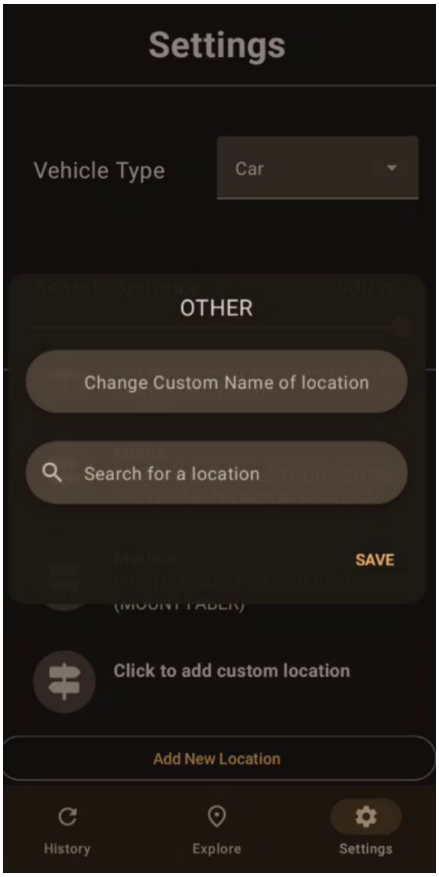
6. Settings Page

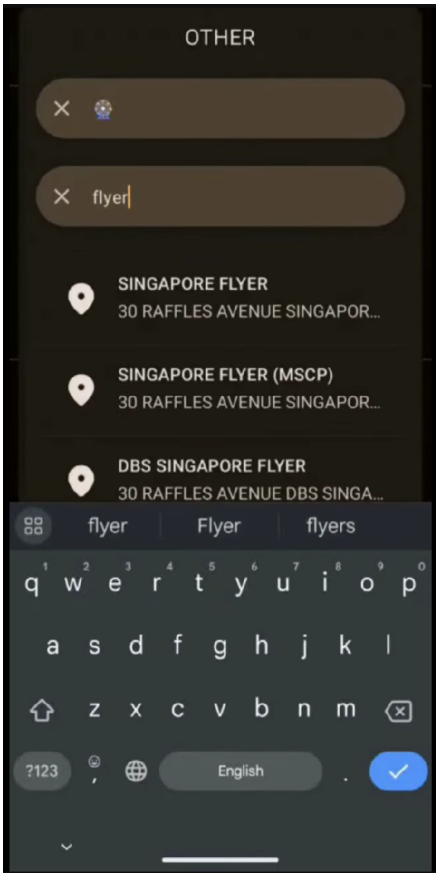
- Users can select their vehicle type by clicking on the drop down
- Users can also change their search distance via an intuitive slider
- The list of saved locations are also stored in a scrollable list that is easy to read.
- The add new location button is also large enough for users to add a new location easily.

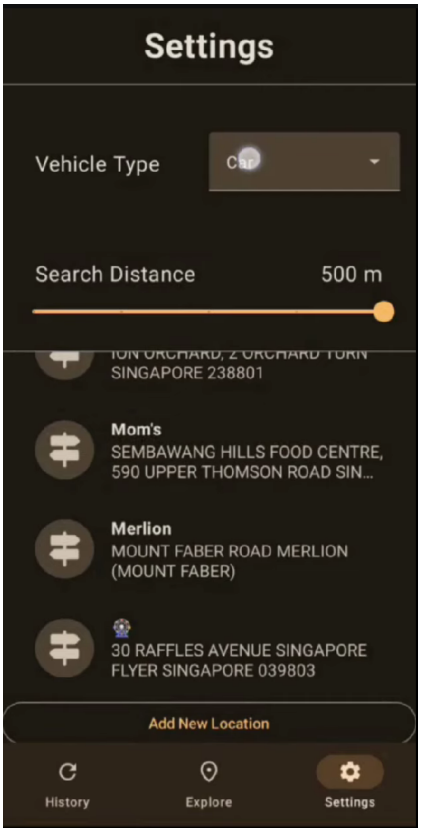


7. Add New Location

- The UI only contains two simple text fields for users to click and enter information
- Upon searching for a location in the add new location, the suggested results are shown as small list below the text field.







Hardware Interfaces

This section describes the requirements for hardware interfaces for the application to perform to it's expected functionalities. This includes supported device types, the nature of the data and control interactions between the software and hardware, and the communicate protocols to be used.

1. **Client Requirements**

- **Park in Peace** supports mobile phones running on Android, with **minimum SDK version of 33** as well as having the latest software updates installed (Android 13). The device must also have a working browser such as Google Chrome or Samsung Internet, such that the application is able to reroute the user to online maps if that phone does not have a dedicated maps application.
- The application also requires the approval of the user to get the user's current location, as well as the ability to use the mobile phone's capability to connect to the internet.

2. **Server Requirements**

- The backend server must be hosted and run on a local computer. The server performs basic functionalities such as Create, Read, Update, Delete (CRUD) operations as well as connect to relevant APIs via REST API from the client to the server back end. The server also

handles all the calculations of API calls such as data.gov.sg API calls, and returns the relevant details to the client.
- The server must also be left on for a day, so that it is able to return predicted lots data to the client from the previous day.

Software Interfaces

This section describes the connections between this product and other specific software components such as databases, operating systems, tools, libraries or integrated commercial components.

1. **Software Components**

- The back-end server of **Park in Peace** is implemented using Ktor as the framework to build asynchronous servers and clients. The server also uses Koin as a dependency injection framework. The client connects to our server through Rest API.

- APIs used in **Park in Peace**:
    1. Data.gov.sg API: To retrieve HDB carpark information
    2. LTA Datamall API: To retrieve LTA, HDB and URA carpark information
    3. URA API: To retrieve URA carpark information and pricing
    4. OneMap API: To retrieve proper location results from user queries
    5. MapBox API: To display an interactive map on the screen.

2. **Software Architecture**

- **Park in Peace** uses Model-View-ViewModel (MVVM) seperation, to separate program logic and user interface controls. **Park in Peace** also follows design principles such as code to interfaces, strategy and adapter patterns, null safety, as well as dependency injection.

Communications Interfaces

This section discusses the requirements associated with the communications functions for **Park in Peace** to work accurately and to our specifications.

The client of our application implements the REpresentational State Transfer architectural style which provides standards between computer systems on the internet, making it easier for systems to communicate with each other. Hence, our client and server can be developed in parallel, each with their own logic and separating them from each other, decoupling the application's dependencies. For proper communicate, the client will communicate with the server by sending requests to retrieve or modify data, and the server would send back their responses to these requests.

For example, the client can communicate with the server by invoking the GET or POST requests, which will allow the client to retrieve some data from the server or send data for processing, and may not modify the server's state. It is to be noted that GET performs better than POST due to the simpel nature of appending the values into URLs, and that this method supports only string data

types. Also, due to the possible time taken by these calls due to the communication overhead from client to server, the code must be written asynchronously for better performance.

4. System Features

This section illustrates the functional requirements of the product by system features through use cases. The section also contains a list of functional requirements that are stated first, such that the features listed below are more compact and precise with no repeats of functional requirements in sub features.

The list of functional requirements are as follows:

Functional Requirements

F1 The user must be able to search for car parks near a location.
- **F1.1** The system must support searching by a valid Singapore postal code.
- **F1.2** The system must support searching by building/street name.
- **F1.3** The system must support searching by full Singapore mailing address.
- **F1.4** The system must support searching by current user location.
- **F1.5** The system must support searching through the use of saved locations such as
  - **F1.5.1** Frequented locations
  - **F1.5.2** Custom locations, such as “Home” and “Work”
- **F1.6** The system must reject any address/location not within mainland Singapore, and allow the user to pick again.

F2 When searching, the system must present the closest car parks to the destination searched within a preset search radius.
- **F2.1** If there are no car parks within the search radius, the search must instead return up to 3 nearest car parks closest to the location regardless of the price.
- **F2.2** For each subsequent result beyond the first, if the next best car park by distance is further than 1km beyond the distance of the first car park to the desired location, they must be omitted.

F3 The system must visually identify the location indicated in **(F1)** via UI elements such as drop pins.

F4 The system must visually identify the locations of the nearest car parks on the map via UI elements such as drop pins.
- **F4.1** The drop pin must be greyed out in the case that there are no vacant lots available at the indicated car park.
- **F4.2** The drop pin must display full information about the car park, as elaborated in **(F5)**, when tapped on.

F5 When returning the list of search results, the system must include the following information about each car park.
- **F5.1** The name of the car park.
- **F5.2** The address of the car park

F5.3 The estimated distance of the car park from the original location, in metres.

F5.4 The price of parking at the location, in dollars per hour.

F5.5 The number of vacant lots in the car park, in real time.

F5.5.1 The lots displayed must correspond to the user's current vehicle type, if such information had been shared prior by the user.

F5.6 The estimated travelling time to reach the car park by car, assuming ideal traffic conditions.

F5.7 The projected number of vacant lots in the car park at the expected time of arrival, assuming **(F5.5)** and that the user departs immediately; with the use of historical data.

F5.7.1 The lots displayed must correspond to the user's current vehicle, if such information had been shared prior by the user.

F5.8 The special features of the car park, if any, including but not limited to:

F5.8.1 Electric vehicle charging lots

F5.8.2 Vehicle washing bays

F6 The user must be able to set their global/long-term user preferences.

F6.1 The user must be able to specify their vehicle type, such as

F6.1.1 Car

F6.1.2 Motorcycle

F6.1.3 Heavy vehicle

F6.1.4 (Electric Vehicle)

F6.2 The user must be allowed to save custom locations.

F6.2.1 Each location must be a valid location within Singapore

F6.2.2 Each location must have a custom name given by the user, such as "Home" and "Work"

F7 The user must be able to add filters to the search.

F7.1 The following filters must be implemented

F7.1.1 Pricing

F7.1.2 Preferred distance from destination (preset search radius)

F7.1.3 Mode of payment

F7.1.4 Vehicle Type

F7.1.5 Minimum vacant lots

F7.2 Results that do not match the filter criteria must be hidden.

F7.3 The user must be able to combine multiple filters.

F7.4 The user must be allowed to clear a filter.

F7.5 The user must be allowed to clear all filters at once.

F8 The user must be allowed to define a sorting criteria.

F8.1 The following sorting criteria must be made available.

F8.1.1 Sorting by price – the system must sort by price followed by distance.

F8.1.2 Sorting by distance – the system must sort by distance followed by price.

F8.1.3 If unspecified, the system should default to **(F8.1.1).**

F9 The system must redirect the user to Google Maps for directions.

F9.1 The system should support choosing between opening Google Maps on a web browser or via the installed app.

F10 The system must be able to store the previous search history of the user.

F10.1 The system must be able to let user clear a search history

F10.2 The system must be able to let user clear all search history at once

F11 The system must be able to store the user's visited car parks.

F11.1 The system must be able to let user clear a visited car park history

F11.2 The system must be able to let user clear all visited car park history at once

F12 While travelling, the system must automatically monitor the availability of the chosen carpark.

F12.1 If the car park becomes full, the system must automatically select the next best car park based on the preferences selected by the user.

F12.1.1 The system must notify the user through a notification bubble that must include

F12.1.1.1 The information that the car park being full.

F12.1.1.2 The location of the next best car park.

F12.1.1.3 A button to get directions to the new car park on Google Maps.

For the following features, the word use-case means the same as features.

4.1

FIND NEARBY CAR PARK

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-10-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1^**, **F2**, **F3**, **F4^**, **N5**, **N6**
description	*As a driver, I want to explore and compare possible places to park my vehicle, before I set off to my destination.* This use case takes in a guest-supplied location and returns the nearest car parks that the guest can choose to park at.
preconditions	N/A
postconditions	N/A
priority	1
frequency of use	Daily

flow of events	1. **Guest** types in a location in the search bar 2. **SearchViewModel** retrieves a list of closest matching locations to the given query as described in **(4.5 Autocomplete Query)** 3. **Guest** picks one of the suggestions returned by **SearchViewModel** 4. **MapViewModel** narrows down list of car parks close to the provided location 5. If there are nearby car parks found in the carpark list, the **MapUI** plots the results on the map
alternative flows	AF2. The location supplied does not exist or is not within mainland Singapore 1. The system informs the guest that no results were found AF4. No car parks are within preset search radius of the location 1. The system will proceed into **(4.2 Widen Search Area)**.
exceptions	EX1. The guest is not connected to the internet 1. The system informs the guest that he/she is not connected to the internet EX2. The app does not have the “Location” permission 1. The system directs the guest to grant location permission to continue using the app
includes	1. **(4.5 Autocomplete Query)**
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A

notes and issues	N/A
functional Requirements	

4.2

WIDEN SEARCH AREA

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F2.1, F2.2, F4^, N5, N6**
description	*As a driver, when there are no car parks decently near the destination, I want to know about the next best possible car park(s), before I set off to my destination.* This use case follows up on **(4.1 Find nearby car park)** and activates when there is no nearby car park within the preset search radius. As there are no nearby car parks, the system will then return the closest car park to the destination. The system will additionally return up to 2 more car parks provided that they are no more than 1 km away from the first car park found.
preconditions	1. Can only be entered from **(4.1 Find nearby car park)** 2. Location is valid
postconditions	N/A
priority	4
frequency of use	Rarely

flow of events	1. No car parks are within preset search radius of the location 2. **NavigationService** finds the closest car park by distance to the destination. The distance is labelled d. 3. **NavigationService** expands the search range to $d+1\ km$. 4. **NavigationService** retrieves two other car parks within the search range, if any 5. **MapUI** plots the results on the map
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A
notes and issues	N/A

4.3

FILTER SEARCH RESULTS

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F7^, N5, N6**
description	*As a driver, I would like to hide car parks that are overpriced or have no vacancy.* This use case follows up on **(4.1 Find nearby car park)** and allows the guest to add filters to hide car parks that do not meet the criteria.
preconditions	1. Can only be entered from **(4.1 Find nearby car park)**
postconditions	1. Items that do not match the criteria are also hidden from the map
priority	3
frequency of use	Monthly
flow of events	1. Guest opens the nearby car parks list. 2. Guest adds one or more filters. 3. **MapViewModel** hides results that do not satisfy the filter criteria 4. **MapUI** plots the results on the map

alternative flows	AF1. Guest wants to remove a filter 1. Guest removes a filter 2. System shows all results that matches existing filters AF2. Guest wants to remove all filters 1. Guest removes all filters 2. System shows all results AF3. No results after filtering 1. Guest adds a filter and no results satisfies the existing criteria 2. System displays a helpful message indicating that no results matches the guest's criteria
exceptions	N/A
includes	N/A
special requirements	1. The system must return the search results within 1 second. 2. The system must display nearby car parks on the map within 1 second
assumptions	N/A
notes and issues	N/A

4.4

SORT SEARCH RESULTS

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F8^, N5**
description	*As a driver, I would like to sort by price, distance or vacancy when I am in a rush to get to my destination.* This use case follows up on **(4.1 Find nearby car park)** and allows the guest to sort the results by a predefined sorting criteria.
preconditions	1. Can only be entered from **(4.1 Find nearby car park)**
postconditions	N/A
priority	3
frequency of use	Monthly
flow of events	1. **Guest** opens nearby car park list. 2. **Guest** selects a sorting criteria in nearby car park list. 3. **MapViewModel** arranges the results based on sorting criteria
alternative flows	N/A
exceptions	N/A
includes	N/A

special requirements	1. The system must return the search results within 1 second.
assumptions	N/A
notes and issues	For full list of sorting criteria, refer to **(F8)**

4.5

AUTOCOMPLETE QUERY

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1.5^, F10**
description	*As a driver, I would like to have previous destinations saved so that I do not need to recall/go through the hassle of reentering the address of my destination.* This use case is an extension of **(4.1 Find nearby car park)** and allows the guest to pick one location from a list of previously searched locations as the intended destination. Additionally, it also suggests custom locations and frequent locations.
preconditions	N/A
postconditions	1. Successful new searches are saved
priority	5
frequency of use	Daily

flow of events	1. **Guest** is able to view past locations visited and past searches after entering HistoryUI. 2. **Guest** types a location into the search bar in **SearchUI** 3. **SearchViewModel** repeatedly calls into **LocationService** which presents a. all relevant past searches for the guest to pick from b. Closest locations to the query 4. **LocationService** continually refines suggestions as guest continues typing 5. **Guest** picks his intended destination 6. **SearchHistoryDb** saves the query if the guest did not choose the presented suggestions
alternative flows	N/A
exceptions	EX1. Guest cancels the search 1. System discards and does not save the query EX2. Invalid location entered 1. System discards and does not save the query
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

4.6

SET VEHICLE INFO

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F6.1**
description	*As a driver, I would like the application to remember the type of vehicle I drive so that it would not suggest vehicle lots that I cannot park in.* This use case acts when the guest wants to amend the information of their vehicle.
preconditions	N/A
postconditions	The information of the vehicle the guest drives is stored in the application.
priority	7
frequency of use	Rarely
flow of events	1. **Guest** opens the SettingsUI 2. **SettingsUI** presents the guest with options of vehicle types to choose from 3. **Guest** chooses their vehicle type 4. The data will be saved in the **UserPrefsDb**
alternative flows	N/A

exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	See Data Dictionary entry for “Vehicle Type” for full list of vehicle types supported.

4.7

VIEW CAR PARK INFO

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F5**
description	*As a driver, I would like to know the information of the car park I am travelling to, so that I can make an informed decision on what to choose.* This use case acts when the guest clicks to view the additional information of any car park they would like to travel to. These destinations could be either in search results or in the custom locations created.
preconditions	N/A
postconditions	N/A
priority	1
frequency of use	Daily

flow of events	1. **Guest** selects a car park. 2. **CarparkInfoUI** fetches car park information from **MapViewModel** 3. **CarparkInfoUI** displays the information about the car park such as distance from searched location(if applicable), price of parking and availability of car park.
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	It is not possible to show distance from intended location if guest enters this use case via other means other than **(4.1 Find nearest car park)**

4.8

CREATE CUSTOM LOCATIONS

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F6.2**
description	*As a driver, I would like to tag certain locations with a more recognisable name for convenience.* This use case allows the guest to save specific locations with a special name, such as “Home” or “Work”.
preconditions	N/A
postconditions	The custom locations created are saved into the system.
priority	7
frequency of use	Rarely

flow of events

1. **Guest** clicks on create custom location in **SettingsUI**
2. **Guest** types the name the location should be saved under into first text field
3. **Guest** enters address of the location via second text field
4. **SettingsViewModel** repeatedly calls into **LocationService** which presents
    a. all relevant past searches for the guest to pick from
    b. Closest locations to the query
5. **LocationService** continually refines suggestions as guest continues typing
6. **Guest** picks his intended destination
7. **Guest** clicks save to finalise
8. **Guest's** custom location is saved into **UserPrefsDb**

alternative flows	AF1. The **Guest** wants to rename a custom location 1. The **Guest** selects the custom location 2. The **Guest** selects to rename the location 3. The **Guest** renames the location 4. The **Guest** presses to save the name, and **UserPrefsDb** saves it AF2. The guest wants to delete a custom location 1. The guest selects the custom location to be deleted 2. The guest selects to delete the custom location 3. **UserPrefsDb** deletes the saved custom location and displays that the location has been deleted
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

4.9

VISIT CAR PARK

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F12**
description	*As a driver, I would like to get driving directions to my intended car park. Additionally, I would like to be notified if my intended car park is no longer vacant so that I may pick an alternative car park ahead of time instead of getting disappointed and delaying my travel time when I inevitably have to find another car park.* This use case acts as a parent use case for the following use cases: 1. **(4.12 Launch Google Maps)** 2. **(4.10 Monitor car park vacancy)** 3. **(4.11 Log car park visit)**
preconditions	1. Car park exists 2. At most 1 instance of this use case can be active per device. In other words, starting another visit will cancel any previous trip.
postconditions	N/A
priority	2

frequency of use	Daily
flow of events	1. Guest taps on the “Google Maps” button 2. **NavigationService** launches **(4.10 Monitor car park vacancy)** 3. **MapViewModel** launches **(4.12 Launch Google Maps)** for the guest to get directions 4. **Guest** drives to the car park following the directions in (3) 5. **Guest** successfully finds a vacant parking lot 6. **Guest** marks trip as complete 7. **NavigationService** launches **(4.11 Log car park visit)** 8. Use case terminates successfully
alternative flows	AF1. **Guest** cancels the visit mid-trip 1. Child use cases are cancelled 2. **NavigationService** informs the guest that the visit is cancelled 3. **Guest** is redirected to the previous screen
exceptions	EX1. **Guest** was unable to find a parking lot 1. **Guest** cancels the visit 2. **Guest** enters **(4.1 Find nearby car park)** EX2. **Guest** starts another visit while one is currently active 1. Previous visit is automatically cancelled 2. **NavigationService** proceeds as normal
includes	1. **(4.12 Launch Google Maps)** 2. **(4.10 Monitor car park vacancy)** 3. **(4.11 Log car park visit)**

special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must not require the guest to re-input any details again.
assumptions	N/A
notes and issues	The real-time API has a lag time of 3-5 minutes

4.10

MONITOR CAR PARK VACANCY

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F12**
description	*As a driver, I would like to know in real time whether the car park I am travelling to is going to still be available* This use case is utilised when the guest has decided on their preferred car park and will travel to it. The system will run to check the availability of the car park in real time and return it to the guest.
preconditions	The guest has decided on a car park and is travelling to it
postconditions	1. **NavigationService** maintains the subscription while the guest has not reached the destination. 2. **NavigationService** drops the subscription when the guest reaches the destination or cancels the trip or changes destination.
priority	6
frequency of use	Daily

flow of events	1. **Guest** selects their destination car park to travel to. 2. **NavigationService** informs **server** that it wants to be kept updated about destination car park 3. **Guest** starts driving to the destination car park 4. **Server** periodically refreshes the number of vacant lots in the car park 5. **Guest** successfully parks at destination car park 6. Subscription is cancelled
alternative flows	N/A
exceptions	EX1. The carpark the guest is travelling to becomes full 1. **Server** executes warn car park full **(4.13 Warn Carpark Full)** and the following use cases
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

4.11

LOG CAR PARK VISIT

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Guest
described by	**F1.5.1, F11**
description	*As a driver, I would like to have previous destinations saved so that I do not need to guest recall/go through the hassle of reentering the address of my destination.* This use case is a child use case of **(4.7 Visit car park)**. It is responsible for storing car parks successfully visited by the guest.
preconditions	N/A
postconditions	1. Visit must be logged into database
priority	5
frequency of use	Daily
flow of events	1. **Guest** reaches intended car park 2. **Guest** marks trip as complete 3. **MapViewModel** logs visit into **VisitedLocationsDb**
alternative flows	N/A
exceptions	N/A
includes	N/A

special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must ensure that only the final destination is recorded and not the unsuccessful ones.
assumptions	N/A
notes and issues	N/A

4.12

LAUNCH GOOGLE MAPS

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
G R Devansh Rao **last author**

actor	Guest
described by	**F9, F12.1.1.3**
description	*As a driver, I would like to receive directions to get to my destination car park.* This use case is a child use case of **(4.9 Visit car park)**. It will redirect the guest to Google Maps with the intended destination car park, to facilitate faster navigation.
preconditions	1. Can only be entered from **(4.9 Visit car park)**
postconditions	The guest is redirected to Google Maps with the preferred location in the destination box
priority	3
frequency of use	Daily

flow of events	1. **Guest** clicks on the button to redirect to Google Maps for the navigation 2. **Guest** gets redirected to Google Maps with the location in their destination box 3. **Guest** choose their start location and begin navigation to the car park.
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	N/A

4.13

WARN CAR PARK FULL

2023-09-03 **date of creation**
Leong Wei De **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	Server
described by	**F12, N4**
description	*As a server, I would like to inform the system when the car park is full* This use case is called by the server when it detects that the destination car park is no longer vacant.
preconditions	1. The app has an active subscription to monitor the vacancy of the car park server side.
postconditions	1. Message must be sent to the correct device at the end of this use case.
priority	6
frequency of use	Yearly

flow of events	1. **NavigationService** informs **server** that it wants to be kept updated about destination car park () 2. **Guest** starts driving to the destination car park 3. **Server** periodically refreshes the number of vacant lots in the car park 4. **Server** notifies the **NavigationService** if car park is full before the guest reaches 5. **NavigationService** launches **(4.15 Notify car park full)** 6. Subscription cancelled if the guest changes destination car park
alternative flows	N/A
exceptions	N/A
includes	N/A
special requirements	Destination car park may change several times throughout the lifetime of this use case. The system must ensure that it is subscribed to the latest destination car park. Server must achieve an SLA of 0.001% uptime.
assumptions	N/A
notes and issues	The real-time API has a lag time of 3-5 minutes. Multiple subscriptions are allowed but should be cleared within 2-3 hours to reduce load on the server.

4.14

CHANGE CAR PARK

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	System, Guest
described by	**F12.1**
description	*As a driver, when I want to change the car park I am travelling to, I would want the system to change the destination of the car park.* This use case is a child use case of **(4.15 Notify Carpark Full)**, and is called to change the destination of the car park.
preconditions	The new car park has been decided
postconditions	The new car park becomes the destination
priority	7
frequency of use	Often
flow of events	1. **(Entry - 4.15 Notify Car Park Full)** 2. **Guest** accepts the suggested car park 3. **MapViewModel** switches the destination to the suggested car park.

alternative flows	AF1. **(Entry - 4.7 View Car Park Info)** 1. **Guest** previews a different car park from the destination car park 2. **Guest** clicks “Google Maps” button for the new car park **(4.9 Visit Car Park)** 3. **MapViewModel** switches the destination to the suggested car park.
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	The next car park is decided beforehand.
notes and issues	This use case affects the child use cases of **(4.9 Visit car park)**

4.15

NOTIFY CAR PARK FULL

2023-09-03 **date of creation**
G R Devansh Rao **first author**

2023-09-22 **date of revision**
Isaac Chun **last author**

actor	System
described by	**F12.1.1.2**
description	*As a driver that has been informed that the car park I am travelling to is not available, I would like to know the next best alternative and make a decision to change my car park destination.* This use case is done by the system to prepare to inform the driver that the car park is full.
preconditions	The server has notified the system that the car park they are travelling to is full.
postconditions	Notification sent to the guest
priority	7
frequency of use	Often

flow of events	1. **NavigationService** sends a notification to the guest informing them that the car park they are travelling to is full 2. **NavigationService** finds the next most relevant car park 3. The notification will also have the next most relevant car park shown and gives the **Guest** a decision whether they would like to change their car park.
alternative flows	AF1. **Guest** wishes to swap destination 1. **NavigationManager** carries out **(4.14 Change car park)** AF2. **Guest** wishes to keep driving 1. **NavigationService** does not change the destination
exceptions	N/A
includes	N/A
special requirements	N/A
assumptions	N/A
notes and issues	The system uses the stored car park locations from the initial search to the general location as viable car parks. From these, the system will check their availability and return the most relevant car park.

5. Other Nonfunctional Requirements

Usability Requirements

1. Minimal size for buttons and icons should be 48px by 48px.
2. The contrast ratio between the buttons, icons, text and background must be at least 4.5:1.
3. The map should be full-screen.
4. The color scheme of the user interface should be dynamic to the user's phone theme (dark or light mode)

Performance Requirements

1. The system must return the search results within 1 second.
2. The system must display nearby car parks on the map within 1 second.
3. The system must achieve a minimum SLA of 0.001% uptime.
4. The system must acquire carpark information within 1 second.
5. The system must allow the users to save their custom locations within 1 second.

Supportability Requirements

1. Documentation
    a. Comprehensive documentation of the parking app's architecture, components, and APIs should be available
    b. How-to guides and FAQs must be provided for users and be easily accessible for users' reference.
    c. All documentation must be regularly reviewed and updated every 6 months to reflect any changes in the application.
2. Monitoring and Logging
    a. Robust monitoring and logging systems must be implemented to track the app's performance and usage.
    b. Important events, errors, and users actions must be logged for troubleshooting and auditing purposes.
3. User Communication:
    a. Develop a strategy for proactively communicating with users about maintenance windows, updates, and any disruptions in service.
    b. Implement a user-friendly mechanism for users to report issues and provide feedback. This can include:
    c. In-app feedback forms or buttons.
    d. A dedicated support email address.
    e. Social media channels for support and feedback.

Scalability Requirements

1. The system should automatically scale resources up or down based on demand to optimise performance.
    a. Use load testing tools like Apache JMeter, Gatling, or locust.io to simulate a large number of virtual users accessing the car park application concurrently.
    b. Gradually increase the load on the system to mimic real-world scenarios where user traffic fluctuates.
2. Regularly review and update capacity planning based on usage trends and growth projections
    a. Periodically conduct load testing to validate the capacity planning assumptions and projections.
    b. Measure the response times for critical user actions and compare them against the defined requirement (100ms in this case).
3. Ensure that the database system can scale to accommodate growing data volumes and User base.
    a. Consider configuring auto-scaling for your database infrastructure. Cloud platforms like AWS, Azure, and Google Cloud offer auto-scaling capabilities based on usage metrics.
    b. Consider implementing caching mechanisms (e.g., Redis or Memcached) to reduce the load on the database for frequently accessed data.

Software Quality Attributes

The application was built with the purpose of providing users a good user experience and allowing easy access to their data. The architecture of **Park in Peace** should be as loosely coupled as possible, and utilise as much of good principles as possible. **Park in Peace** realizes this by ensuring that there are tests for relevant parts of the program by doing unit testing, as well as black and white box testing on key functions of the application.

The following summarises some keypoints of the applications positive attributes:

- **Usability**: The application is easy to navigate, making it easy for new users or testers without technical knowledge to use our application, knowing exactly what they should do.
- **Reliability**: In the event that API fails, the application has a fallback that allows the application to still be functional, as well as ensuring consistent performance and data accuracy
- **Flexibility**: The application is designed to be as loosely coupled as possible, such that code changes in a particular part of the program would not need changes in another part of the program, ensuring safer and easier changes of code.
- **Scalability**: The application does its work mostly through interfaces and dependency injection, allowing future modules to be added with minimal work.

- **Robustness**: The application minimizes the risk of unexpected and invalid inputs without compromising the overall functionality.
- **Maintainability:** Since the application uses RestAPI, the client and server can be maintained and developed in parallel without causing any disruptions to services.

Business Rules

There are no business rules regarding this application as there are no social interactions in this application. The only functions a user is allowed to do is to search for locations and view the necessary carpark details. The data returned by the application has no risk of criminal intent.

Also, the data provided by the application provides little to no risk of any personal information as there are no sign ups required, and merely serves as a purpose of guiding any user with as little hassle as possible.

6. Other Requirements

The current state of **Park in Peace** is not to be deployed as it belongs to the intellectual property of Nanyang Technological University. Without the authorisation of relevant stakeholders, the development of **Park in Peace** shall be kept discreet until authorised to be disclosed.

The features in **Park in Peace** can also be reused and developed upon by future developers as long as they abide to the rules set by the organisation.

Appendix A: Glossary

This section defines all the terms, acronyms and abbreviations necessary to properly interpret the SRS.

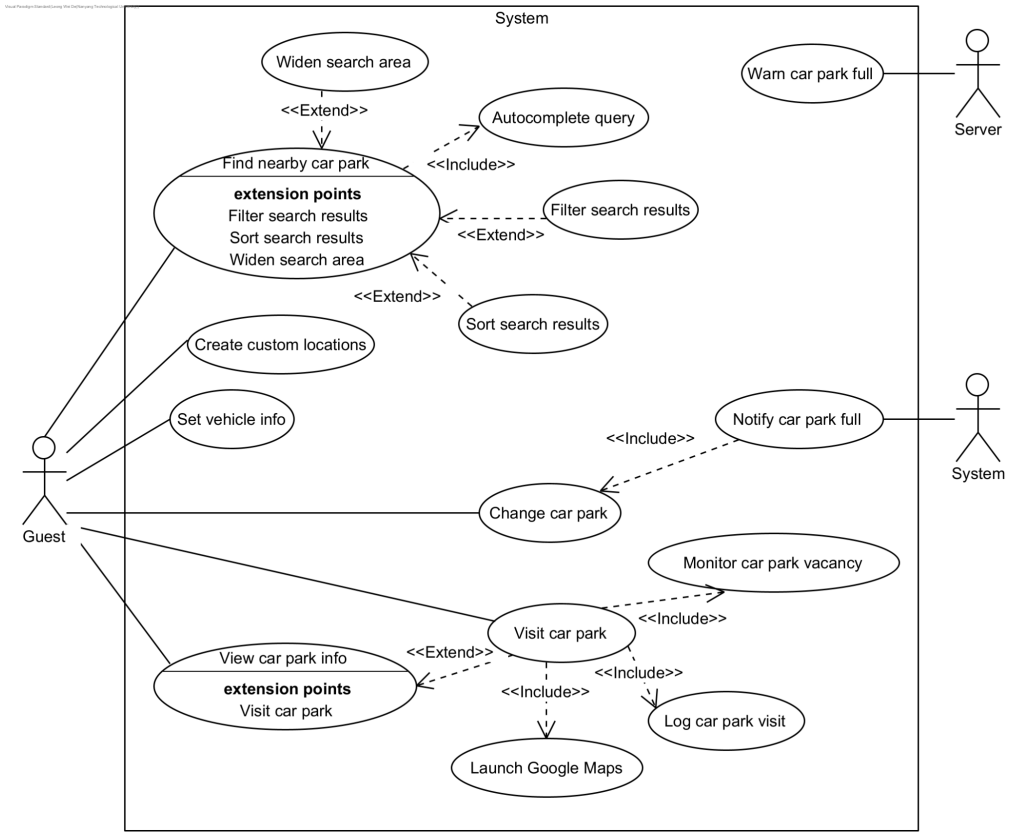
Data Dictionary

TERM	DEFINITION
availability/ vacancy	Indicates whether the car park is currently operating or closed and number of available slots left per vehicle type.
API	Application Programming Interface
car park	A designated area or facility where motor vehicles, such as cars and motorcycles can be temporarily parked or stored. Car parks often charge a fee and have limited vacancy for vehicles. Should a user arrive when a car park is full, they have no choice but to find a different location to park. For the purposes of our system, only designated car parks recognised by URA will be in scope.
car park features	Additional features present in the car park, such as electric vehicle charging lots and vehicle washing bays.
destination	The intended location the user wishes to go to.
destination car park	The intended car park the user wishes to go to.
drop pin	The virtual marker that is placed on a map to denote a specific location of interest. By tapping or clicking on the map at the desired Destination, the pin is dropped, serving as a reference point for the app and user and providing relevant information and directions to that specific destination.
fuel type	The fuel that powers the vehicle, such as gasoline or electric hybrid.
guest	An unregistered user of the application. Unless otherwise stated, a guest may access all features of the application.
map	A graphical representation of geographic data or information within the application. The map provides a visual interface for users to view and interact with.
preset search radius	A zone around the intended destination of the user extending up to 500m. The system will consider car parks

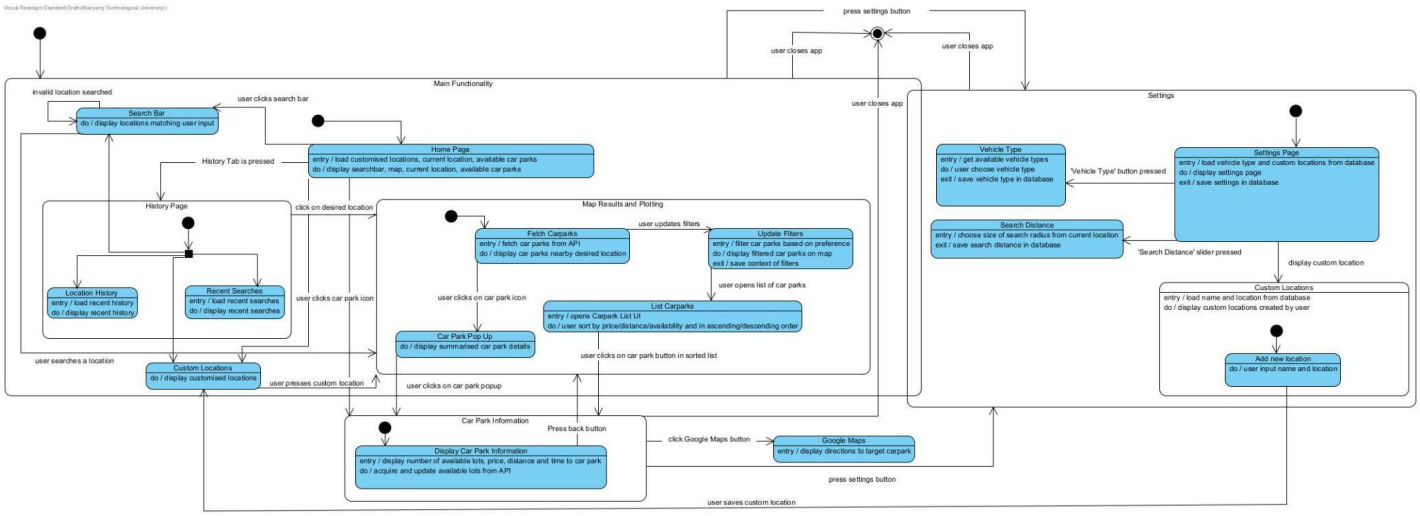
	within this zone as “nearby” unless otherwise specified. The preset search radius may be extended in certain cases.
price	The cost or fee that a person is required to pay for parking their vehicle in a car park. Car park fares vary depending on factors such as the location, duration of parking, and the policies of the specific car park or parking facility. Usually measured in dollars per hour.
SLA	Service Level Agreement - a contract between an end-user and company that outlines minimum expected service requirements, including quality, availability, and timeliness.
user	A person that is presently accessing the system.
vehicle type	The class of the vehicle such as car, motorcycle or heavy vehicle.
UI	User interface, represents the visuals a user sees when interacting with our application
SDK	A software development kit (SDK) is a set of platform-specific building tools for developers.
REST	A software architectural style that was created to guide the design and development of the architecture for the World Wide Web

Appendix B: Analysis Models

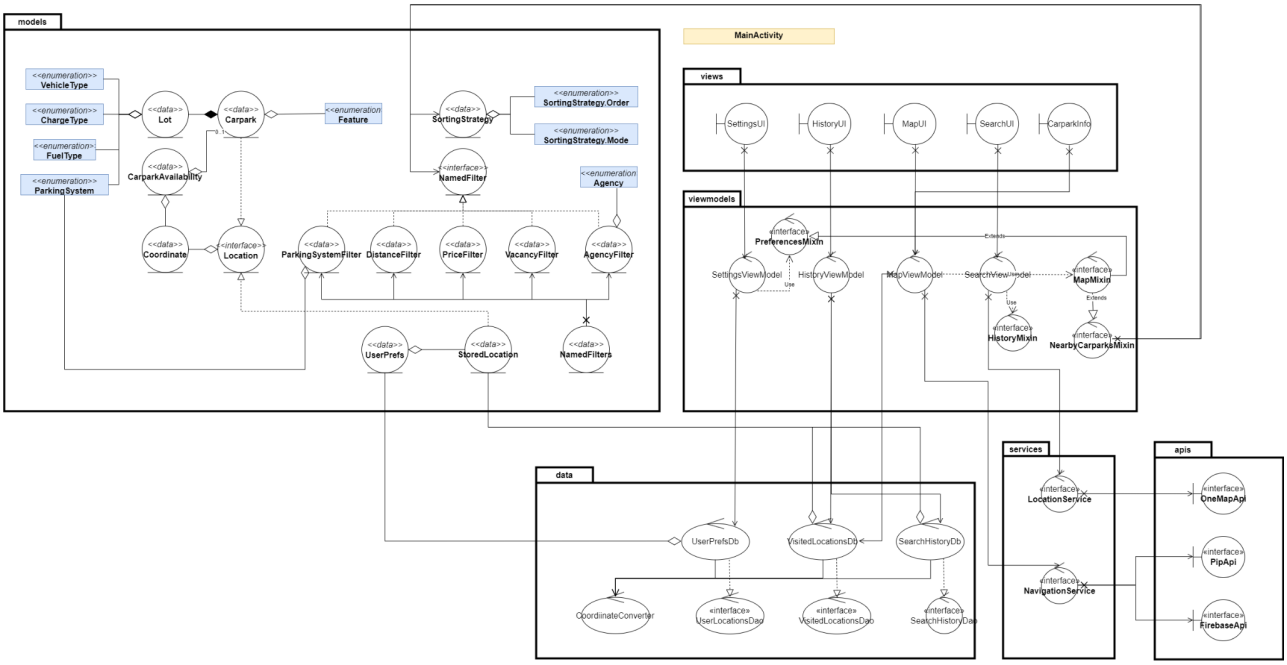
- **Use Case Diagram**



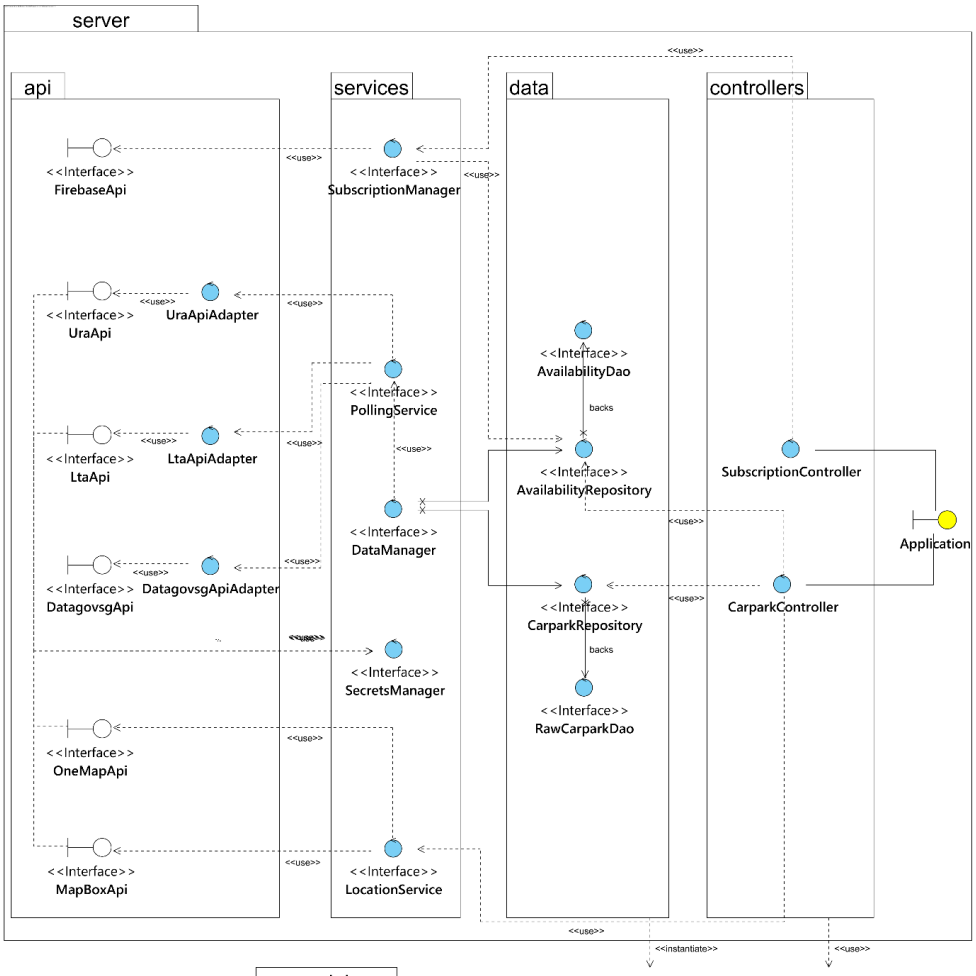
- **Dialog Map (State Machine Diagram)**

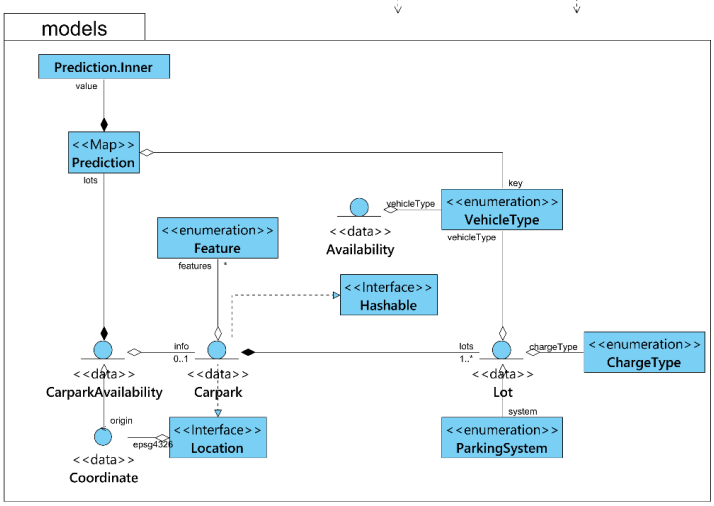


- **UML diagrams - conceptual**
  - **Client**



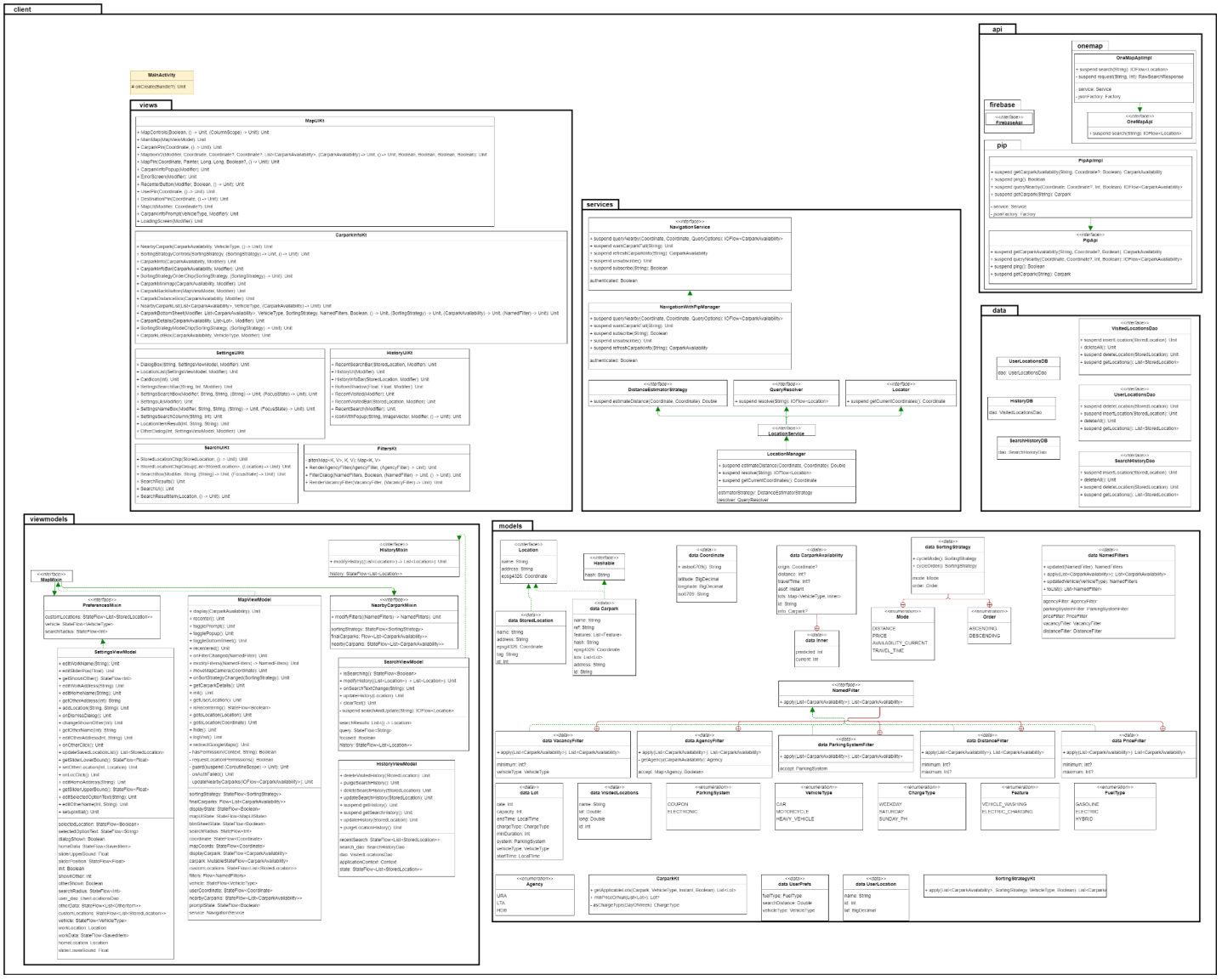
- **Server**


server
api
services
data
controllers
<<Interface>>
FirebaseApi
<<Interface>>
SubscriptionManager
<<Interface>>
UraApi
UraApiAdapter
<<Interface>>
PollingService
<<Interface>>
LtaApi
LtaApiAdapter
<<Interface>>
DataManager
<<Interface>>
DatagovsgApi
DatagovsgApiAdapter
<<Interface>>
SecretsManager
<<Interface>>
OneMapApi
<<Interface>>
MapBoxApi
<<Interface>>
LocationService
<<Interface>>
AvailabilityDao
backs
<<Interface>>
AvailabilityRepository
<<Interface>>
CarparkRepository
backs
<<Interface>>
RawCarparkDao
SubscriptionController
CarparkController
Application
<<use>>
<<instantiate>>

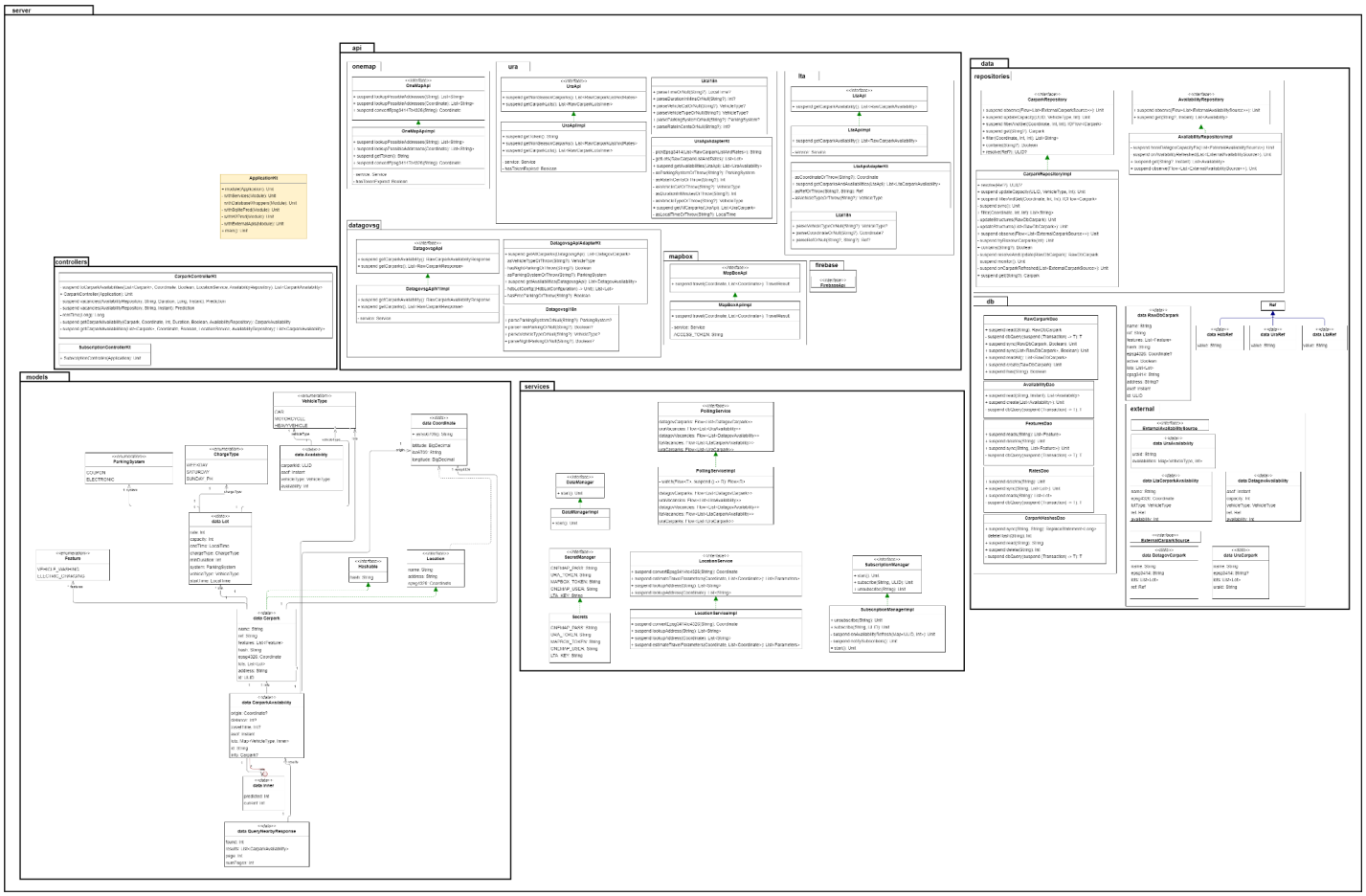

models
Prediction.Inner
value
<<Map>>
Prediction
lots
<<enumeration>>
Feature
features
<<data>>
Availability
vehicleType
<<enumeration>>
VehicleType
key
vehicleType
<<Interface>>
Hashable
<<data>>
CarparkAvailability
info
0..1
<<data>>
Carpark
lots
1..*
<<data>>
Lot
chargeType
<<enumeration>>
ChargeType
origin
epsg4326
<<Interface>>
Location
<<data>>
Coordinate
system
<<enumeration>>
ParkingSystem

- **UML Diagram - Design**

  - **Client**

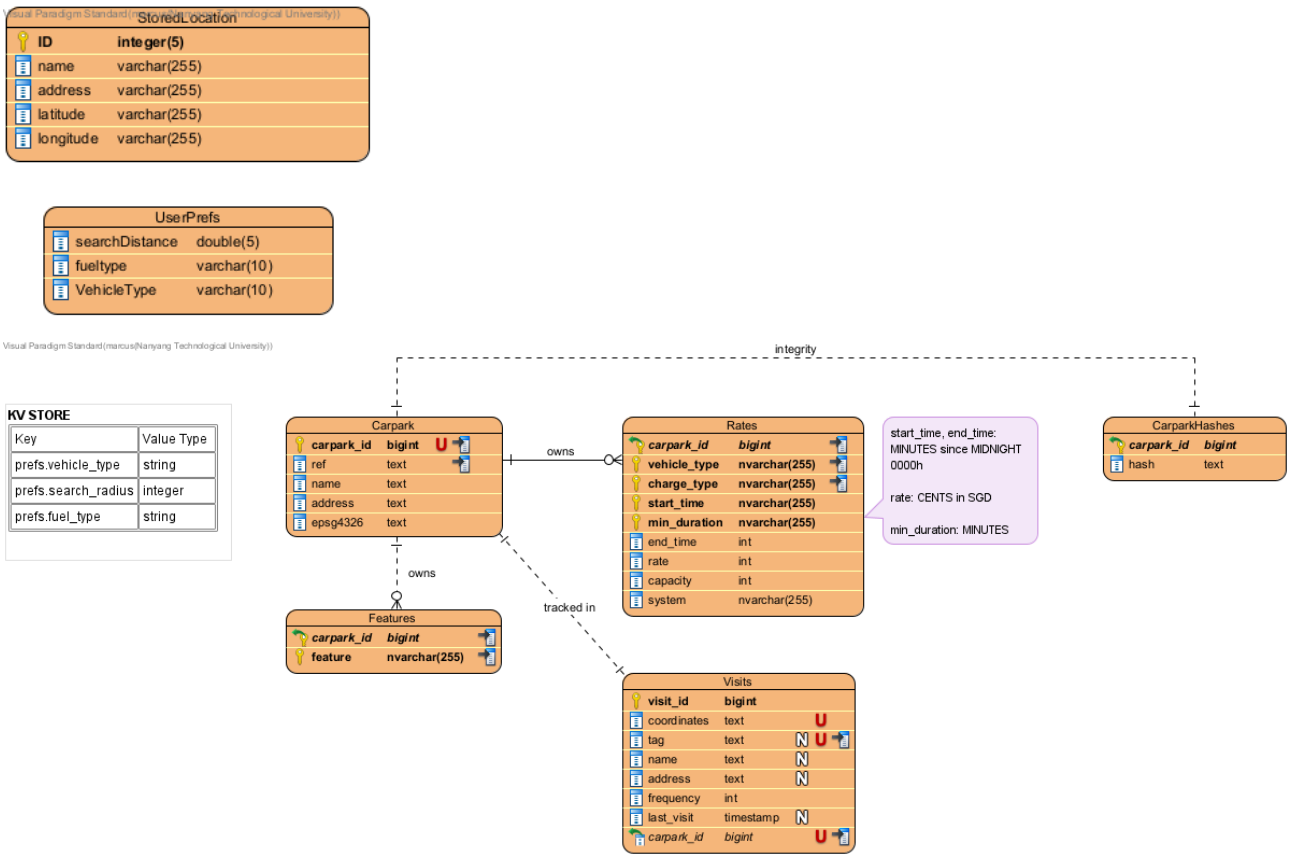


- **Server**

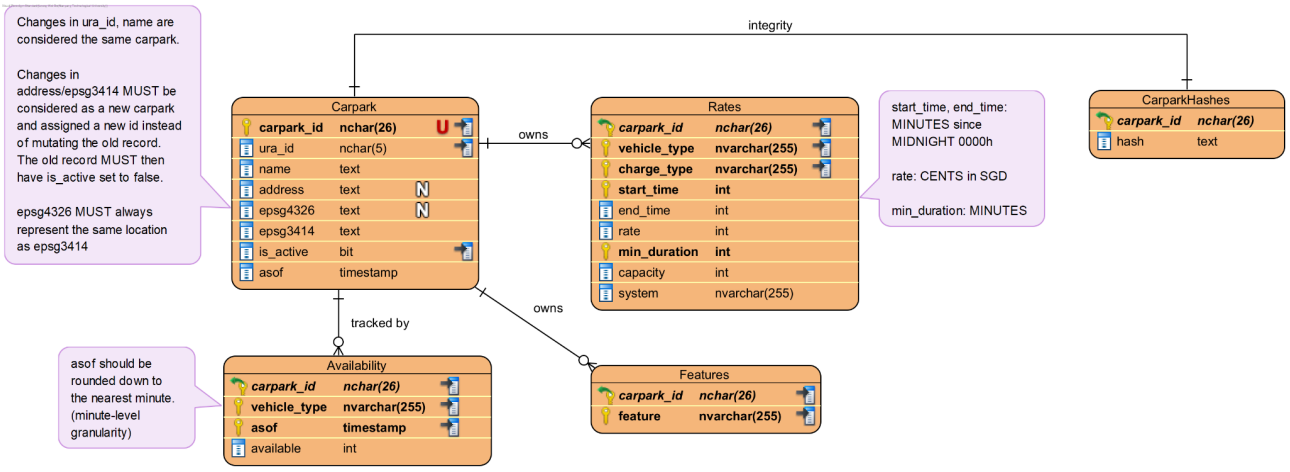

server
api
onemap
ura
lta
datagovsg
mapbox
firebase
controllers
models
services
data
repositories
db
external

- **Entity-Relationship Diagram**

    - Client

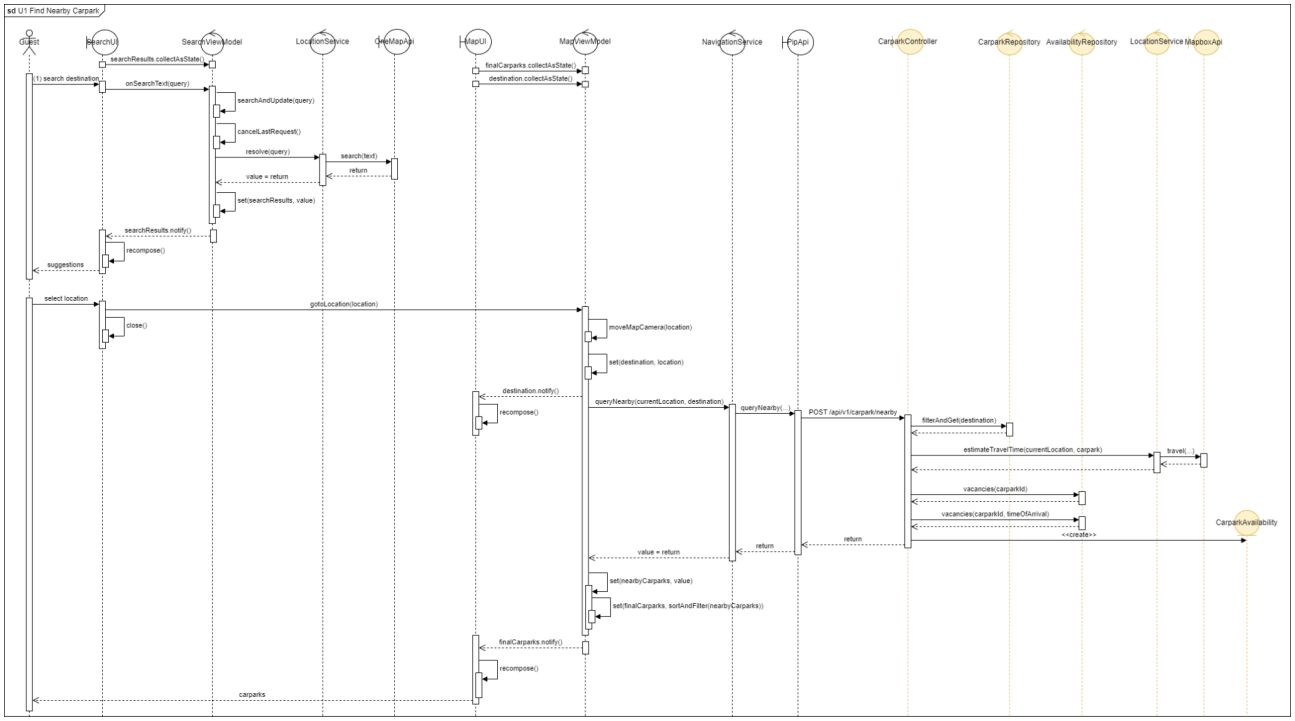


- Server

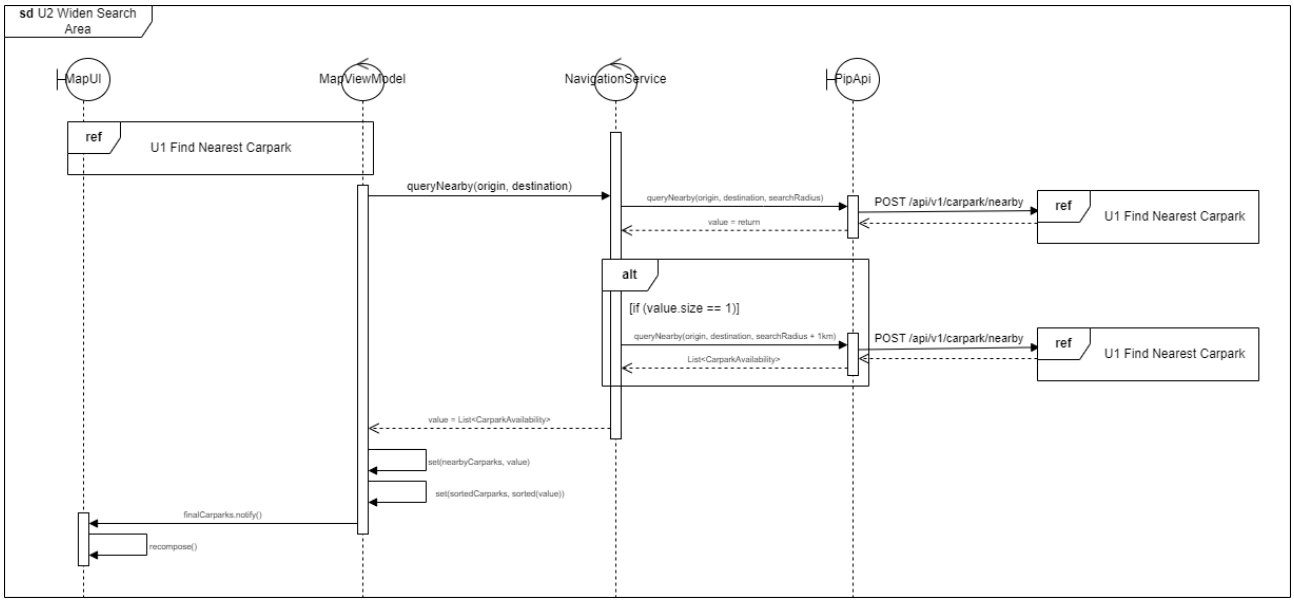


- **Sequence diagrams**

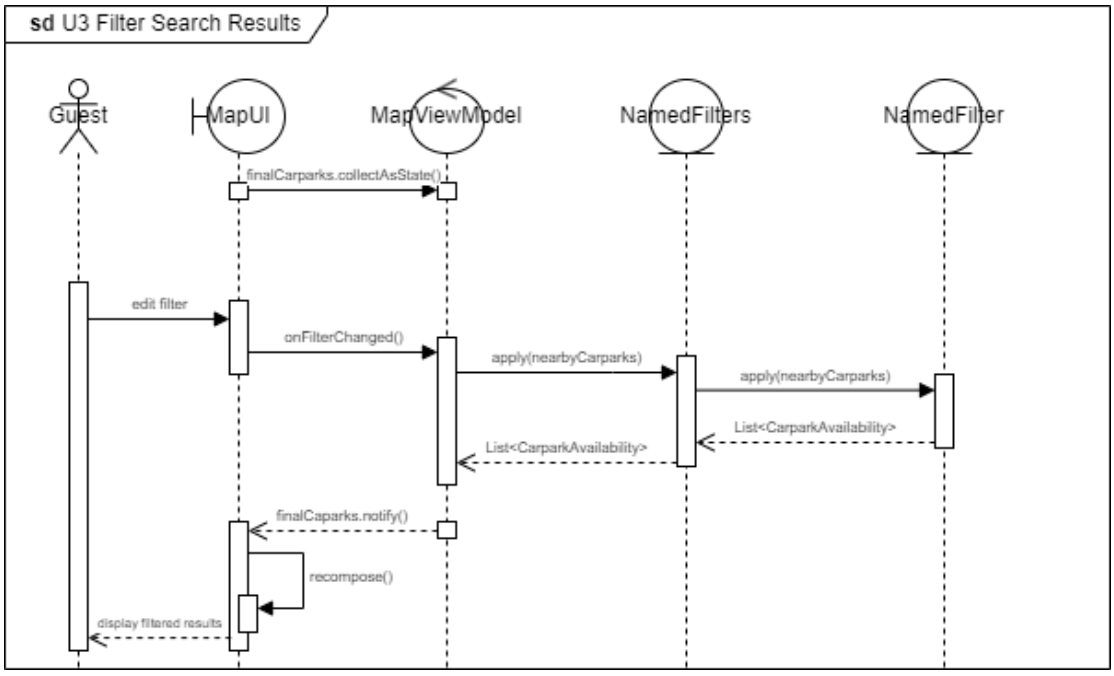
- U1 Find Nearby Car Park



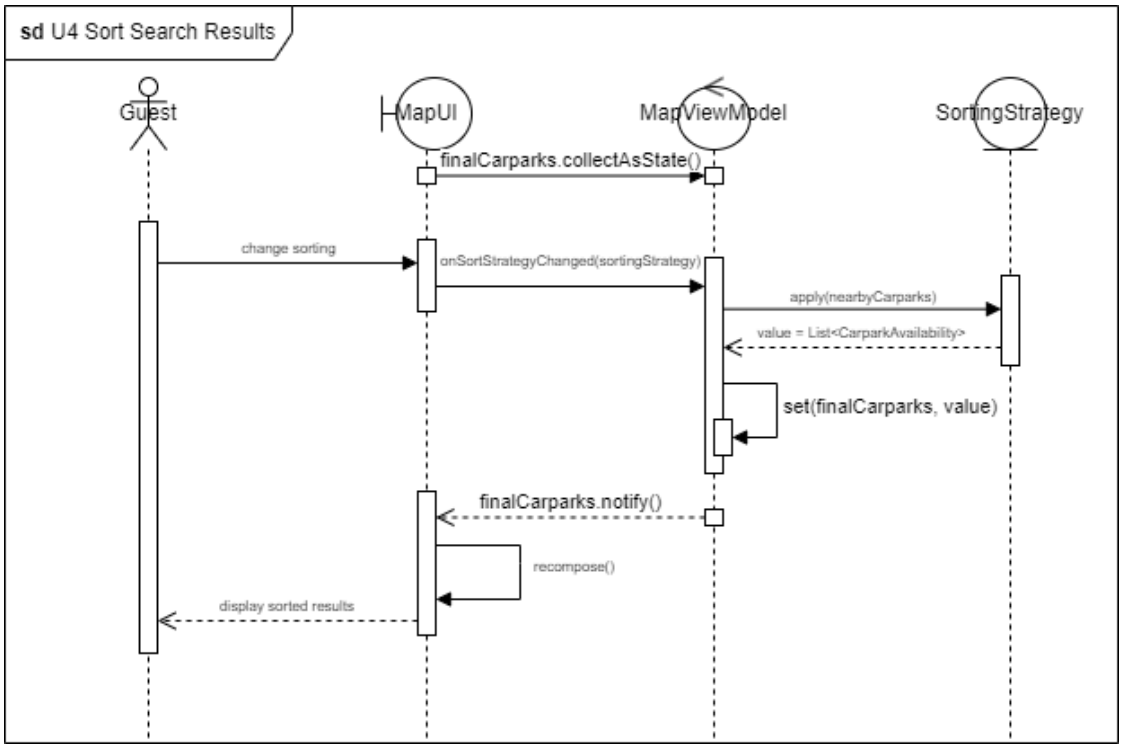
- U2 Widen Search Area



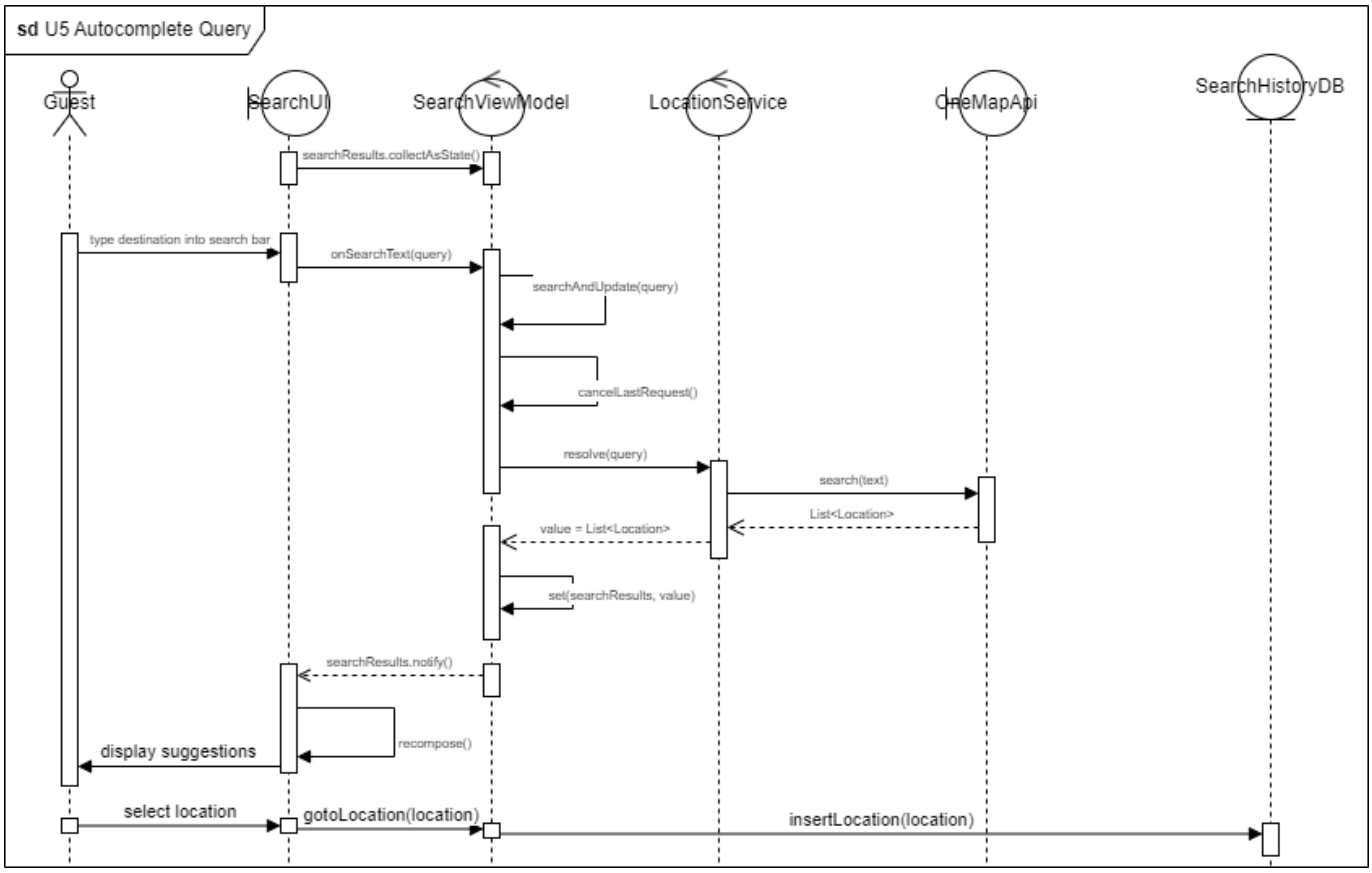
- U3 Filter Search Results



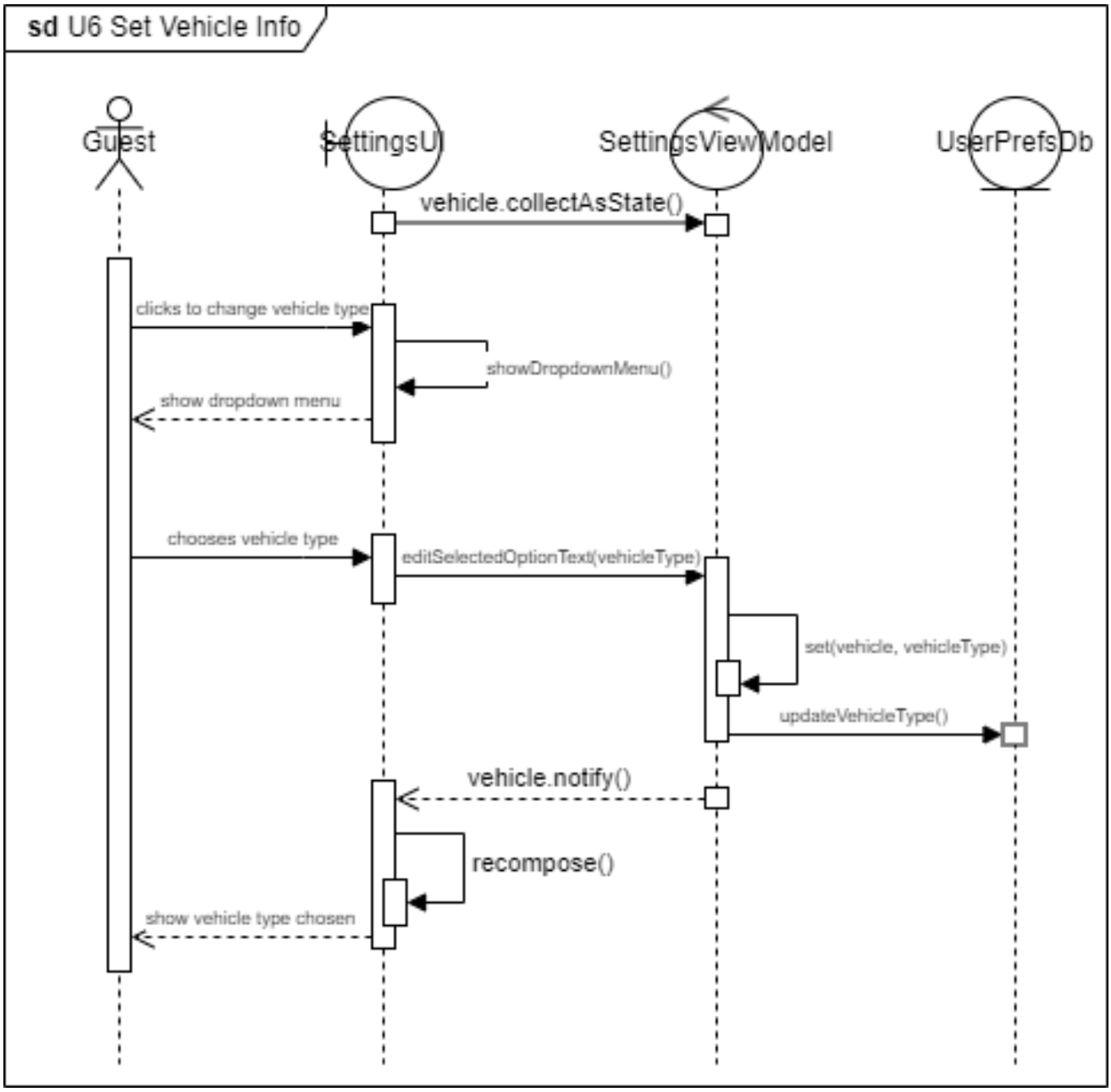
- U4 Sort Search Results



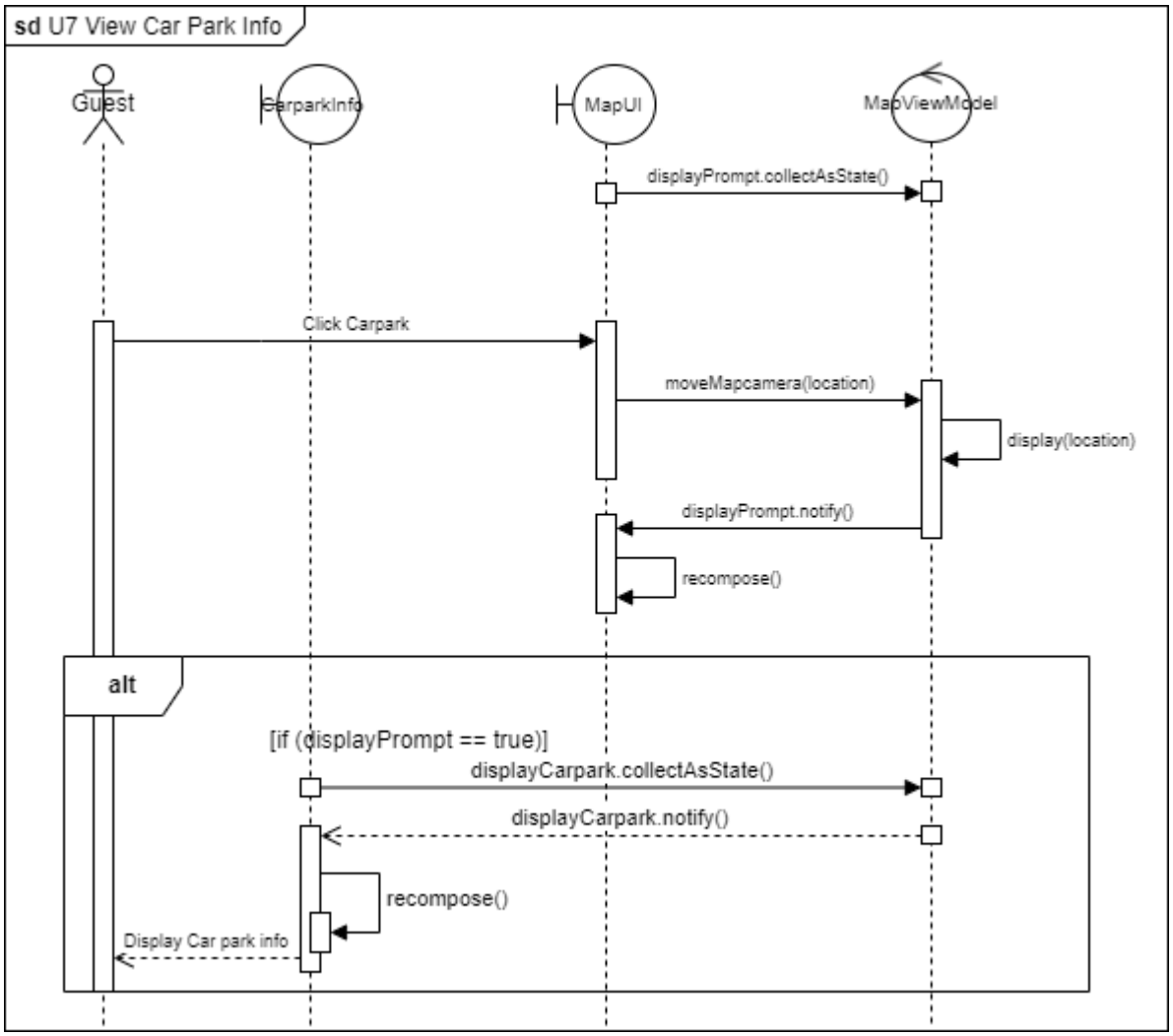
- U5 Autocomplete Query



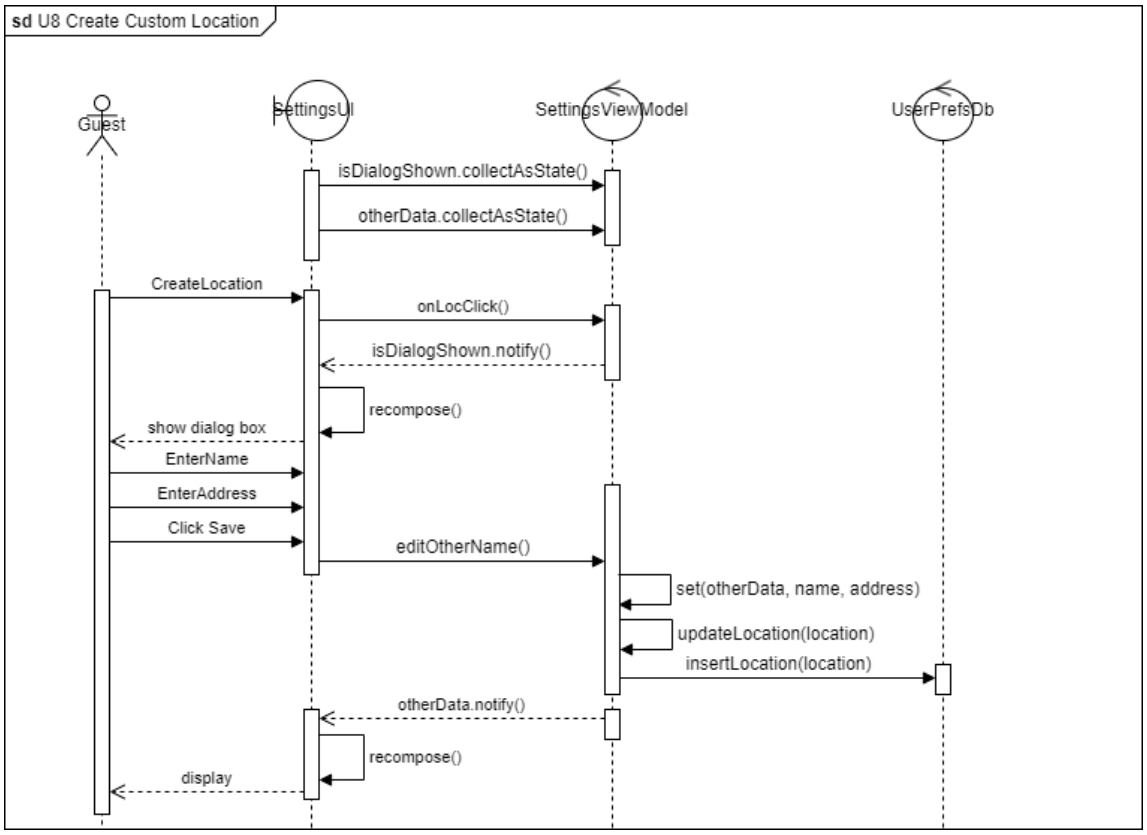
- U6 Set Vehicle Info



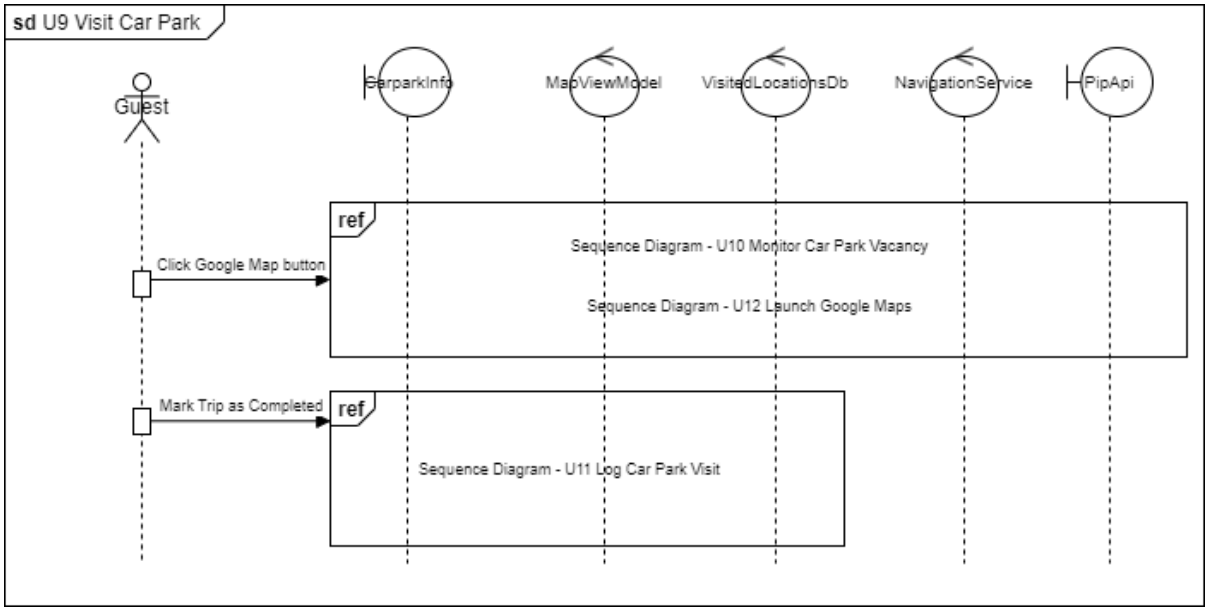
- U7 View Car park Info



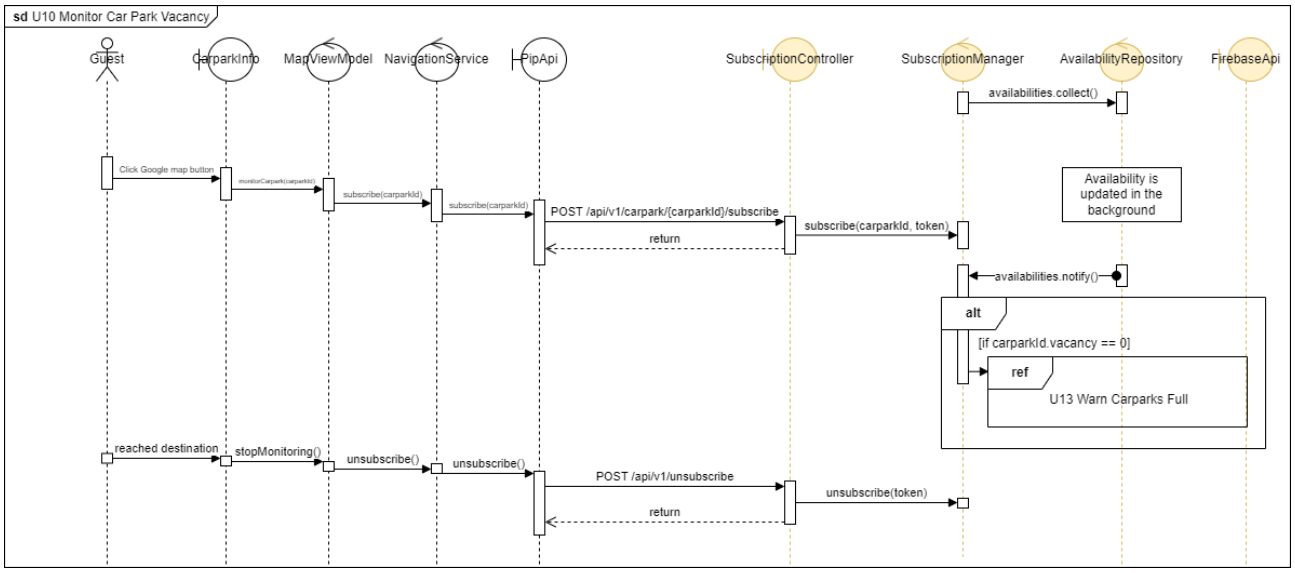
- U8 Create Custom Locations



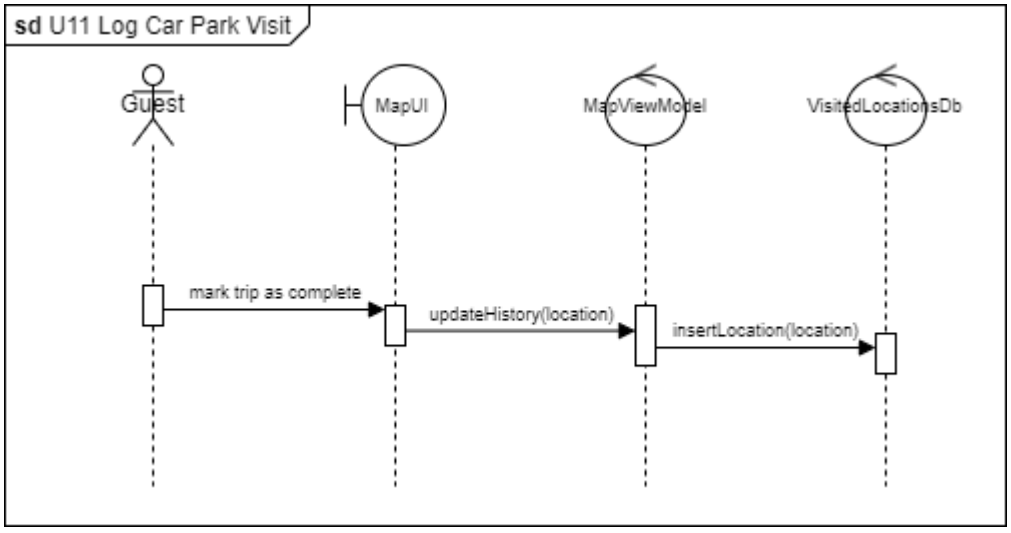
- U9 Visit Car park



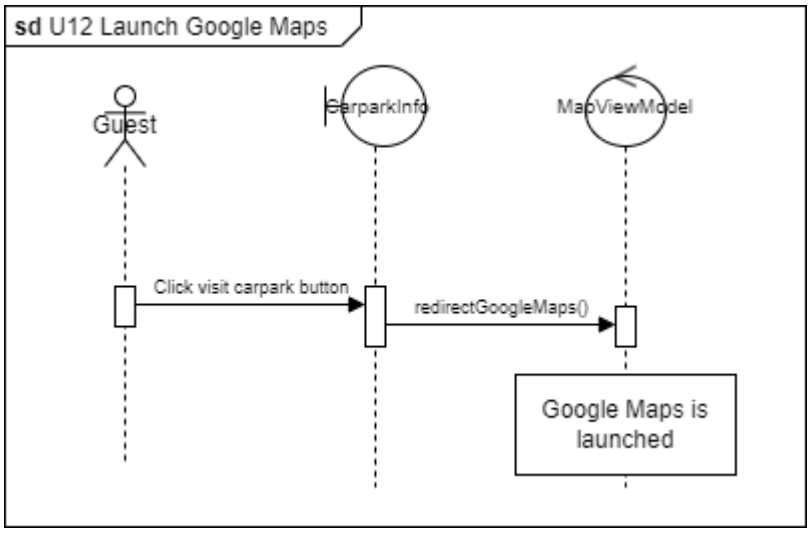
- U10 Monitor Car Park Vacancy



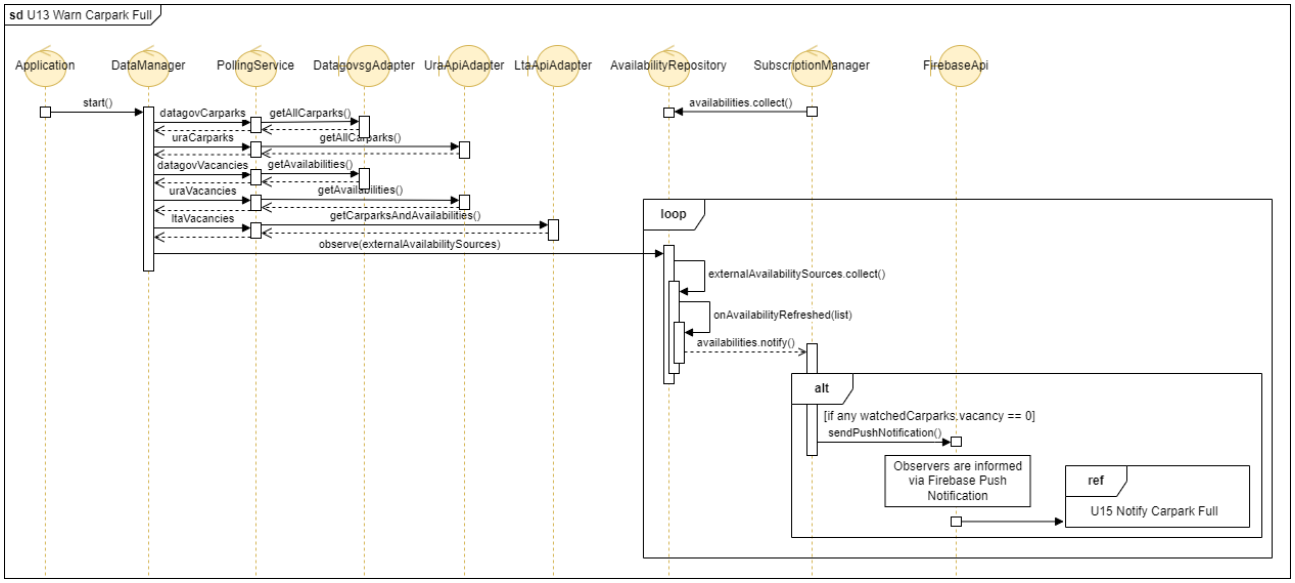
- U11 Log Car Park Visit



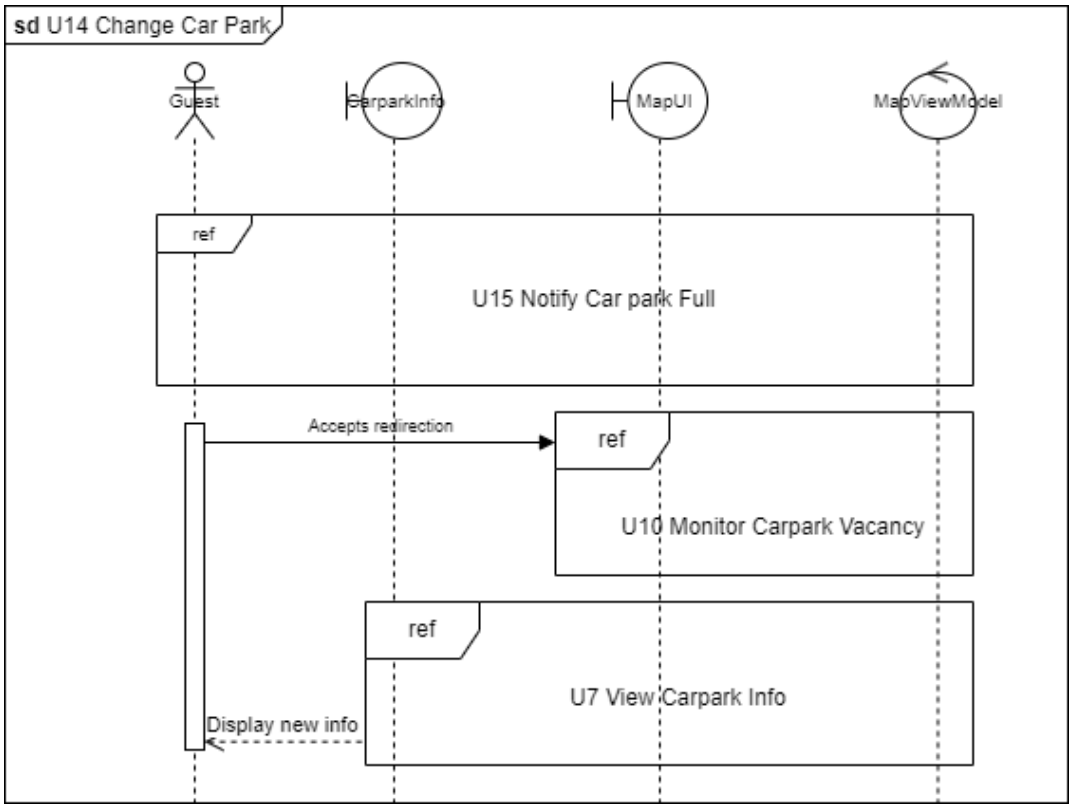
- U12 Launch Google Maps



- U13 Warn Car Park Full



- U14 Change Car Park



- U15 Notify Car Park Full