**Software Requirements Specification**

**for**

**ParkProximity**

**Version 1.6 approved**

**Prepared by**

|  |  |
| --- | --- |
| **Ooi Chee Han** | **U2222143G** |
| **Ang Ming Ze, Aaron** | **U2122736A** |
| **Arjun Kumaresh** | **U2221225J** |
| **Choh Lit Han Owen** | **U2221094K** |
| **Joel Chng Ming Xuan** | **U2121344D** |
| **Michael Santoso** | **U2221525L** |
| **Parashar Pranav** | **U2222548E** |

**Syntax Symphony**

**8 September 2023**

**Table of Contents**

[**1. Introduction 1**](#_heading=h.1fob9te)

[1.1 Purpose 1](#_heading=h.3znysh7)

[1.2 Document Conventions 1](#_heading=)

[1.3 Intended Audience and Reading Suggestions 2](#_heading=)

[1.4 Product Scope 3](#_heading=)

[1.5 References 3](#_heading=)

[**2. Overall Description 3**](#_heading=)

[2.1 Product Perspective 3](#_heading=h.2s8eyo1)

[2.2 Product Functions 4](#_heading=)

[2.3 User Classes and Characteristics 4](#_heading=)

[2.4 Operating Environment 4](#_heading=)

[2.5 Design and Implementation Constraints 4](#_heading=)

[2.6 User Documentation 4](#_heading=)

[2.7 Assumptions and Dependencies 5](#_heading=)

[**3. External Interface Requirements 5**](#_heading=)

[3.1 User Interfaces 5](#_heading=h.2jxsxqh)

[3.2 Hardware Interfaces 5](#_heading=)

[3.3 Software Interfaces 5](#_heading=)

[3.4 Communications Interfaces 5](#_heading=)

[**4. System Features 6**](#_heading=)

[4.1 Location Services 6](#_heading=)

[4.2 Carpark Data 7](#_heading=)

[4.3 Search and Filter 7](#_heading=h.8r2c4iocdm)

[4.4 Map View 7](#_heading=h.tkqb13jvdv9g)

[4.5 User Specific Features 8](#_heading=h.4vuyf6p88lsv)

[**5. Other Nonfunctional Requirements 8**](#_heading=h.3whwml4)

[5.1 Performance Requirements 8](#_heading=h.2bn6wsx)

[5.2 Safety Requirements 8](#_heading=)

[5.3 Security Requirements 8](#_heading=)

[5.4 Software Quality Attributes 9](#_heading=)

[5.5 Business Rules 9](#_heading=)

[**6. Other Requirements 10**](#_heading=)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Michael, Owen, Joel | 14/09/23 | Moving Lab 1 Deliverables to SRS template | 1.0 |
| Chee Han | 16/09/23 | Adding non-functional requirements | 1.1 |
| Chee Han, Michael, Aaron | 21/09/23 | Adding functional requirements, use case diagram | 1.2 |
| Aaron, Michael | 11/11/23 | Update on system features and non-functional requirements | 1.3 |
| Arjun | 11/11/23 | Update on the introduction and overall description | 1.4 |
| Chee Han | 11/11/23 | Inclusion of Dialogue map, Class diagram, System architecture, White box testing, and ER diagram  Inclusion of user interfaces  Update of use case diagram | 1.5 |
| Owen | 11/11/23 | Added Dependencies to “Assumptions and Dependencies”  Updated White Box Testing for createFavourite | 1.6 |

# Introduction

## Purpose

Given the recent congestion at car parks in Singapore, a user-friendly car-parking app that displays the availability of nearby car parks becomes essential. ParkProximity is a mobile application that displays the availability of car parks near the user and provides directions to the selected car park. Additional features include navigation to the nearby car park, pinpointing parked car locations, retrieving a record of the user’s parking history, and saving favourites.

## Document Conventions

This Software Requirement Specification (SRS) follows the following conventions:

1. Font and Typography:
   1. Text is presented in Times New Roman font, size 11.
   2. Headings are in bold.
2. Text Formatting:
   1. Italic text is used for citations and references.
3. Highlighting:
   1. Critical requirements are highlighted in bold and red.
4. Priority Notation:
   1. Priorities for every requirement statement are to have their own priority.
5. Numbering or Labeling:
   1. Sections are numbered in the format Section X.X.
6. Terminology
   1. Specific terminology is defined in Appendix A (Glossary).
7. References:
   1. External references are cited following the APA citation style in Section 1.5.
8. Revision History:
   1. This document follows version 1.0 and was last revised on 12/11/2023
   2. Changes to this document are tracked in the revision history section.
9. Additional Notes:
   1. Tables, diagrams, and charts are labeled and formatted following the IEEE standards.

## Intended Audience and Reading Suggestions

This Software Requirements Specification (SRS) is intended for various stakeholders involved in the project. The document is structured to cater to the specific needs of the following audience types:

1. **Developers**:
   1. Developers will primarily refer to this SRS to gain a high-level understanding of the project's requirements, including functional and technical specifications
2. **Project Managers:**
   1. Project Managers will use this document to assess project scope, timeline, and resource allocation. They should start with the Project Overview and Scope sections.
3. **Marketing Staff:**
   1. Marketing staff may find value in understanding the key features and user experience described in this document. They are encouraged to focus on the User Requirements and User Interface sections.
4. **Users**:
   1. End-users will benefit from a basic understanding of the system's capabilities and limitations. They should begin with the Introduction and User Requirements sections.
5. **Testers**
   1. Testers will rely on this document to create test cases and validate the system against requirements. They should start with the Functional Requirements and Testability sections.
6. **Documentation Writers**:
   1. Documentation writers should carefully review the User Documentation Requirements section to gather information for user manuals and help documentation.

**Document Contents:**

This SRS contains information related to project scope, functional and non-functional requirements, system constraints, assumptions, and other critical project details.

**Organizational Overview:**

The document is organized into the following sections: Introduction, Project Overview, Scope, Requirements, Constraints, Assumptions, and Appendices.

**Suggested Sequence for Reading:**

* Start with the Introduction to get an overall understanding.
* For a broad view of the project, proceed to the Project Overview and Scope sections.
* Developers should delve into the Requirements section, while testers can focus on the Functional Requirements and Testability sections.
* Marketing staff and users may find the User Requirements and User Interface sections most informative.
* Project Managers can gain insights from the Project Overview and Constraints sections.
* Documentation writers should thoroughly review the User Documentation Requirements section for content creation.

Tailor your reading sequence based on your specific role and objectives to extract the most relevant information from this document.

## Product Scope

The app will make use of government and public APIs to achieve the purpose laid out in 1.1.

The APIs are provided by Land Transport Authority and Google Maps.

The software framework we will use to develop the mobile app is Springboot Framework.

Springboot was chosen as it allows us to easily publish the app to Android and IOS users.

The database used for the project is in PostgreSQL.

The framework chosen for the frontend was the Flutter Framework which uses the dart language. Flutter was chosen as it allows for cross-platform development(the app can be viewed across different android devices) and the app will

## References

* + 1. APIs
       1. Carpark APIs

* + - * 1. [API from data.gov.sg](https://beta.data.gov.sg/datasets/85/view)

* + - * 1. [Land Transport Authority API](https://datamall.lta.gov.sg/content/datamall/en/dynamic-data.html)

* + - * 1. [Google Map API](https://developers.google.com/maps/documentation)

* + 1. [Flutter](https://flutter.dev/)

* + 1. [Spring Boot with Gradle](https://spring.io/projects/spring-boot)

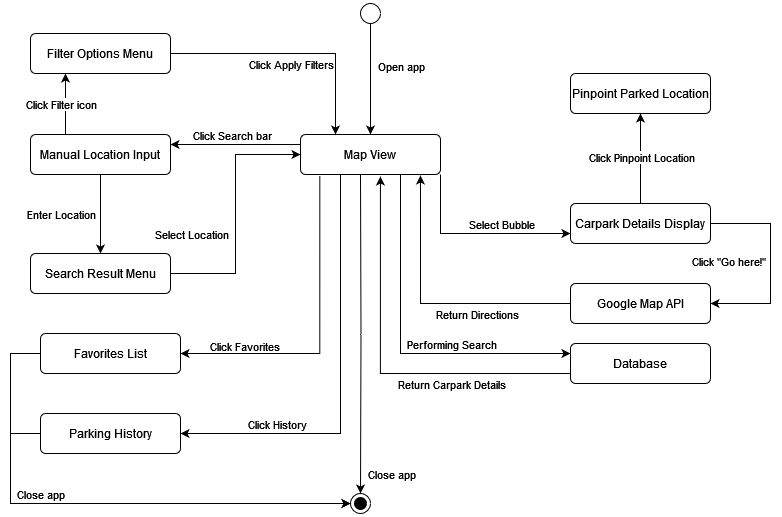
* + 1. [PostgreSQL](https://www.postgresql.org/)

# Overall Description

## Product Perspective

This product is to help users make use of the publicly available datasets from data.gov.sg and Land Transport Authority APIs. The product adds to these datasets by providing search functionality and provides directions using the Google Maps API.

The image below shows the dialogue map for our app.



## Product Functions

* + 1. Search and filter function for users to search for carparks using their current location or manual input and filter according to distance, remaining lots, and carpark price.
    2. The product must allow users to obtain directions from the current location to the selected carpark.
    3. The product must allow users to enter details to save their car location to help them find their car.
    4. The product must have a favourite function to save the selected carpark for them to access easily later.

## User Classes and Characteristics

The normal users will only be able to affect their local copy of the Android application. No level of technical expertise is required for these users.

The developers will not have a different interface as compared to the normal users when using the Android application. They should be the only ones with direct access to the backend server where they will be able to access the database as well.

## Operating Environment

The front end should be operated in an Android system.

The back-end server will be operating on a Linux system, specifically Ubuntu.

## Design and Implementation Constraints

The back-end server is implemented with libraries specifically for PostgreSQL as the database management system.

The back-end is organized based on the entity classes (e.g. users, favourites, parked) and each entity should have its repository and controller. The service class used for the data access can be combined up to the developer’s preferences.

The front end is implemented with the Flutter framework.

## User Documentation

There is a YouTube video showcasing how our software will work. The link to that video can be found [here](https://www.youtube.com/watch?v=N9LZaZcG9Xw).

## Assumptions and Dependencies

Dependency:

1. Carpark API provider:
   1. Dependency: The app relies on the government's carpark availability data provided through their official API.
   2. Project Impact: Any changes, outages, or discontinuation of this API by the government may disrupt the app's carpark availability information service.

# External Interface Requirements

## User Interfaces

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *1. Initial Page* | *2. Carpark Details* | *3. Favourites Page* |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *4. History Page* | *5. Search* | *6. Filter Preferences* |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *7. Directions* | *8. Pinpoint Parked Location* | *9. Side Menu* |

## Software Interfaces

1. Spring Boot Framework (Version 3.1.4) with Gradle (Version 8.2.1)
2. PostgreSQL Database (Version 16)
3. Ubuntu (Version 23.10)
4. Flutter (Version 3.13.7)

## Communications Interfaces

The main form of communication between the front and backend is via HTTP requests and responses.

Methods used are GET, POST, PUT, and DELETE.

Both request and response message formats are in JSON.

# System Features

Below is our user case diagram of the carpark application:

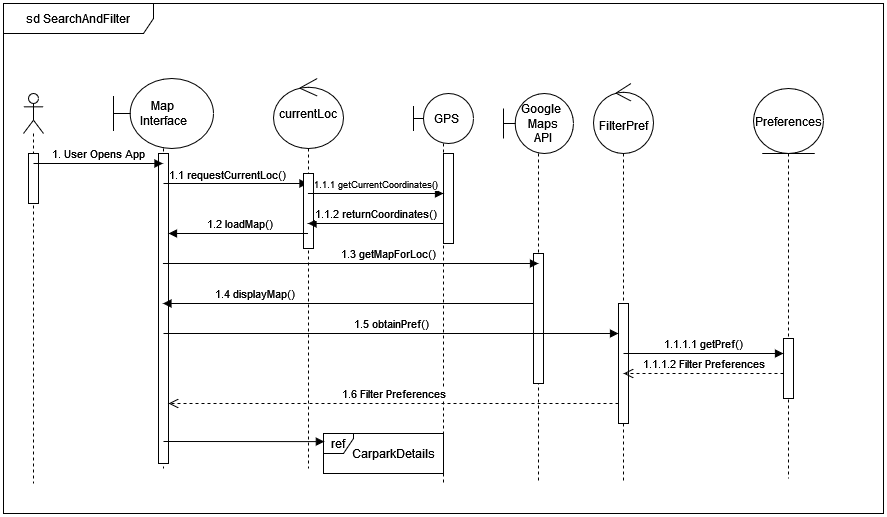
**

## Search & Filter

4.1.1 Description and Priority

The use case involves searching for carparks based on location, distance, and availability of car park slots. This feature has a high priority.

4.1.2 Stimulus/Response Sequences

**

4.1.3 Functional Requirements

REQ-1: Users must be able to search for carparks based on location, distance, and availability

REQ-2: Users should be able to set a preferred search radius (e.g. 1 km, 2km, 5 km) for nearby carparks

REQ-3: Users should be able to set carpark availability in terms of fully occupied, semi occupied and/or available

REQ-4: The app must filter and display carparks that are located within the user's chosen radius

REQ-5: Carparks outside selected radius must be excluded from the results

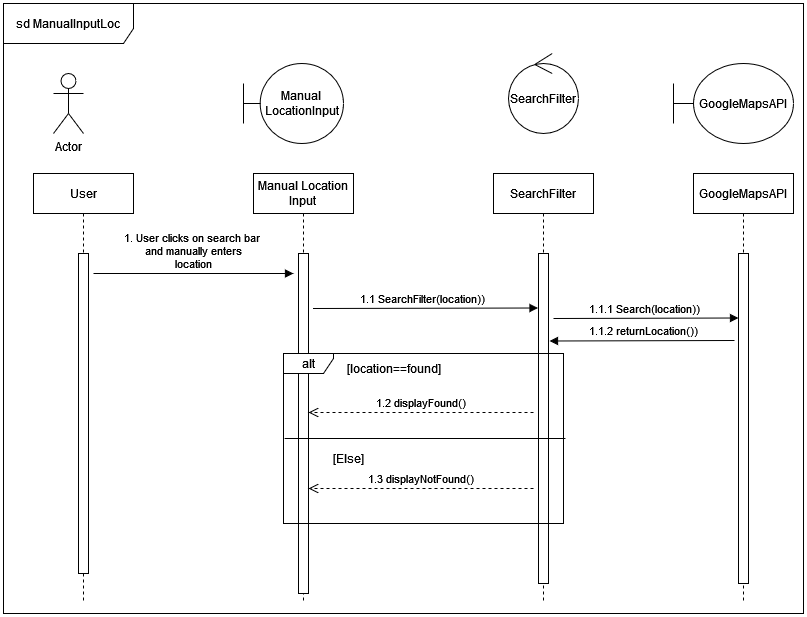
REQ-6: Carparks with no more capacity can also be excluded

## Manual Location Input

* + 1. Description and Priority

This use case allows the user to manually input a location either through an address, postal code or name of a specific car park. This feature has a medium priority.

* + 1. Stimulus/Response Sequence

**

* + 1. Functional Requirements

REQ-1: Carpark data must include names, addresses, capacity, available slots, distance from users, and opening hours

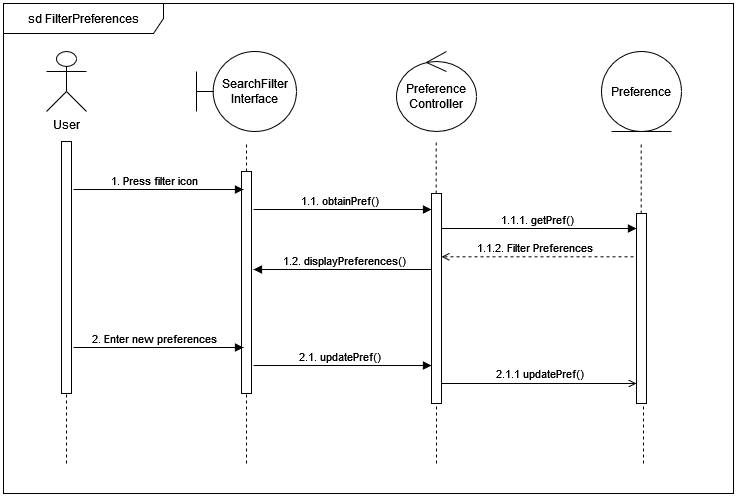
## 

## Filter Preferences

* + 1. Description and Priority

Users can set a preferred search radius (e.g. 1km, 5km, etc) for nearby carparks and in terms of the carpark’s slots availability. This feature has a medium priority.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

REQ-1: Users must be able to set a preferred search radius for nearby carparks from a range of options given to them in a selection.

REQ-2: Users must be able to set a preferred carpark availability filter for nearby carparks from a range of options given to them in a selection list.

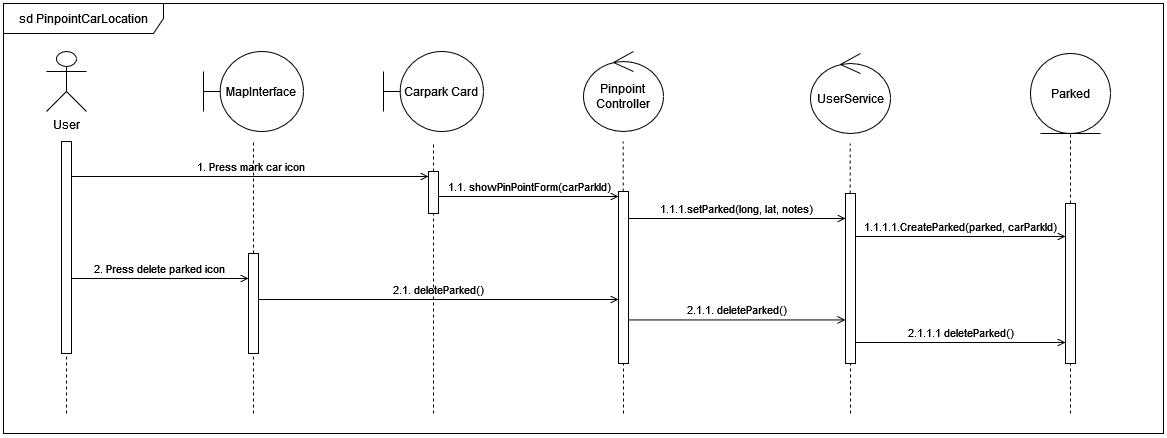
## Pinpoint Car Location on Map

* + 1. Description and Priority

Users can pinpoint their car’s location on the map or unmark it after leaving the car park.

This feature has a low priority.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

REQ-1: ParkProximity must provide pinpoint features so that users can track the car's location on the map.

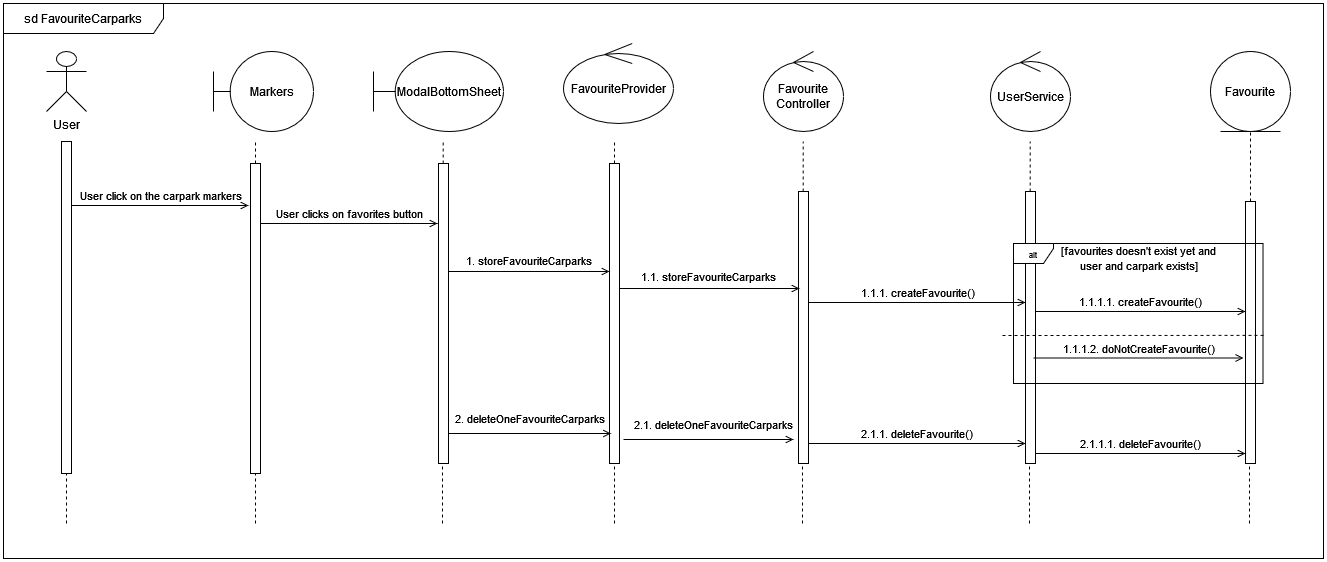
## 

## Mark Favourite Carparks

* + 1. Description and Priority

Users can bookmark their handpicked car parks for future usage. This feature has a low priority.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

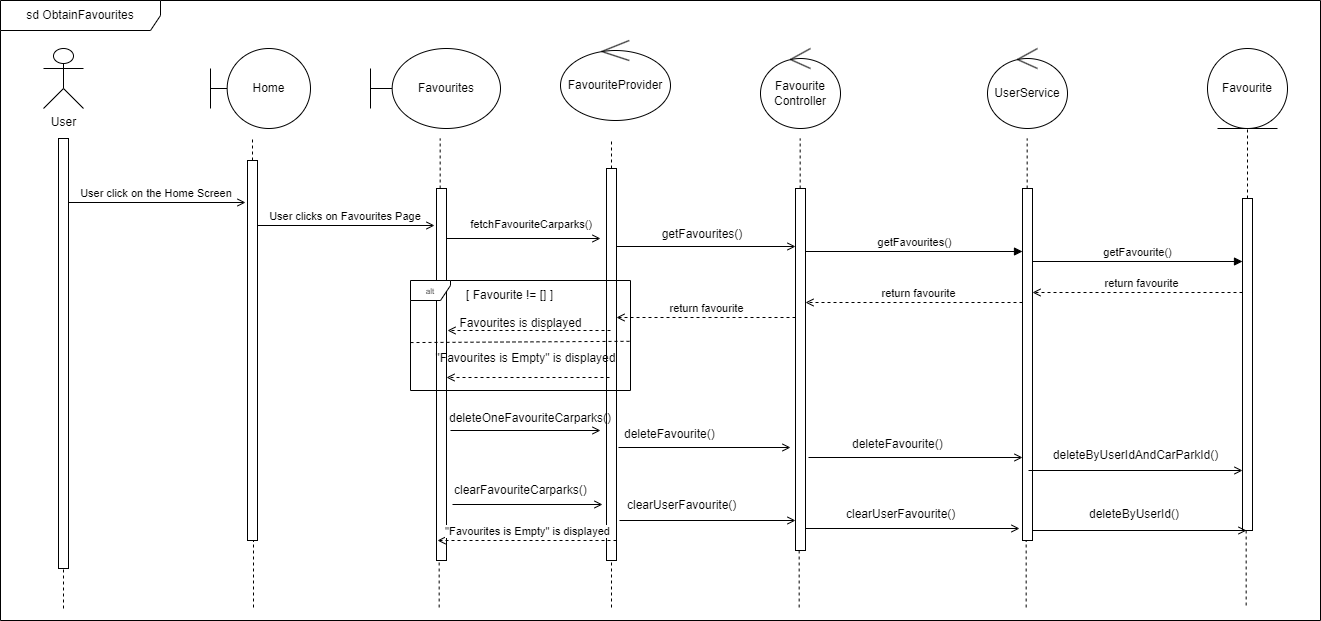
REQ-1: ParkProximity must provide an interface for a user to identify their favourite carpark(s) within the search radius revolving around their current location.

## Obtain Favourites

* + 1. Description and Priority

User can view their favourites either in the form of the favourite page or within the information displayed after clicking on the bubble. This feature has a low priority.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

REQ-1: ParkProximity must provide favourite tabs to view a list of users’ handpicked car parks.

## History Management

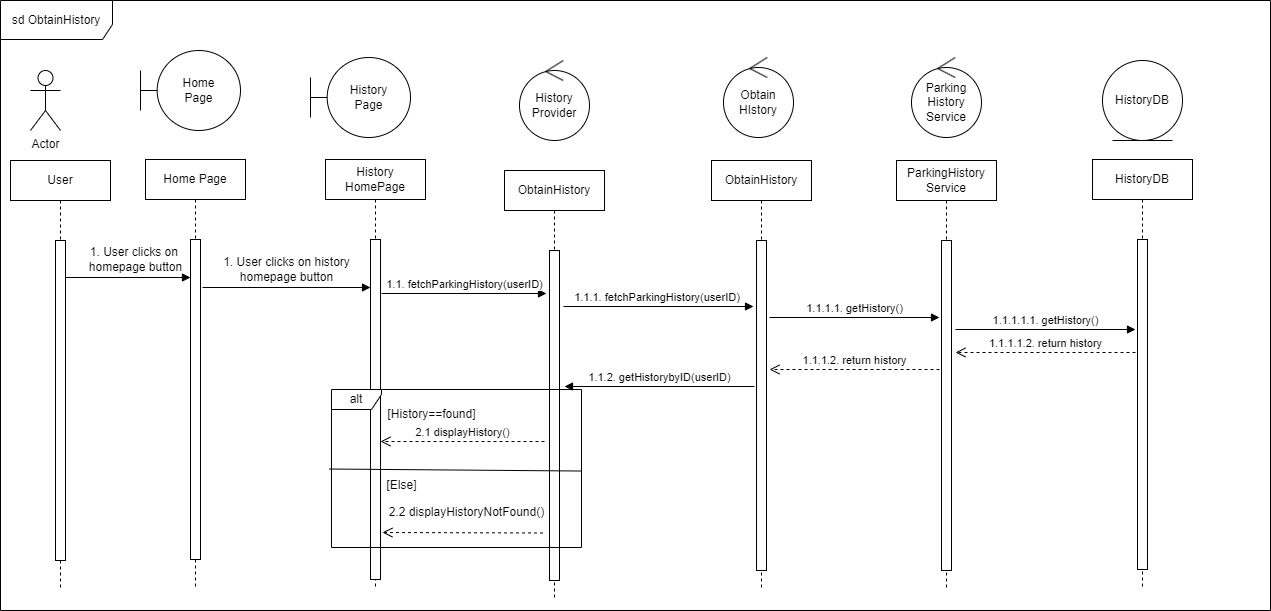
* + 1. Description and Priority

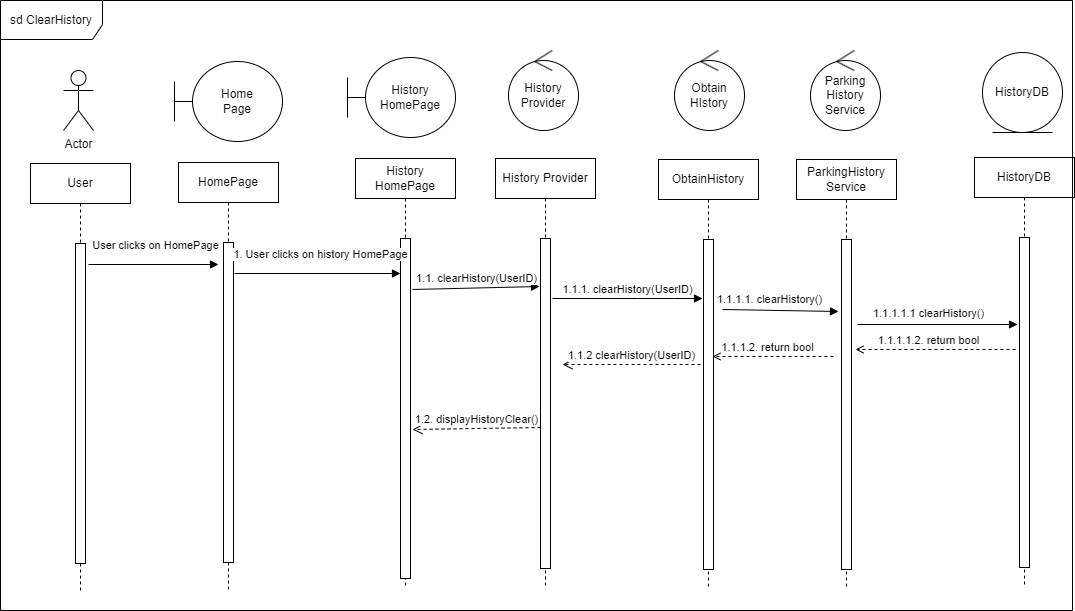
Users can view their carpark parking history inclusive of carpark location details (specifically address, duration of parking, start time, and end time) on a parking history page.

Users can also clear their parking history after accumulating too many records either manually or will be automatically removed every three months.

As this feature is not the main objective of our target users, users would not be referring to their parking history often. Therefore, this feature would be of low priority.

* + 1. Stimulus/Response Sequence





* + 1. Functional Requirements

REQ-1: ParkProximity must have history-tracking functions to allow users to read and/or delete their history independently.

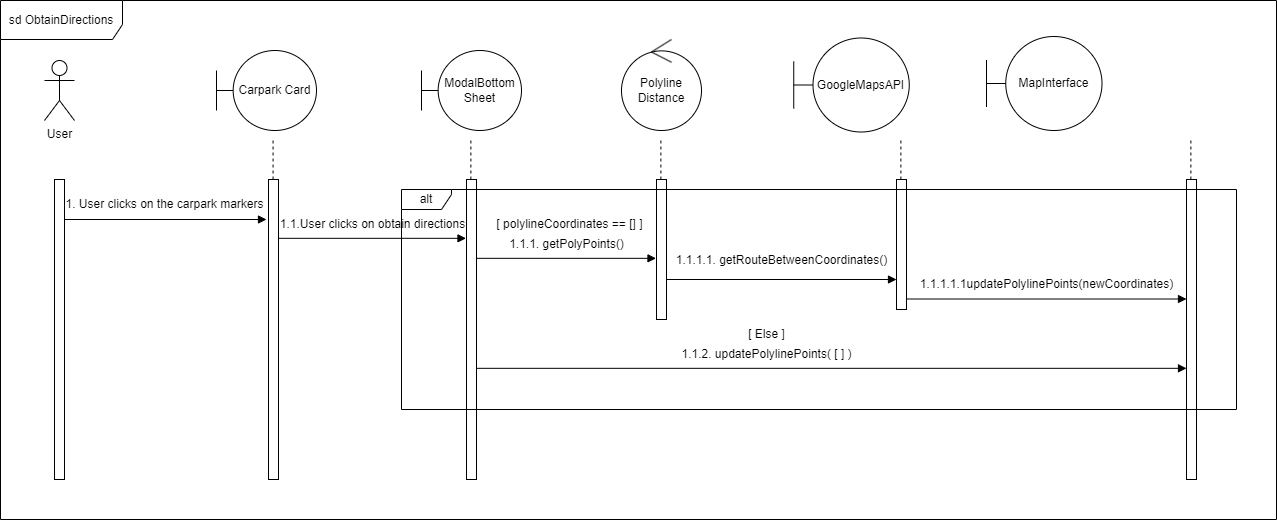
REQ-2: ParkProximity must allow users to view a list of car-parking transactions done by the users.

## Obtain Directions

* + 1. Description and Priority

After the user has chosen the specific carpark, the Google Maps API will be utilised to obtain directions to that carpark from the user’s current location. A polyline between the user’s current location and the selected carpark will be displayed for the user’s easy reference (as carpark markers will be shown around the user’s current location while the user is driving). This feature has a medium priority.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

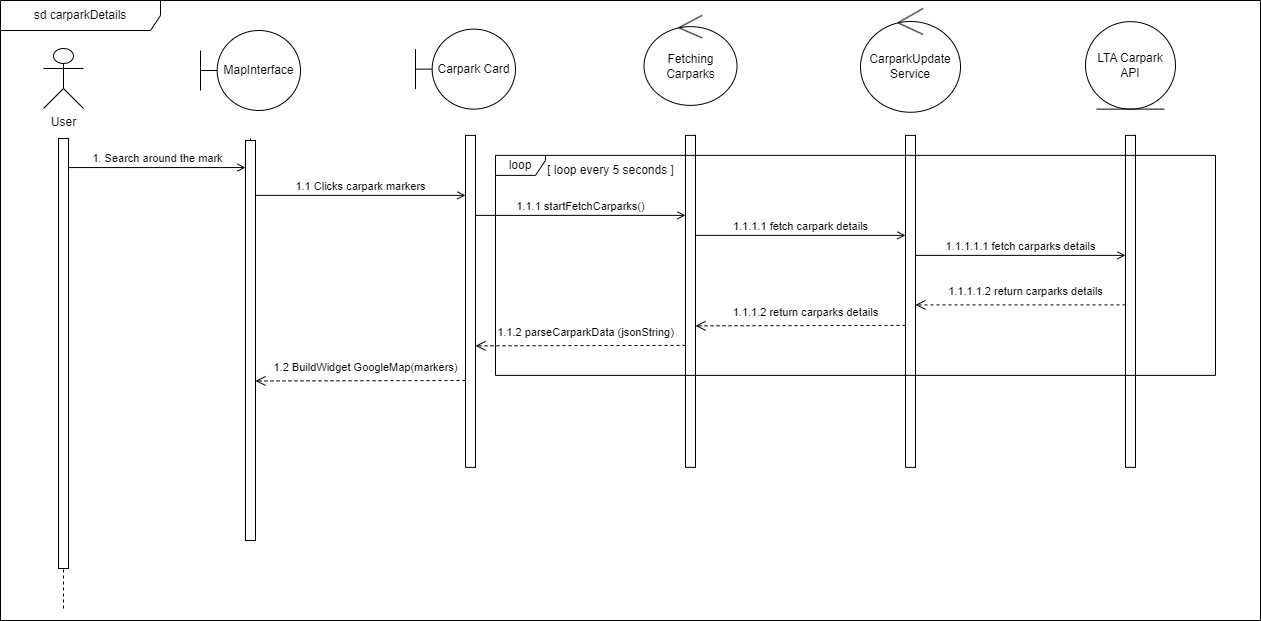
REQ-1: Users must have the ability to obtain directions to a selected carpark from their current location using the map.

## Obtain Carpark Details

* + 1. Description and Priority

The carpark details (slots availability, carparkID, slot types available, total number of car park slots within a particular carpark) are retrieved from the carpark API. Such details can be represented by the colours of bubbles in the map view. This feature has a high priority due to it being an essential requirement of ParkProximity.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

REQ-1: Carparks’ available slots are represented in terms of coloured bubbles:

* Green indicates the carpark is available (with available lots >=100).
* Yellow indicates that the carpark is more than half occupied (with available lots >=50 and <100).
* Red indicates that the carpark is fully occupied.

REQ-2: Tapping the bubble (map marker) will show more information about a specific carpark inclusive of carpark name, address, available slots, and distance from the user.

## View Current Location

* + 1. Description and Priority

Upon clicking on the current location button, the map displays the area in the perspective of the user’s current location. A blue pin indicates the user’s current location in the map view.

This feature has a high priority since the user needs to refer to his/her current location concerning other carparks of interest to the user.

* + 1. Stimulus/Response Sequence



* + 1. Functional Requirements

REQ-1: The user must be able to view his/her current location despite moving the point of interest to a place/carpark far from his/her current location.

# Other Nonfunctional Requirements

## Performance Requirements

* + 1. App must load within 3 seconds
    2. App must respond to user actions within 5 seconds
    3. Carpark details updates must occur every 60 seconds

## Safety Requirements

* + 1. User must have the ability to disable location sharing at any time via disabling their phone’s GPS access

## Software Quality Attributes

* + 1. Usability
       1. Users should be able to start using the app intuitively, without the need for extensive tutorials or guidance
       2. Users must be able to find available parking in no more than 2 steps
       3. Users must be able to adjust the search radius with a selection
       4. In-app support must be available to address user questions and issues related to map functionality and location settings
    2. Compatibility
       1. Map view must be compatible with a variety of devices and screen sizes

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Pixel 3a* | *Pixel 7* | *Pixel 4* |

* + 1. Accessibility
       1. The map interface must include appropriate labels and alternative text for screen readers
       2. Database needs to save user history from the past 3 months.
    2. Accuracy
       1. The car park information displayed should be accurate
       2. The pin-point marker must be small and visible enough to accurately indicate carpark location when there are multiple in the same vicinity
    3. Support
       1. Proper documentation must be provided for future developers
       2. The app must include help and feedback buttons for the users

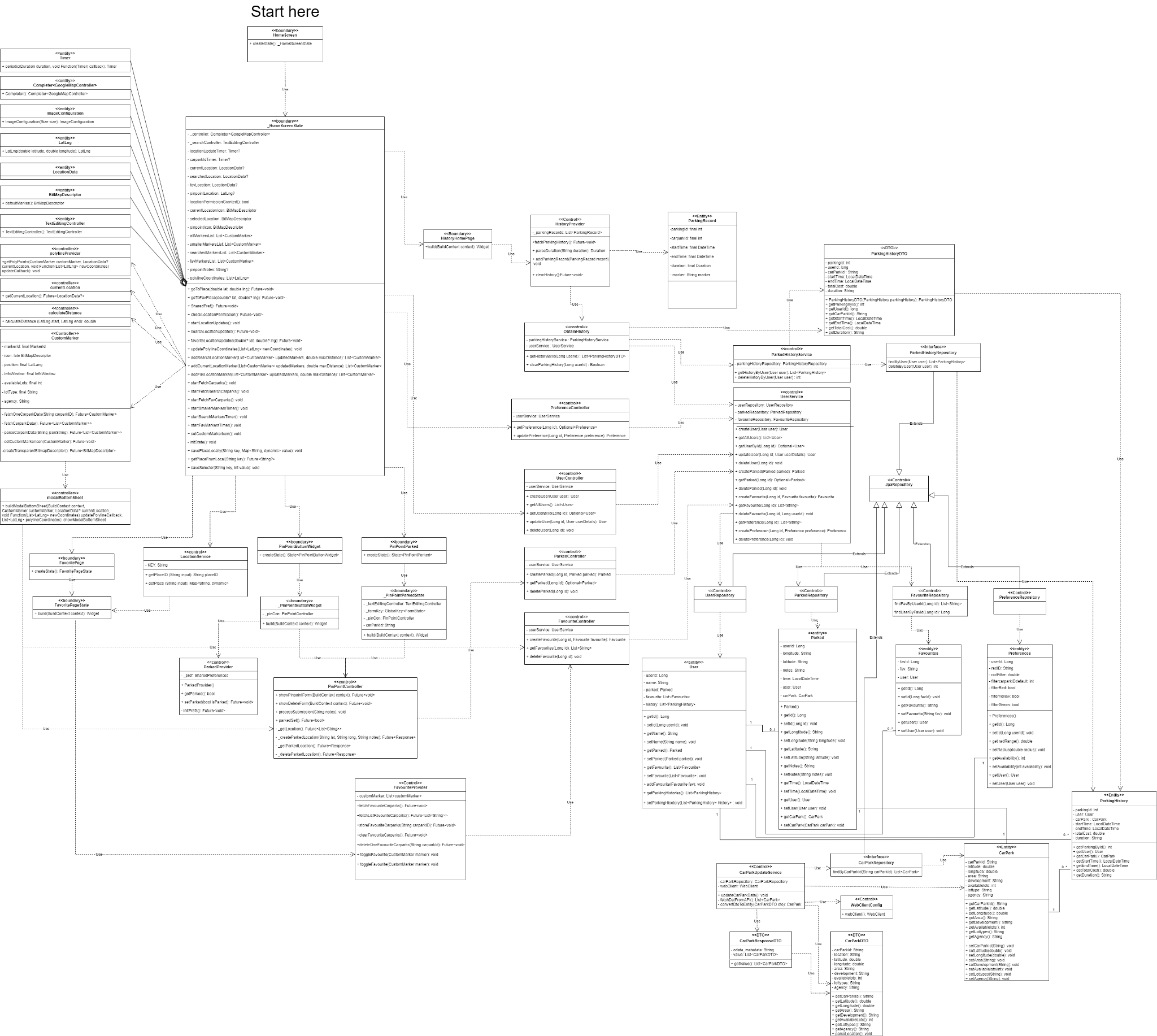
**Appendix A: Glossary**

**Data Dictionary**

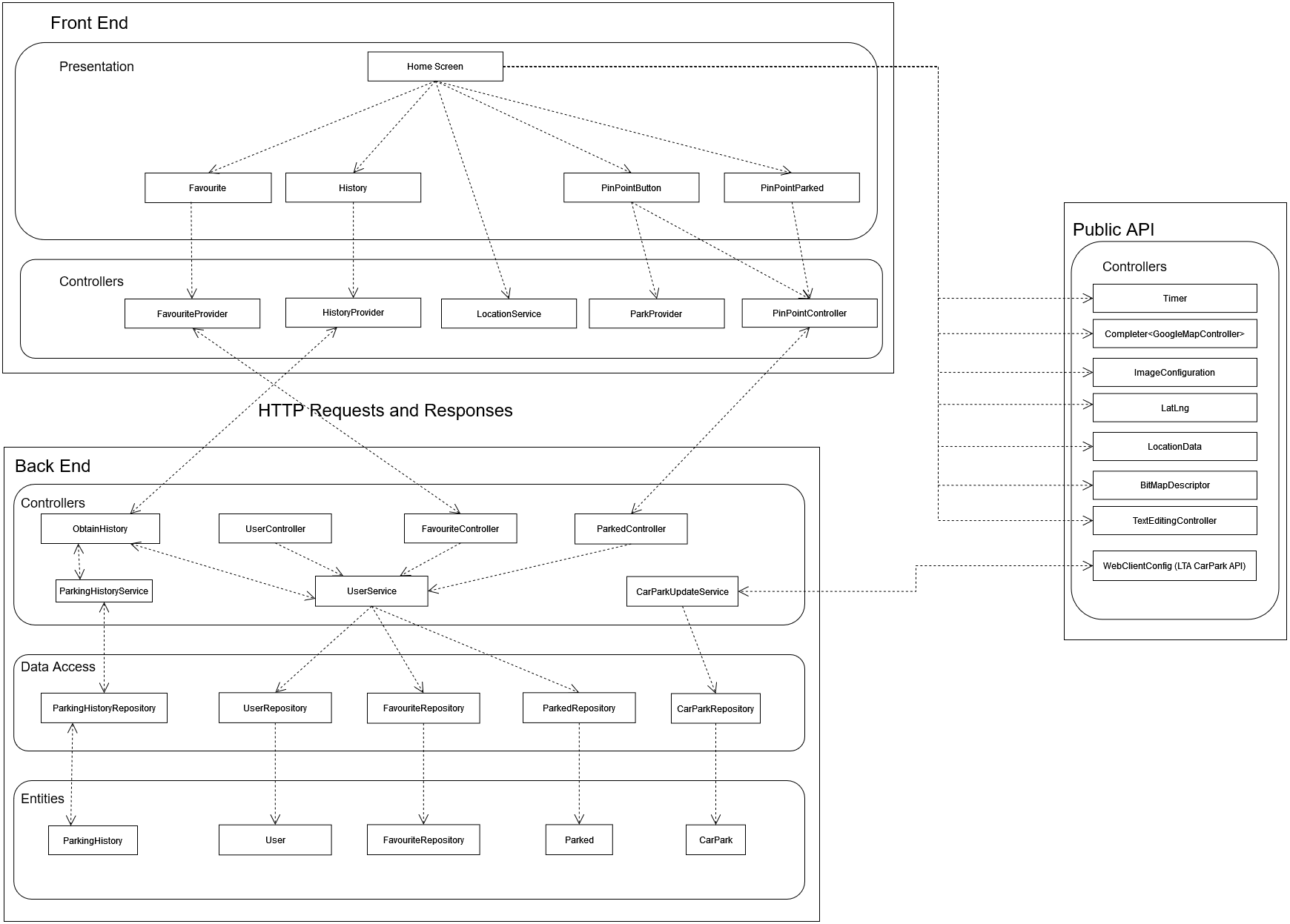
|  |  |
| --- | --- |
| **Term** | **Definition** |
| User | A person who uses our application |
| Carpark | An area where cars or other vehicles can be left temporarily |
| Global Positioning System (GPS) | A service that provides users with positioning, navigation, and timing (PNT) services |
| Location services | General term for software service for mobile users to identify their current physical location |
| Map view | A drawing of a particular area, showing its main features as viewed from above |
| Application Programming Interface (API) | A software service acting as an intermediary layer which returns the data requested by our application |
| Direction | A simple drawing on the map view to show direction of travel between two locations |
| Search | Returns the specified carpark or the nearest carpark from the searched address |
| Filter | Displays the search results according to the specified rule which is by search radius |
| Favourite | To record the selected address or carpark for future quick access |
| Frequently Asked Question (FAQ) | A list of questions and their answers, intended to help users understand how to use a particular function of our application |
| Search radius | A shaded circular region on the map view to indicate carparks within the specified distance from a location |

**Appendix B: Analysis Models**

**Class Diagram**

****

**System Architecture**

****

**Black Box Testing**

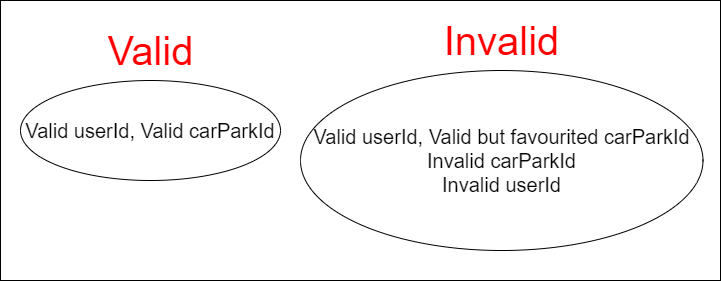
Controller: FavouriteController.java

Location: Backend/IntegratedImplementation/src/main/java/com/syntax/symphony/Favourite

/FavouriteController.java

Function: createFavourite(Long id, String carParkId)

**Equivalence class**



**Test Cases**

1. Valid userId, Valid carParkId
2. Valid userId, Valid but favourited carParkId
3. Valid userId, Invalid carParkId
4. Invalid userId, Valid carParkId

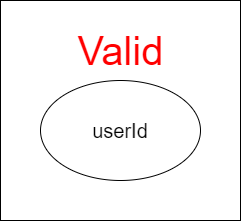
**Results Format**

The function will make use of the service UserService.java to save favourites if the inputs are valid. The results of the service will be fed to a switch case. This will in turn send a response back to the requester.

|  |  |  |
| --- | --- | --- |
| **Test Input** | **Expected Output** | **Actual Output** |
| Valid UserId, Valid carParkId | 200 OK  added 1 favourites to user: <userId> |  |
| Valid UserId, Valid but Favourited carParkId | 409 Conflict  duplicate favourite found |  |
| Valid UserId, Invalid carParkId | 404 Not Found  carpark id <carParkId> not found |  |
| Invalid UserId, Valid carParkId | 404 Not Found  user id <userId> not found |  |

Function: getFavourites(Long id)

**Equivalence class**

****

**Test Cases**

1. Valid userId, With favourites
2. Valid userId, No favourites
3. Invalid userId

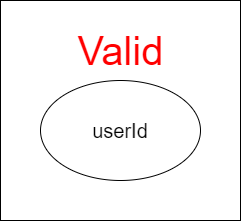
**Results Format**

The function will make use of the service UserService.java to find the favourite entries. The results of the service are returned and sent a response back to the requester.

|  |  |  |
| --- | --- | --- |
| **Test Input** | **Expected Output** | **Actual Output** |
| Valid userId, With favourites | 200 OK  [<list of carpark id>] |  |
| Valid userId, No favourites | 200 OK  [] |  |
| Invalid userId | 200 OK  [] |  |

Function: clearUserFavourite(Long userid)

**Equivalence class**

****

**Test Cases**

1. Valid userId
2. Invalid userId

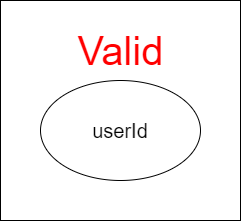
**Results Format**

The function will make use of the service UserService.java to delete all favourites of the given user id and send a 200 OK response back to the requester..

|  |  |  |
| --- | --- | --- |
| **Test Input** | **Expected Output** | **Actual Output** |
| Valid userId | 200 OK  user favourites cleared |  |
| Invalid userId | 200 OK  user favourites cleared |  |

Function: deleteFavourite(Long userid, String carparkId)

**Equivalence class**

****

**Test Cases**

1. Valid userId, Valid carparkId
2. Valid userId, Invalid carparkId
3. Invalid userId

**Results Format**

The function will make use of the service UserService.java to delete a specified favourite of the given user id and send a 200 OK response back to the requester.

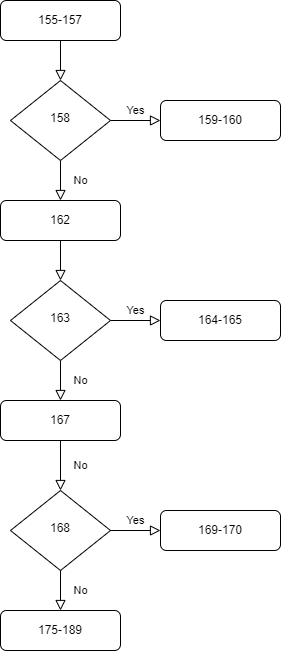
|  |  |  |
| --- | --- | --- |
| **Test Input** | **Expected Output** | **Actual Output** |
| Valid userId | 200 OK  carpark id deleted |  |
| Valid userId, Invalid carparkId | 200 OK  carpark id deleted |  |
| Invalid userId | 200 OK  carpark id deleted |  |

**White Box Testing Function 1**

Function: createFavourite(Long id, String carParkId)

Location: Backend/IntegratedImplementation/src/main/java/com/syntax/symphony/User/UserService.java

**Control Flow Graph**

****

**Cyclomatic Complexity:**

Decision Points = 3

CC = 4

Therefore, 4 basis paths to test.

**Execution Paths:**

1. 155-157, 158, 162, 163, 167, 168, 175-189
2. 155-157, 158, 159-160
3. 155-157, 158, 162, 163, 167, 168, 169-170
4. 155-157, 158, 162, 163, 164-165

**Results Format**

Results of this function will be returned to another controller (FavouriteController.java), which will be fed to a switch case. This will in turn send a response back to the requester.

**Automated Testing**

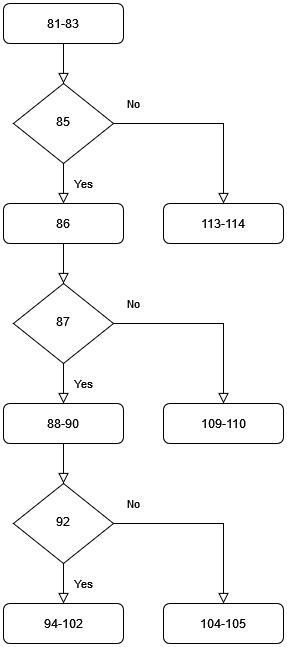
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User ID Input** | **Carpark ID Input** | **Duplicate Entry** | **Expected Output** | **Actual Output** |
| Valid | Valid | False | 1 | 1 |
| Valid | Valid | True | -3 | -3 |
| Valid | Invalid | - | -1 | -1 |
| Invalid | Valid | - | -2 | -2 |

**White Box Testing Function 2**

Function: createParked(Parked parked, String carParkId)

Location: Backend/IntegratedImplementation/src/main/java/com/syntax/symphony/User/UserService.java

**Control Flow Graph**



**Cyclomatic Complexity**

Decision Points = 3

CC = 4

Therefore, 4 basis paths to test.

**Execution Paths**

1. 81-83, 85, 86, 87, 88-90, 92, 94-102
2. 81-83, 85, 86, 87, 88-90, 92, 104-105
3. 81-83, 85, 86, 87, 109-110
4. 81-83, 85, 113-114

**Test Cases**

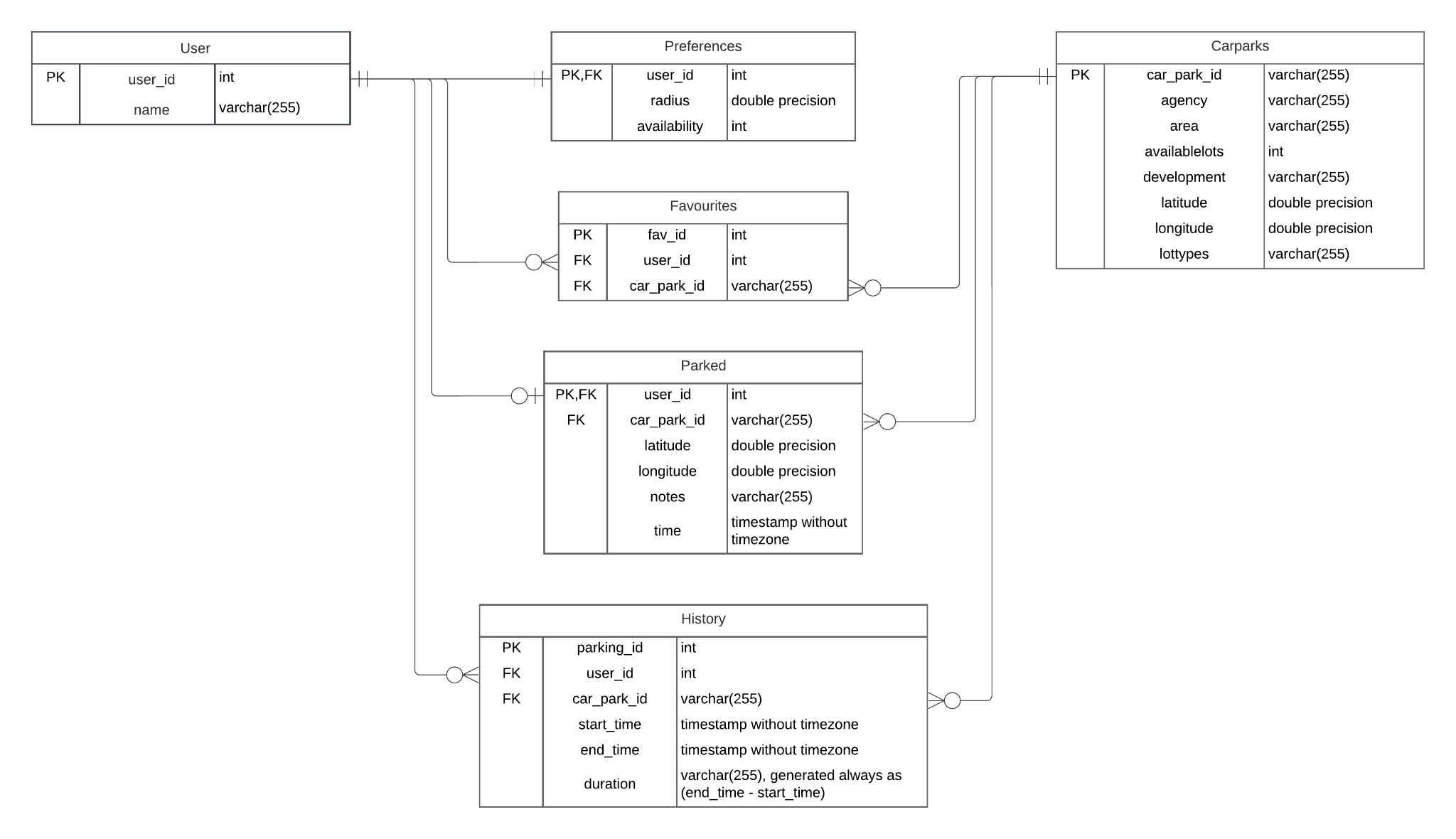
1. Valid user that has not parked yet, Valid carParkId
2. Valid user that has parked, Valid carParkId
3. Valid user, Invalid carParkId
4. Invalid user, Valid carParkId

**Results Format**

Results of this function will be returned to another controller (FavouriteController.java), which will be fed to a switch case. This will in turn send a response back to the requester.

|  |  |  |
| --- | --- | --- |
| **Test Inputs** | **Expected Output** | **Actual Output** |
| Valid not parked user, Valid carParkId | 200 OK  Parked location added for user <userId> |  |
| Valid parked user, Valid carParkId | 409 Conflict  User <userId> has already parked a location |  |
| Valid user, Invalid carParkId | 404 Not Found  Carpark id <carParkId> not found |  |
| Invalid user, Valid carParkId | 404 Not Found  User <userId> not found |  |

**Entity-Relationship Diagram**

****

**Appendix C: TBD**

*<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>*

Source: http://www.frontiernet.net/~kwiegers/process\_assets/srs\_template.doc