
Software Requirements Specification

for

<ParkIt!>

Version 1.0 approved

Prepared by <Aaron, Chong Yao, Jeremy, SiTong, Yu Xiu, Zong Zhe>

Nanyang Technological University

14 Nov 2024

Table of Contents

Table of Contents	2
Revision History	4
1. Introduction	5
1.1 Purpose	5
1.2 Document Conventions	5
1.3 Intended Audience and Reading Suggestions	6
1.4 Product Scope	6
2. Overall Description	7
2.1 Product Perspective	7
2.2 Product Functions	7
2.2.1 Navigation	7
2.2.2 Carpark Location	8
2.2.3 Bookmarking Locations	8
2.2.4 Settings	8
2.3 User Classes and Characteristics	8
2.3.1 Casual Drivers	8
2.4 Operating Environment	10
2.4.1 User Device Permissions	10
2.4.2 External API Permissions	10
2.4.3 Local Environment	10
2.4.4 Development Environment	11
2.5 Design and Implementation constraints	13
2.6 User Documentation	14
2.6.1 User Manual (For User)	14
2.7 Assumptions and Dependencies	14
3. External Interface Requirements	15
3.1 UI Mockups	15
3.2 User Interfaces	18
3.2.1 Landing Page	18
3.2.1.1 Search Bar	19
3.2.1.2 Map	19
3.2.2 Bookmark Page	20
3.2.3 Carparks Page	20
3.2.4 Settings Page	21
3.3 Hardware Interfaces	21
4. System Features	22
4.1 View Map	22
4.1.1 Description and Priority	22

4.1.2 Stimulus/Response Sequences	23
4.1.3 Functional Requirements	24
4.2 Search for a Location	24
4.2.1 Description and Priority	24
4.2.2 Stimulus/Response Sequences	24
4.2.3 Functional Requirements	26
4.3 Get Navigation Routes	26
4.3.1 Description and Priority	26
4.3.2 Stimulus/Response Sequences	26
4.3.3 Functional Requirements	27
4.4 View Nearby Carparks from a Location	28
4.4.1 Description and Priority	28
4.4.2 Stimulus/Response Sequences	28
4.4.3 Functional Requirements	30
4.5 Save and Load Bookmarks	30
4.5.1 Description and Priority	30
4.5.2 Stimulus/Response Sequences	31
4.5.3 Functional Requirements	32
4.6 Toggle Bookmark to Navigation	33
4.6.1 Description and Priority	33
4.6.2 Stimulus/Response Sequences	33
4.6.3 Functional Requirements	35
4.7 Toggle Carpark Sorting Filter	35
4.7.1 Description and Priority	35
4.7.2 Stimulus/Response Sequences	35
4.7.3 Functional Requirements	37
4.8 Toggle Vehicle Type	38
4.8.1 Description and Priority	38
4.8.2 Stimulus/Response Sequences	38
4.8.3 Functional Requirements	39
5. Non-Functional Requirements	40
5.1 Performance Requirements	40
5.2 Security Requirements	41
5.3 Software Quality Attributes	41
6. Other Requirements	42
7. Appendix A: Data Dictionary	43
8. Appendix B: Analysis Models	44
White Box Testing	50
Search Carparks	50
Navigate to Carpark	53
Bookmarks	56

Revision History

Name	Date	Reason For Changes	Version
Tan Yu Xiu	5th November 2024	initial writeup for introduction and overall description	1.0
Solis Aaron Mari Santos	12th November 2024	Product Functions, User Classes and Characteristics, Operating Environment, Design and Implementation Constraints, Assumptions and Dependencies.	1.0
Jiang Zong Zhe	13th November 2024	User Interfaces, Hardware Interfaces, Software Interfaces, Communication Interfaces, System Features, Performance Requirements,	1.0
Jiang Zong Zhe	15th November 2024	User Manual	1.0
Quek Jun Siong	16th November 2024	Safety Requirements, Security Requirements, Software Quality Attributes, Business Roles, Other Requirements	1,0
Sun Sitong	16th November 2024	White-box testing, Glossary, Diagram Documentation	1.0
Tan Chong Yao	17th November 2024	Edited the Diagram Documentation	1.0

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive and detailed description of the requirements and specifications for the ParkIt! software application. This SRS will serve as the foundation for the design, development, testing, and implementation of the ParkIt! system.

The primary objectives of the ParkIt! application are:

Carpark Information: Provide carpark availability and pricing information to users.

Carpark Navigation: Provide relevant navigation information to the desired carpark to users.

This SRS will define the functional and non-functional requirements, as well as the system architecture, user interfaces, and other technical specifications necessary to successfully implement the ParkIt! application. The information contained in this document will ensure that all stakeholders have a clear understanding of the system's intended capabilities, constraints, and design.

The intended audience for this SRS includes the project sponsor, project manager, software developers, quality assurance team, and any other key stakeholders involved in the ParkIt! initiative.

1.2 Document Conventions

This Software Requirements Specification (SRS) document follows standard typographical conventions to ensure clarity and consistency. Text in bold represents GUI elements, user inputs, or other emphasis. Text in italics denotes newly introduced terms, important notes, or areas requiring special attention. Source code, API endpoints, and database queries are presented in monospace font. Priorities for high-level requirements are inherited by detailed requirements unless explicitly stated otherwise.

Software Requirement Specification Standard: IEEE830-1998. Priorities of higher level requirements are inherited by detailed level requirements.

Font: Times New Roman

Heading 1: Size 18, Bold

Heading 2: Size 14, Bold

Heading 3: Size 12, Bold

Content: Size 12

Spacing in Content: 1 line spaced

Further conventions on special terms used throughout this document are described in Appendix A: Data Dictionary.

1.3 Intended Audience and Reading Suggestions

This document is primarily meant for the **Product Manager** helping the development and maintenance of ParkIt. This document also contains useful contextual information that can aid **Software Engineers** in application development.

Product Managers can focus on first reading **2.2: Product Functions** before moving to **3.3: User Interfaces** to understand the application design. After that, **4: System Features** give a detailed breakdown of system capabilities, while **Appendix B: Analysis Diagrams** give visualisations of system design.

Software Engineers can look into **4: System Features** to analyse the capabilities of a particular function that they are working on. Then, **2.6: User Documentation** and **4: System Features** should be updated based on future application developments.

1.4 Product Scope

Google Maps, the most used application used by drivers for navigation, does not provide real-time updates on carpark availability, which is something our drivers are concerned about.

Some developers attempted to tackle this issue by creating websites or web applications that display carpark availability. However, most of the current solutions require users to put in extra effort to find the carpark name, key it in, and then click on the most relevant carpark. This process can be very inconvenient, and especially dangerous if the driver is trying to search for parking lots while driving. Even then, existing solutions do not provide the price of parking, and many drivers may only be aware of how much they've spent on parking after leaving the carpark. On top of that, current solutions also do not provide a navigation route to the selected carpark, making the process even more tedious and inconvenient.

Therefore we came up with the problem statement: How can we help drivers find available and affordable parking by integrating live updates on availability and pricing with map navigation to create an all-in-one platform for drivers?

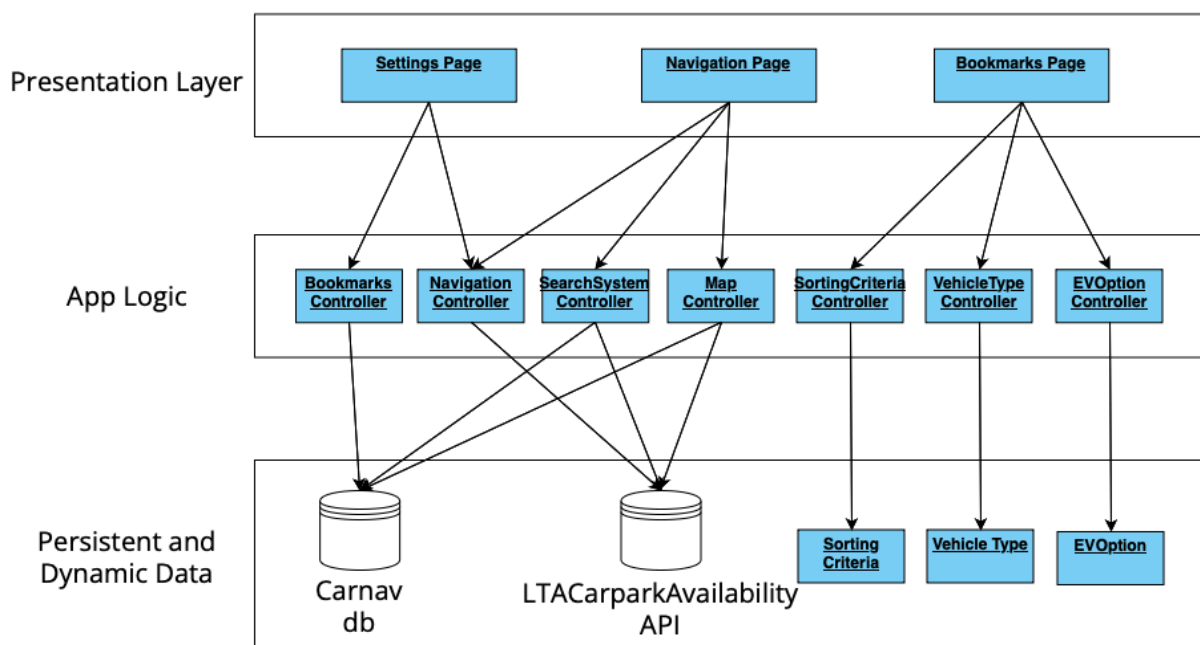
With all these issues in mind, we wanted to create an all-in-one app to reduce the hassle of finding a carpark. Our application ParkIt thus aims to give drivers real-time updates on carpark availability and pricing, fixing the pain point of drivers, while also providing the navigation route. It utilises API calls from LTA's DataMall and HDB's Carpark Availability Systems. It also uses the Mapbox library and API to display maps, and SQLite database to store data. We used React for frontend design and development, and flask for the backend database and API handling.

2. Overall Description

2.1 Product Perspective

ParkIt! Is a new, self-contained product developed to streamline the process of finding real-time parking availability and pricing information for urban drivers. ParkIt! Addresses a growing demand among city drivers for an efficient way to locate parking spaces without the hassle of multiple searches or visiting full car parks. Unlike standard navigation applications, which often lack live parking data, ParkIt! Provides an integrated, user-friendly solution specifically focused on parking management.

A simple system architecture diagram is shown below for a brief overview of the operation of ParkIt!.



2.2 Product Functions

The ParkIt! Application is designed to provide drivers with a seamless parking experience by integrating real-time car park data, navigation, and user customisation features. Below is a high-level summary of the product's major functions, organised into key categories:

2.2.1 Navigation

1. Provides directions to the selected carpark using mapbox's routing capabilities
2. Supports route recalculations based on user inputs.
3. Displays estimated travel distance of carpark from desired location

2.2.2 Carpark Location

1. Displays nearby carpark based on the user's current location or a specified destination
2. Retrieves real-time carpark availability and pricing data via LTA DataMall and HDB API's
3. Allows users to search for carpark near locations using the search bar

2.2.3 Bookmarking Locations

1. Enables users to save frequently used carpark locations for easy future access
2. Allows adding, viewing and removing bookmarks from a dedicated bookmarks page.
3. Syncs bookmarked locations with the search and navigation functionalities for quick navigation
4. Provides a quick search feature within bookmarks to find locations to save efficiently.

2.2.4 Settings

1. Lets users customise sorting preferences based on lot availability, proximity or price
2. Allows users to toggle the app to search for car parking lots or motorcycles parking lots
3. Allows users to highlight specific carpark with special features like an EV charging station.

2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>

2.3.1 Casual Drivers

Aspect	Description
Frequency of Use	Moderate
Subset of Product Functions Used	All
Technical Expertise	<ul style="list-style-type: none">- Access to a mobile phone- Familiar with using a web based application
Age	18 - 60

Characteristics	<ul style="list-style-type: none"> - Infrequent drivers requiring simple, intuitive functionality for occasional trips - Drivers who rent vehicles
-----------------	--

2.3.2 Seasoned Drivers

Aspect	Description
Frequency of Use	High
Subset of Product Functions Used	All
Technical Expertise	<ul style="list-style-type: none"> - Access to a mobile phone - Familiar with using a web based application
Age	18 - 60
Characteristics	<ul style="list-style-type: none"> - Regular commuters seeking optimised parking solutions and customisation for frequent use - Urban drivers seeking parking lot availability in dense urban areas with limited parking availability

2.3.3 EV Drivers

Aspect	Description
Frequency of Use	Moderate to High
Subset of Product Functions Used	All
Technical Expertise	<ul style="list-style-type: none"> - Access to a mobile phone - Familiar with using a web based application - Familiar with maps
Age	18 - 60
Characteristics	<ul style="list-style-type: none"> - Drivers that are focused on EV charging availability as their vehicles need time to recharge

2.3.4 Tourists and Visitors

Aspect	Description
Frequency of Use	Low
Subset of Product Functions Used	All
Technical Expertise	<ul style="list-style-type: none"> - Access to a mobile phone - Familiar with using a web based application
Age	18 - 60
Characteristics	<ul style="list-style-type: none"> - Temporary users unfamiliar with local systems.

2.4 Operating Environment

The production and development environment of ParkIt! Web application will be covered under this section

2.4.1 User Device Permissions

1. Internet Connection: As all functionalities of the application rely on real-time API calls to external services (e.g. LTA DataMall, HDB Carpark Availability Systems, and Mapbox), a stable internet connection is mandatory.
2. Location Access: the application requests permission to access the user's device location to identify nearby carpark and provide navigation routes from the user's current position to their destination

2.4.2 External API Permissions

The following permissions are required for integration with external services

1. Mapbox: For map visualisation and route navigation, Mapbox permissions are needed to retrieve and display map data
2. LTA DataMall: The application uses LTA DataMall API to access real-time car park availability and pricing information
3. HDB Carpark Availability: Data from HDB API provides additional information on availability in Housing and Development Board-managed car parks

2.4.3 Local Environment

The ParkIt! Application operates entirely in a local environment, requiring the following system setup:

- Frontend: Runs locally using React.js, served through a development server such as npm start
- Backend: runs on a local flask server, providing the API layer for handling requests and managing the SQLite database
- Database: Utilises SQLite, a lightweight database that operates locally to store cached carpark data and user preferences like bookmarks.
- Compatibility:

- The application is compatible with modern web browsers like Chrome, Firefox, Safari and Edge for accessing the frontend interface
- Optimised for use on personal computers running Windows, macOS, or Linux

2.4.4 Development Environment

The development of ParkIt! Is carried out in the following local setup:

- System Requirements:
 - Windows, MacOS, or Linux
 - RAM: Minimum 8gb for running the development server and backend processes
 - Internet: Required for testing API integrations and downloading dependencies

Development Environment	Description
Version Control: Git, Github	Git is used for version management and collaborative development, with repositories maintained locally or on platforms like GitHub.
Frontend: React.js	<p>React.js is an open-source JavaScript library created by Facebook, designed for building dynamic and responsive user interfaces, particularly for single-page applications. It allows developers to create reusable UI components that can manage and display data in real-time, offering a streamlined approach to building complex and interactive front ends. React's component-based architecture simplifies the development process, as each component manages its own state, enabling more organised and maintainable code.</p> <p>One of React's key features is its virtual DOM, which enhances performance by only updating parts of the interface that have changed instead of re-rendering the entire page. This optimization results in faster and smoother user experiences. Additionally, React's flexibility and compatibility with other libraries and frameworks make it a preferred choice for developers seeking to build scalable applications with efficient rendering and responsive interactions.</p>
Backend:	Flask is a lightweight and open-source web

Python3 with Flask	<p>framework for Python, designed to create robust and dynamic web applications and APIs. Built with simplicity and flexibility in mind, Flask enables developers to build custom backend systems tailored to the needs of their application. Its minimalistic architecture makes it ideal for managing RESTful APIs, server-side logic, and database interactions, offering a streamlined solution for backend development.</p> <p>Flask's modular design allows developers to include only the necessary components, making it lightweight and efficient. This approach provides a high level of customization while maintaining clarity and organisation in the codebase. Flask includes core functionalities such as URL routing, request handling, and session management, while also supporting integration with external libraries for advanced features, like database handling with SQLite or API requests to external data sources.</p> <p>One of Flask's key strengths is its simplicity, which reduces development time and makes it easier to focus on implementing application-specific logic. Its compatibility with Python's extensive ecosystem of libraries enhances its flexibility, enabling features such as data processing, caching, and real-time updates. Flask's adaptability ensures developers can create scalable and maintainable backends, making it a preferred choice for building dynamic and responsive applications like ParkIt!.</p>
Database: SQLite	<p>SQLite is a lightweight, self-contained, and open-source database engine designed to provide efficient local storage solutions. Known for its simplicity and reliability, SQLite operates directly within the application without requiring a separate database server. This makes it an ideal choice for smaller-scale applications like ParkIt!, where local data storage and minimal setup are essential.</p>

	<p>Unlike traditional database management systems, SQLite uses a serverless architecture, meaning the database is stored as a single file on the local system. This design enables seamless integration with applications, reducing the complexity of configuration and deployment. SQLite supports SQL queries, offering a familiar and powerful way to manage structured data while maintaining a compact and portable footprint.</p> <p>One of SQLite's key features is its ACID compliance (Atomicity, Consistency, Isolation, Durability), ensuring reliable transactions and data integrity. Additionally, it requires minimal system resources, making it highly efficient for applications that store lightweight data such as user bookmarks or cached car park details. Its ability to run on multiple platforms with the same file format makes SQLite both versatile and highly portable.</p> <p>In ParkIt!, SQLite is used to store user data, such as bookmarked car parks and cached parking availability, providing quick and reliable access. Its ease of use and performance make it a practical choice for applications requiring local data management without the overhead of a full database server.</p>
--	--

2.5 Design and Implementation constraints

Constraints Include:

- Limited development budget hinders the usage of more advanced and reliable APIs.
- Use of a specific framework (e.g. React for front-end, Python for back-end) as determined by the development team.
- Reliance on real-time data from external APIs (e.g., LTA DataMall, HDB Carpark Availability Systems) may lead to performance degradation due to rate limits or API downtime.
- The application is locally hosted, requiring users to set up their environment (e.g., installing dependencies, configuring servers) rather than using centralised hosting or cloud services.
- Use of SQLite as the local database restricts scalability and concurrency, making it unsuitable for handling simultaneous user access or large-scale data storage

- Real-time integration between government databases, such as LTA DataMall and HDB Carpark Availability Systems, and the database used for the development of ParkIt! to ensure accurate updates on carpark availability, pricing, and the addition of new carparks or related facilities.

2.6 User Documentation

The User documentation will include:

2.6.1 User Manual (For User)

The User Manual is a comprehensive guide for end-users to use ParkIt!'s core features. It can be accessed at

<https://github.com/ZongZheJiang/SC2006/blob/main/README.md>

2.7 Assumptions and Dependencies

Assumptions:

- End users to give permission to the web application to track the location of the device
- Steady internet connectivity for end-users to access the web application.
- Continuous availability of third-party services for the map function and the database framework provider.

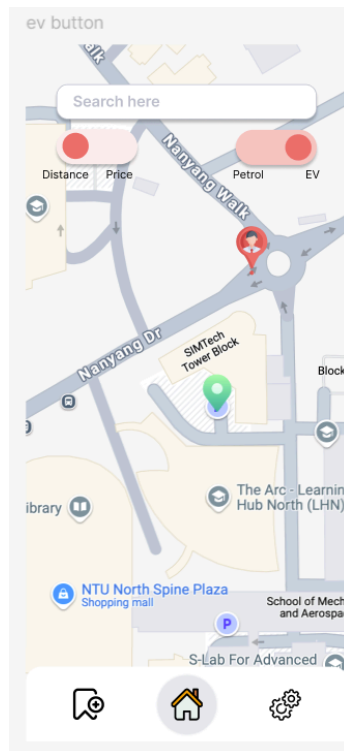
Dependencies:

- Reliance on third-party APIs for syncing with the map.
- The app relies on SQLite for local data storage, including cached carpark details, user preferences, and bookmarks.
- The app's functionality for finding nearby carparks depends on access to the device's location services.
- Accuracy and timeliness of external data sources are crucial for providing meaningful recommendations to users.

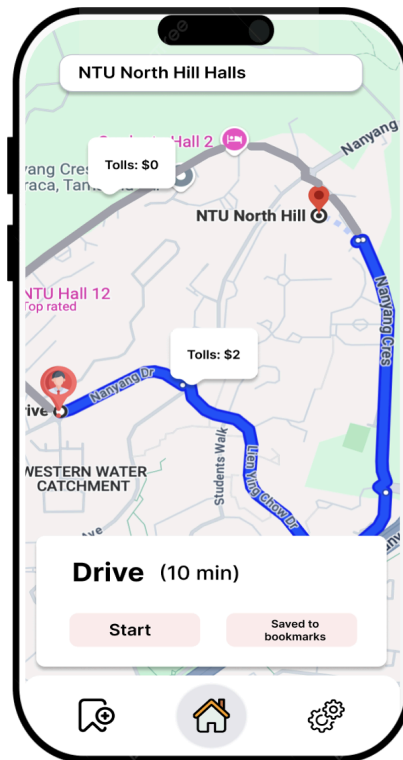
3. External Interface Requirements

3.1 UI Mockups

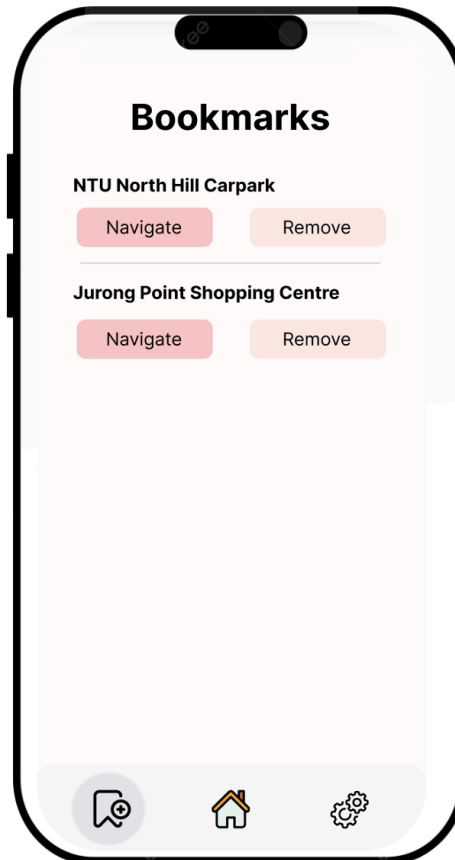
3.1.1 Landing Page

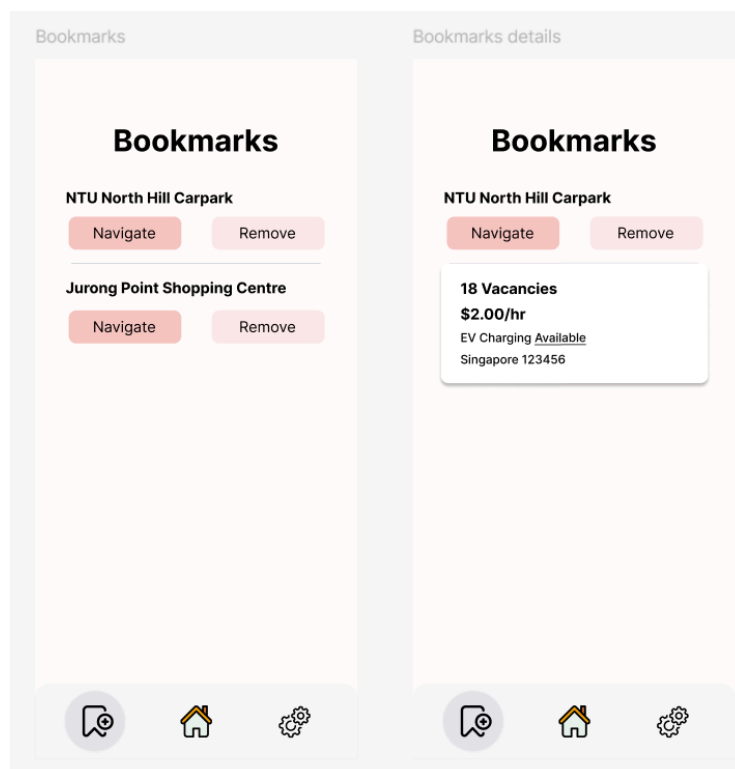
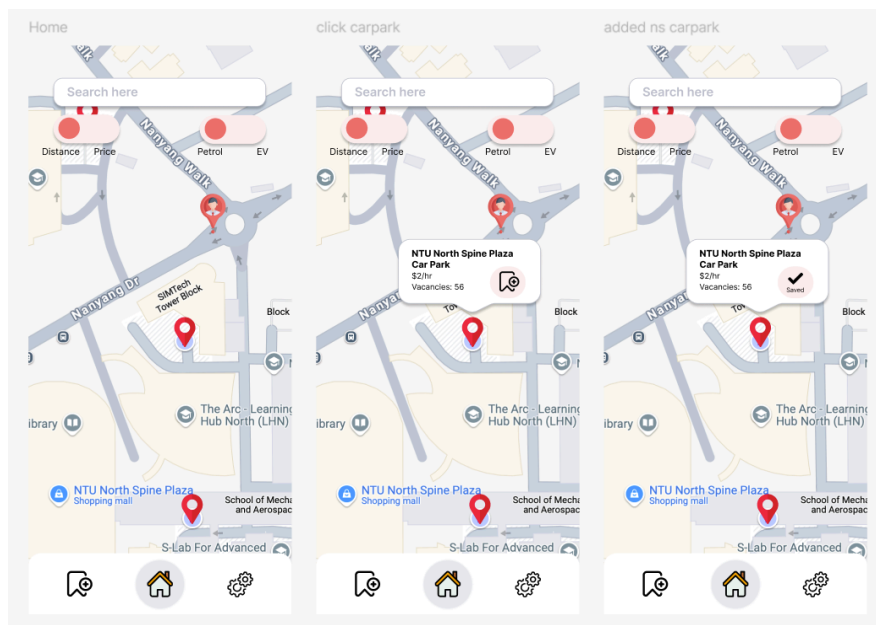
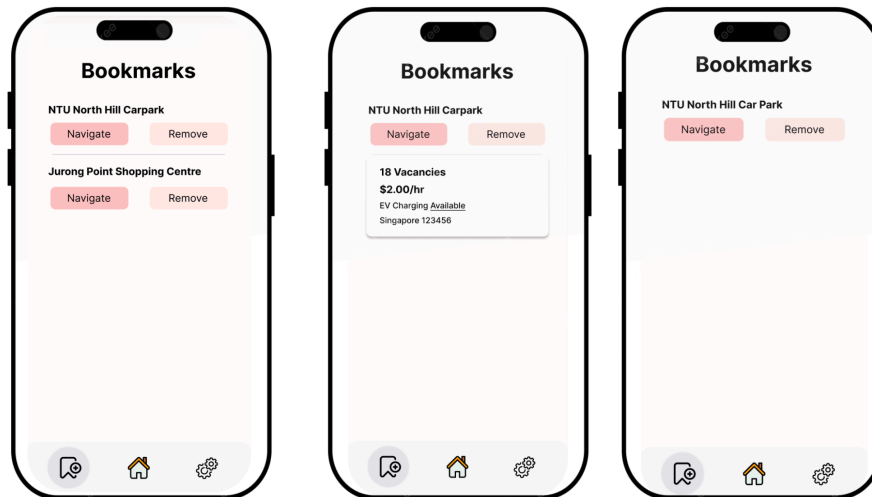


3.1.2 Search Bar



3.1.3 Bookmarks





3.2 User Interfaces

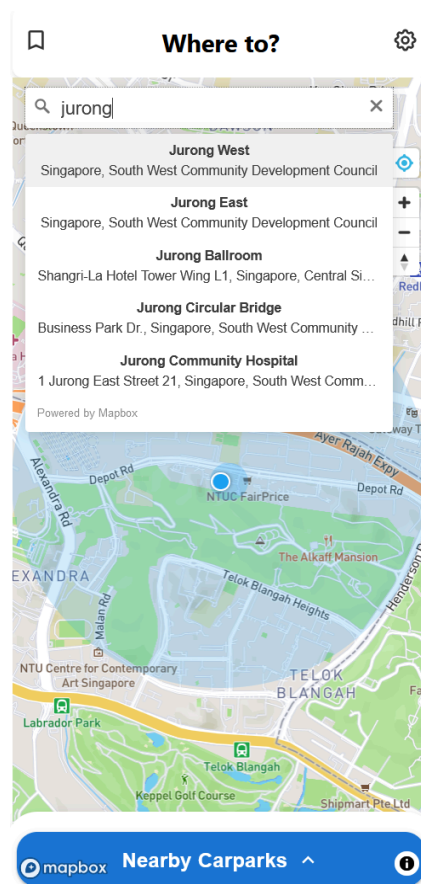
3.2.1 Landing Page

Welcome Back!

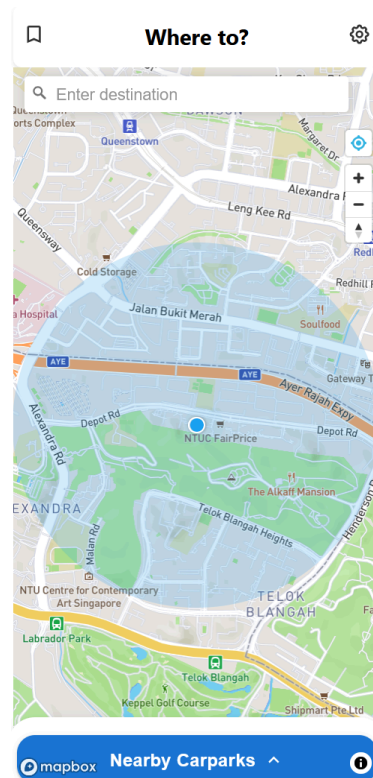
Ready to travel?



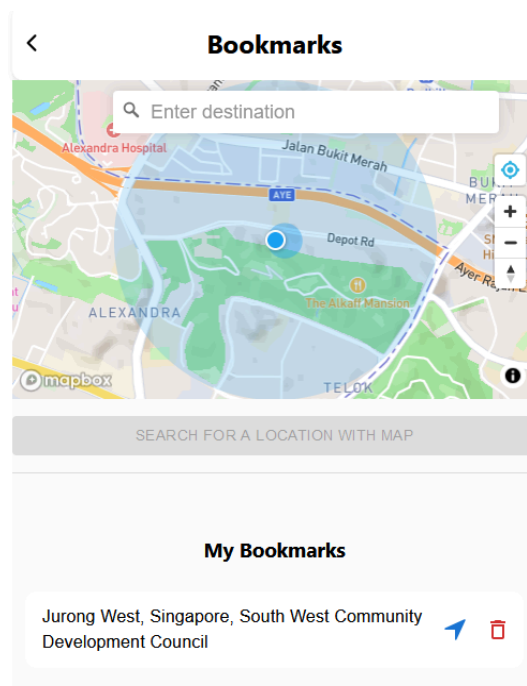
3.2.1.1 Search Bar



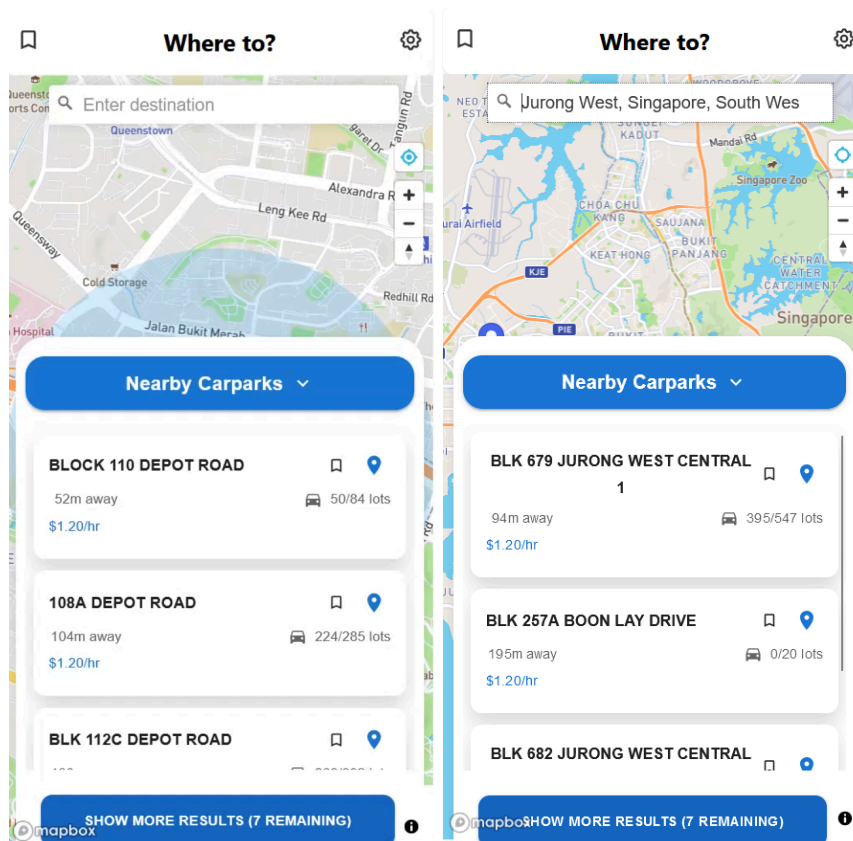
3.2.1.2 Map



3.2.2 Bookmark Page





3.2.3 Carparks Page



3.2.4 Settings Page

< **Settings**

Select your vehicle type


Selected: Car

Customise your search priority

LOCATION PRICE LOTS


The default search option, perfect for when you need to park quickly or prefer minimal walking distance.


Electric Vehicle Preferences



Regular Parking Spaces Only

Maximum parking distance: 100 meters





3.3 Hardware Interfaces

This section describes the hardware interfaces required for the ParkIt! web application to perform effectively and reliably.

Requirements	Description
Operating System	<ul style="list-style-type: none"> - For Windows : Windows 8 and later versions. - For macOS : macOS 10.10 and later version. - For Linux : Ubuntu, Debian, Fedora
Network Connection	Wireless Network Interface Card (WNIC) or a modern chip with cellular modem.
Interaction	A functional touchpad or mouse to navigate through the website.

3.4 Software Interfaces

OS : Windows 11

Tools : React JS, Python, SQLite

Third Party Libraries:

- Leaflet
Lightweight, open-source JavaScript library for creating interactive maps that are mobile-friendly, customizable, and easy to integrate with various mapping services like OpenStreetMap

APIs:

- Mapbox API
The Mapbox API provides developers with a powerful suite of tools to integrate interactive maps, geolocation, and navigation features into their applications. It offers access to high-quality, customizable map data and services, enabling developers to create dynamic user experiences tailored to specific geographic needs.
- HDB Carpark Lots API
The HDB Carpark Lots API allows developers to retrieve real-time data on parking availability across Housing and Development Board (HDB) carpark lots in Singapore. It provides accurate, up-to-date information on carpark occupancy, making it easier to manage and optimise parking solutions for users.
- LTA Carpark Lots API
The LTA Carpark Lots API provides developers with real-time data on carpark availability in Singapore, sourced from the Land Transport Authority's systems. It enables applications to access up-to-date information about carpark lot occupancy and availability, helping users make informed decisions for parking.

4. System Features

4.1 View Map

4.1.1 Description and Priority

Users should be able to view the map. If location access is enabled, the map will hone in on the user's current location and show the surrounding area. If location access is not enabled, the map will show the general area of the country. This map will be used extensively, across other features such as querying for locations and showing navigation routes.

Overall Priority	Description
High	Functionality of location search and navigation routing relies heavily on the map as a display.

4.1.2 Stimulus/Response Sequences

Use Case ID:	#1		
Use Case Name:	View Map		
Created By	Jiang Zong Zhe	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (Initiating), Mapbox Library/API
Description	Users can view the geospatial 2D map
Preconditions	<ol style="list-style-type: none"> 1. User must be connected to the Internet 2. The Mapbox API or library must be accessible and functioning without service interruptions.
Postconditions	<p>Upon map load, user's is prompted for their current location to be revealed. Allowing current location would display the user's location on the map</p> <p>OR</p> <p>The user's browser blocks location access</p>
Priority	High
Frequency of use	<p>High</p> <p>Users will view the map every time the app is initialised, and other functionalities rely on the map display.</p>
Flow of events	<ol style="list-style-type: none"> 1. User opens app and the map initialises with the general map of Singapore 2. User enables location access 3. Map hones in on the user's current location and surrounding area.
Alternative flows	<ol style="list-style-type: none"> 1. User disables location access <ol style="list-style-type: none"> a. Map remains at the general map of Singapore
Exceptions	<ol style="list-style-type: none"> 1. User has no internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown.

Includes	None
Special requirements	Stable Internet Connectivity
Assumptions	None
Notes and Issues	None

4.1.3 Functional Requirements

REQ-1: System must initialise and display a map interface upon the launch of the application using Mapbox API.

REQ-1.1: System must be able to display a loading indicator while map is being initialised.

REQ-1.2: System must be able to handle map initialisation failures

REQ -2: System must request location access permission from users upon first initialisation.

REQ-2.1: When location access is granted, the system must centre the map on the user's current location.

REQ-2.2: When location access is denied, the system must display the default Singapore map view.

REQ-3: The system must support user interaction with the map

REQ-3.1: The system must provide standard map controls:

REQ-3.1.1: Enable users to zoom in/out

REQ-3.1.2: Allow map to be rotatable

REQ-3.1.3: Allow smooth transition while panning across the map

4.2 Search for a Location

4.2.1 Description and Priority

Users must be able to query for locations in Singapore

Overall Priority	Description
High	Users must be able to find nearby car parks for other locations other than their current location so as to plan their navigation routes.

4.2.2 Stimulus/Response Sequences

Use Case ID:	#2-1
--------------	------

Use Case Name:			
Created By	Jiang Zong Zhe	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (Initiating), Mapbox API
Description	Drivers can search for a location of interest
Preconditions	<ol style="list-style-type: none"> 4. The app user has launched the app 5. The app user has access to the internet
Postconditions	<ol style="list-style-type: none"> 1. Successful exit condition: The app user receives information about location of interest and presses the 'leave' tab 2. Unsuccessful exit condition: The system informs the user that no relevant locations were found with the search query or there was an invalid location input
Priority	High
Frequency of use	High
Flow of events	<ol style="list-style-type: none"> 1. User enters a valid location into the search box. 2. System sends a request to the MapBox API and fetches relevant location results. 3. System displays consolidated details of most relevant location results in a dropdown list - LocationName, Address, DistanceAway. 4. User selects the chosen location. 5. System shows the selected location on the map. 6. System shows nearby car parks to that location using the extended use case '<i>View Nearby Car parks</i>'. 7. System shows valid navigation routes using the extended use case '<i>Find Navigation Routes</i>'.
Alternative flows	<ol style="list-style-type: none"> 1. User enters an invalid location into the search box. <ol style="list-style-type: none"> a. System fails to find any relevant locations based on the search query b. System informs the app user that no results were found and suggests refining the search c. The app user refines the query and the system resumes from the normal flow at step #1
Exceptions	<ol style="list-style-type: none"> 1. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown.

Includes	<i>View Nearby Carparks</i> <i>Get Navigation Route</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.2.3 Functional Requirements

REQ-1: The system must allow users to manually enter a location.

REQ-2: The system must process the user's search input.

REQ-2.1: The system must send the search request to Mapbox API.

REQ-2.2: The system must validate the user's input.

REQ-2.2.1: If input is valid, the system must display the nearby carparks' information on the user interface.

REQ-2.2.2: If input is not valid, the system must provide feedback.

4.3 Get Navigation Routes

4.3.1 Description and Priority

Users will be shown the navigation route (via car/motorcycle) to a chosen location.

Overall Priority	Description
High	This works in tandem with the functionality to view the vacancy of nearby carparks, as drivers will likely want to see the route to the carpark after checking its vacancies.

4.3.2 Stimulus/Response Sequences

Use Case ID:	#2-2		
Use Case Name:	Get Navigation Routes		
Created By	Tan Chong Yao	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (initial actor), Mapbox API
-------	----------------------------------

Description	Users can view navigation routes to chosen location
Preconditions	<ol style="list-style-type: none"> 1. User enables location access 2. User has internet access
Postconditions	Successful exit condition: User can view the navigation route to their desired location
Priority	High
Frequency of use	High
Flow of events	<ol style="list-style-type: none"> 1. User searches for a location on the map using extended use case '<i>Search for a Location</i>', and selects the desired location. 2. System sends a request to the MapBox API to find the fastest route. 3. Map displays the route to the location.
Alternative flows	<ol style="list-style-type: none"> 1. User chooses a bookmarked location to navigate to. <ol style="list-style-type: none"> a. User selects "Bookmark" tab on the UI. b. User selects "Navigate To" button on a saved bookmark. c. User is routed to the Main Page. d. System displays navigation route to the saved location.
Exceptions	<ol style="list-style-type: none"> 1. User does not allow location access <ol style="list-style-type: none"> a. No navigation route is shown to the selected location
Includes	<i>Search for a Location</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.3.3 Functional Requirements

REQ-1: The system must check for prerequisites for navigation

REQ-1.1: System must check for location access status.

REQ-1.1.1: If location access is not granted, the system must display an error message.

REQ-1.2: System must check for internet connectivity.

REQ-1.2.1: If the device is not connected to the internet, the system must notify the user.

REQ-2: The system must display navigation routes when all prerequisites for navigation are met.

REQ-3: The system must be able to navigate to bookmarked locations without searching for the exact location in the search function.

REQ-3.1: System must display navigation button in bookmarks page.

REQ-3.2: The system must redirect to the main page with the route displayed.

4.4 View Nearby Carparks from a Location

4.4.1 Description and Priority

Users should be able to view a list of nearby carparks, along with information such as available parking spaces and price, within a 500m radius of their current location or any specified location on the map.

Overall Priority	Description
High	Viewing nearby carparks is a central role in the app's functionality.

4.4.2 Stimulus/Response Sequences

Use Case ID:	#2-3		
Use Case Name:	View Nearby Carparks from a Location		
Created By	Solis Aaron Mari Santos	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (Initiating), Mapbox API, LTA DataMall and HDB API
Description	Users can find carparks with vacancies within 500m near a of a specific location
Preconditions	<ol style="list-style-type: none"> 1. User has GPS and internet access turned on 2. User is located in Singapore
Postconditions	<ol style="list-style-type: none"> 1. User may click on a certain carpark to trigger the Search For a Location functionality (4.3) with that carpark's address 2. Each carpark selected generates a route to that location and leaves a persistent marker on the map to

	allow users to compare relative carpark locations
Priority	High
Frequency of use	High
Flow of events	<ol style="list-style-type: none"> 1. User opens app and application immediately triggers the search for nearby carpark within 500m. 2. System will fetch vacancy and pricing data from LTA DataMall and HDB API, and routing from Mapbox API. 3. System will obtain 10 nearby carpark after fetching the data. 4. Pop-up box on the map will display 3 of the 10 nearby carpark <ol style="list-style-type: none"> a. Users may choose to expand the pop-up to see all 10 nearby carpark. b. Each carpark will display its price and lot information
Alternative flows	<ol style="list-style-type: none"> 1. User enters a location and triggers the search for nearby carpark within 100m. 2. System will fetch vacancy and pricing data from LTA DataMall and HDB API, and routing from Mapbox API. 3. System will obtain 10 nearby carpark after fetching the data. 4. Pop-up box on the map will display 3 of the 10 nearby carpark <ol style="list-style-type: none"> a. Users may choose to expand the pop-up to see all 10 nearby carpark. b. Each carpark will display its price and lot information
Exceptions	<ol style="list-style-type: none"> 1. There are no nearby carpark within 500m 2. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown. 3. There is no lot information for a carpark <ol style="list-style-type: none"> a. Carpark with no lot information will show a No information available line 4. There is no price information for a carpark <ol style="list-style-type: none"> a. Carpark with no price information will show a No information available line

Includes	<i>Search for a Location</i> <i>Get Navigation Route</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.4.3 Functional Requirements

REQ-1: The system must automatically search for nearby car parks upon application initialisation.

REQ-2: The system must be able to fetch data (availability of lots, price, navigation route) from backend APIs.

REQ-2.1: The system must retrieve real-time vacancy and price data from LTA DataMall and HDB API.

REQ-2.2: The system must fetch routing information from Mapbox API.

REQ-2.3: The system must display appropriate error messages when it fails to retrieve data from API calls.

REQ-3: The system must be able to find nearby car parks based on user location.

REQ-3.1: The system must search within 100m radius of user's location

REQ-3.1.1: If there are no car parks within 100m radius, the system will search for nearest car parks beyond 100m radius.

REQ-3.1.2: If there are car parks within 100m radius, the system must identify and sort the 10 nearest car parks based on distance, from nearest to furthest.

REQ-4: The system must display the nearby car parks' information on user interface.

REQ-4.1: The system must display the initial-pop-up of 3 nearest car parks.

REQ-4.1.1: The system must display the vacancy information for each car park.

REQ-4.1.2: The system must display the price per hour for each car park.

REQ-4.1.3: The system must display the name of each car park.

REQ-4.1.4: The system must allow users to navigate to each car park.

REQ-4.1.4.1: The system must display the navigation route to the chosen car park.

4.5 Save and Load Bookmarks

4.5.1 Description and Priority

Users are able to save locations or car parks they visit regularly to a bookmarks page.

Overall Priority	Description
Medium to High	Users will find it more convenient to be able to see frequented locations and carparks.

4.5.2 Stimulus/Response Sequences

Use Case ID:	#3-1		
Use Case Name:	Save and Load Bookmarks		
Created By	Sun Si Tong	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (initial actor)
Description	Users can bookmark locations on the app for search convenience. Users can also view a list of bookmarked locations that they have previously added.
Preconditions	<ol style="list-style-type: none"> 1. User must have internet access on their phone 2. User allows cookie access
Postconditions	<ol style="list-style-type: none"> 1. A new bookmark is added OR 2. A bookmark is deleted OR
Priority	Medium
Frequency of use	Medium
Flow of events	<ol style="list-style-type: none"> 1. User selects “Bookmark” tab on the UI. 2. System retrieves any bookmarks saved by the user from server-side database and displays them on the UI. 3. User searches for a location on the map using extended use case ‘<i>Search for a Location</i>’. 4. User selects “Add Bookmark” button on the UI. 5. System saves the bookmarks added by the user to server-side database. 6. System displays the new bookmark on the UI.
Alternative flows	<ol style="list-style-type: none"> 1. User saves bookmarks from the Main Page <ol style="list-style-type: none"> e. User searches for a location on the map using

	<p>extended use case ‘<i>Search for a Location</i>’.</p> <ol style="list-style-type: none"> f. System displays nearby carpark to the location g. User selects “Add Bookmark” button for a specific carpark h. System saves the bookmarks added by the user to server-side database. <ol style="list-style-type: none"> 2. User removes an existing bookmark <ol style="list-style-type: none"> a. User selects “Remove Bookmark” button on the UI. b. System removes the bookmark from the server-side database, and updates the UI correspondingly.
Exceptions	<ol style="list-style-type: none"> 1. User does not allow cookie access <ol style="list-style-type: none"> a. If the user saves a bookmark and refreshes the page, the saved bookmark will not be reflected on the app. 2. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown.
Includes	<i>Search for a Location</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.5.3 Functional Requirements

REQ-1: System must verify prerequisite conditions.

REQ-1.1: System must check internet connectivity.

REQ-1.1.1: System must notify user if internet is unavailable.

REQ-1.1.2: System must provide retry option when connection is restored.

REQ-1.2: System must verify cookie access permission.

REQ-1.2.1: System must prompt user to enable cookie access if disabled.

REQ-1.2.2: System must explain importance of cookie access for bookmark functionality.

REQ-2: The system must display a bookmark tab in the main user interface.

REQ-3: The system must handle bookmark addition.

REQ-3.1: The system must allow users to save locations to bookmarks directly from the location search function at the navigation interface.

REQ-3.2: The system must handle bookmark storage

REQ-3.2.1: The system must save bookmarks to the database.

REQ-3.2.2: The system must update UI to show new bookmarks saved.

REQ-3.3: The system must not save duplicate bookmarks.

REQ-3.3.1: When the user tries to save a duplicate bookmark from bookmark interface, the bookmark will not be added

REQ-3.3.2: When the user tries to save a duplicate bookmark from the navigation interface, it will be indicated that the location has already been added.

REQ-4: The system must handle bookmark removal.

REQ-4.1: The system must allow user to remove bookmark from the bookmark interface.

REQ-4.2: The system must allow user to remove bookmark from the navigation interface.

REQ-4.3: The system must update the UI to reflect the bookmark has been removed.

REQ-4.4: The system must remove bookmark from the database.

REQ-5: System must handle data persistence.

REQ-5.1: System must maintain bookmark synchronisation.

REQ-5.1.1: System must sync bookmarks across sessions.

REQ-5.2: System must manage cookie-related scenarios.

REQ-5.2.1: System must store bookmark data in cookies.

REQ-5.2.2: System must handle cookie access denial.

4.6 Toggle Bookmark to Navigation

4.6.1 Description and Priority

Users are able to click on bookmarks to toggle location navigation route and find nearby car parks to that bookmarked location

Overall Priority	Description
Medium to High	Users will find it more convenient to be able to get the navigation routes to their bookmarked locations

4.6.2 Stimulus/Response Sequences

Use Case ID:	#3-2
--------------	------

Use Case Name:	Toggle Bookmark to Navigation		
Created By	Tan Yu Xiu	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (initial actor)
Description	Users can toggle a bookmark by clicking on the navigation icon next to the display bookmark to enable navigation route calculation to that location
Preconditions	<ol style="list-style-type: none"> 3. User must have internet access on their phone 4. User allows cookie access 5. User may have saved bookmarks before
Postconditions	<ol style="list-style-type: none"> 3. User is routed to the navigation page, route is shown
Priority	Medium
Frequency of use	Medium
Flow of events	<ol style="list-style-type: none"> 4. User selects “Bookmark” tab on the UI. 5. System retrieves any bookmarks saved by the user from server-side database and displays them on the UI. 7. User clicks on the “navigate” button next to each Bookmark card. 8. User is routed to the main navigation page 9. Nearby carparks are shown
Alternative flows	
Exceptions	<ol style="list-style-type: none"> 1. User does not allow cookie access <ol style="list-style-type: none"> a. If the user saves a bookmark and refreshes the page, the saved bookmark will not be reflected on the app. 2. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown.
Includes	<i>Save and Load Bookmarks</i> <i>Get Navigation Routes</i> <i>Show Nearby Carparks</i>
Special requirements	- Stable Internet Connectivity

Assumptions	- None
Notes and Issues	- None

4.6.3 Functional Requirements

REQ-1: System must verify prerequisite conditions.

REQ-1.1: System must check internet connectivity.

REQ-1.1.1: System must notify user if internet is unavailable.

REQ-1.1.2: System must provide retry option when connection is restored.

REQ-1.2: System must verify cookie access permission.

REQ-1.2.1: System must prompt user to enable cookie access if disabled.

REQ-1.2.2: System must explain importance of cookie access for bookmark functionality.

REQ-2: The system must display a bookmark tab in the main user interface.

REQ-3: The system must handle bookmark selection.

REQ-3.1: The system must showcase a navigation to button next to each bookmarked location

REQ-3.1.1: The system must allow users to click on bookmark's navigate to button

REQ-3.2: The system must handle the routing to the main map page

REQ-3.3: The system must load in the bookmark's coordinates into the search function of the map page

REQ-3.3.1: The map must be recentered and zoomed into that location

REQ 3-4: The system must generate the route from the user's current location to said bookmarked location.

REQ-3.5: The system must display the list of nearby carparks to the bookmarked location upon loading the route

4.7 Toggle Carpark Sorting Filter

4.7.1 Description and Priority

Users are able to toggle the filter by which nearby carparks are sorted by. These filters are distance, number of vacancies, and pricing rates respectively.

Overall Priority	Description
Medium	Users will not be likely to toggle the sorting filter much.

4.7.2 Stimulus/Response Sequences

Use Case ID:	#4-1		
Use Case Name:	Toggle Carpark Sorting Filter		
Created By	Quek Jun Siong	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (initial actor)
Description	Users can toggle sorting filter for carpark.
Preconditions	1. User must have internet access on their phone
Postconditions	1. User's search options are updated to reflect changes in the search results
Priority	Medium
Frequency of use	Low
Flow of events	<ol style="list-style-type: none"> 1. System displays nearby carpark to user's current location, or a searched location, using extended use case 'View Nearby Carpark'. 2. User selects "Settings" tab on the UI. 3. User selects sorting filter out of 3 options - Sort by Distance, Sort by Vacancies, and Sort by Pricing Rates. 4. Nearby carpark will be sorted depending on the filter.
Alternative flows	-
Exceptions	<ol style="list-style-type: none"> 1. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown. 2. There are no lot information for a carpark <ol style="list-style-type: none"> a. Carpark with no lot information will be pushed to the back in the sort by lot filter b. If all carpark have no price, the sort by distance will be used 3. There are no price information for a carpark <ol style="list-style-type: none"> a. Carpark with no price information will be pushed to the back in the sort by price filter b. If all carpark have no price, the sort by

	distance will be used
Includes	<i>View Nearby Carparks</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.7.3 Functional Requirements

REQ-1: System must integrate with carpark display functionality.

REQ-1.1: System must link with "View Nearby Carparks from a location" feature.

REQ-1.1.1: System must access current carpark list.

REQ-1.1.2: System must maintain real-time data synchronisation.

REQ-2: System must provide filter interface.

REQ-2.1: System must display "Settings" tab in UI.

REQ-2.2: System must show sorting options.

REQ-2.2.1: System must provide "Sort by Distance" option.

REQ-2.2.2: System must provide "Sort by Vacancies" option.

REQ-2.2.3: System must provide "Sort by Pricing Rates" option.

REQ-2.3: System must indicate currently active filter.

REQ-2.3.1: System must highlight selected filter option.

REQ-3: System must implement sorting functionality.

REQ-3.1: System must sort by distance.

REQ-3.1.1: System must calculate distance from current/searched location.

REQ-3.1.2: System must arrange carparks in ascending order of distance.

REQ-3.2: System must sort by vacancies.

REQ-3.2.1: System must retrieve real-time vacancy data.

REQ-3.2.2: System must arrange carparks in descending order of vacancies.

REQ-3.3: System must sort by pricing rates.

REQ-3.3.1: System must compare carpark pricing data.

REQ-3.3.2: System must arrange carparks in ascending order of rates.

REQ-4: System must handle display updates.

REQ-4.1: System must refresh carpark list display.

REQ-4.1.1: System must update list immediately after filter change.

REQ-4.1.2: System must maintain smooth transition between sorts.

REQ-4.2: System must preserve essential information.

REQ-4.2.1: System must maintain all carpark details during sort.

REQ-4.2.2: System must retain user's view position after sorting.

REQ-5: System must maintain filter state.

REQ-5.1: System must save user's filter preference.

REQ-5.1.1: System must store last used filter.

REQ-5.1.2: System must apply saved filter on app restart.

4.8 Toggle Vehicle Type

4.8.1 Description and Priority

Users are able to toggle their preferred vehicle type. Available options are fuel cars, motorcycles, and EV cars.

Overall Priority	Description
Medium	Most users are likely to be fuel car drivers, and likely only drive one vehicle type, thus being unlikely to have to toggle vehicle types.

4.8.2 Stimulus/Response Sequences

Use Case ID:	#4-2		
Use Case Name:	Toggle Vehicle Type		
Created By	Quek Jun Siong	Last Updated By:	Tan Yu Xiu
Date Created	27/8/2024	Date Last Updated:	15/11/2024

Actor	User (initial actor), LTA DataMall and HDB API
Description	Users can toggle their preferred vehicle type .
Preconditions	1. User must have internet access on their phone

Postconditions	1. User's vehicle options are updated to reflect changes in the search results
Priority	Medium
Frequency of use	Low
Flow of events	<ol style="list-style-type: none"> 1. System displays nearby carpark to user's current location, or a searched location, using extended use case '<i>View Nearby Carparks</i>'. 2. User selects "Settings" tab on the UI. 3. User selects preferred vehicle type out of 3 options - Fuel Car, Motorcycle, or EV car. 4. System sends a request to LTA DataMall and HDB API to fetch carpark results that match the chosen vehicle type. 5. System returns number of vacancies depending on the chosen vehicle type.
Alternative flows	-
Exceptions	<ol style="list-style-type: none"> 4. User does not have internet access <ol style="list-style-type: none"> a. An empty screen will be rendered and only the top-navbar will be shown. 5. There are no lot information for motorcycle parking <ol style="list-style-type: none"> a. Carparks with no lot information will be pushed to the back in the sort by lot filter b. If all carparks have no price, the sort by distance will be used 6. There are no price information for motorcycle parking <ol style="list-style-type: none"> a. Carparks with no price information will be pushed to the back in the sort by price filter b. If all carparks have no price, the sort by distance will be used
Includes	<i>View Nearby Carparks</i>
Special requirements	- Stable Internet Connectivity
Assumptions	- None
Notes and Issues	- None

4.8.3 Functional Requirements

REQ-1: System must integrate with carpark display system.

REQ-1.1: System must link with "View Nearby Carparks" feature.

REQ-1.1.1: System must access current carpark list.

REQ-1.1.2: System must maintain vehicle type filters.

REQ-2: System must provide vehicle type interface.

REQ-2.1: System must display "Settings" tab in UI.

REQ-2.2: System must show vehicle type options.

REQ-2.2.1: System must provide "Fuel Car" option.

REQ-2.2.2: System must provide "Motorcycle" option.

REQ-2.2.3: System must provide "EV Car" option.

REQ-2.3: System must indicate active vehicle type.

REQ-2.3.1: System must highlight selected vehicle option.

REQ-2.3.2: System must persist vehicle selection.

REQ-3: System must handle API interactions.

REQ-3.1: System must communicate with LTA DataMall API.

REQ-3.1.1: System must send vehicle type parameter.

REQ-3.1.2: System must handle API response data.

REQ-3.2: System must communicate with HDB API.

REQ-3.2.1: System must send vehicle type parameter.

REQ-3.2.2: System must handle API response data.

REQ-3.3: System must manage API errors.

REQ-3.3.1: System must implement error handling for API failures.

REQ-3.3.2: System must provide retry mechanisms.

REQ-4: System must process vacancy data.

REQ-4.1: System must filter vacancies by vehicle type.

REQ-4.1.1: System must show car lots for fuel car and EV selection.

REQ-4.1.2: System must show motorcycle lots for motorcycle selection.

REQ-4.2: System must update vacancy display.

REQ-4.2.1: System must refresh vacancy numbers after type change.

REQ-4.2.2: System must indicate lot type on display.

5. Non-Functional Requirements

5.1 Performance Requirements

1. Each page, including all the carpark information on the pages, should be loaded within 3 seconds.
2. The search results should be loaded within 1 second upon valid user input

3. The system must be able to display the navigation route within 3 seconds
4. The system must be able to synchronise data from past bookmarks saved within 3 seconds.

5.2 Security Requirements

1. User's location and navigation routes data will be anonymised, or not stored at all for privacy purposes.
2. User input will be validated to prevent SQL injection, XSS attacks and similar security breaches.

5.3 Software Quality Attributes

1. The generated carpark availability and rates must be at least 95% accurate to ensure that the users have a safe ride.
2. The navigation route must not include roads or bridges that do not exist anymore at least 99% of the time.
3. The system should be modular such that individual components (such as map display, carpark info, etc) have minimal impact on other components and can be updated independently.
4. The code will always be accompanied with relevant comments when necessary for code interpretability, reusability, collaboration and maintenance.
5. The API endpoints implemented by the backend server will always be accompanied with a documentation to indicate the endpoint url, required request body, and expected response body. The API documentation can be viewed [here](#).

6. Other Requirements

i. Database Requirements: The application's data handling mechanisms must support transactional integrity for operations, such as bookmarking carpark and displaying navigation routes, to ensure data consistency. The system should be capable of efficiently handling requests for nearby carpark information and updating bookmark data, even under high demand, such as during peak hours with hundreds of simultaneous users.

ii. Reuse Objectives: Where possible, the application should be designed to allow for the reuse of core components, such as the map display, navigation functionality, and data retrieval services. This modular design approach will enable easy integration of these components in future updates.

Rationale:

These requirements ensure that ParkIt! is secure and scalable, even without a user account system. By leveraging cookies for data storage and designing reusable components, the application can provide a seamless user experience while being adaptable to future enhancements.

7. Appendix A: Data Dictionary

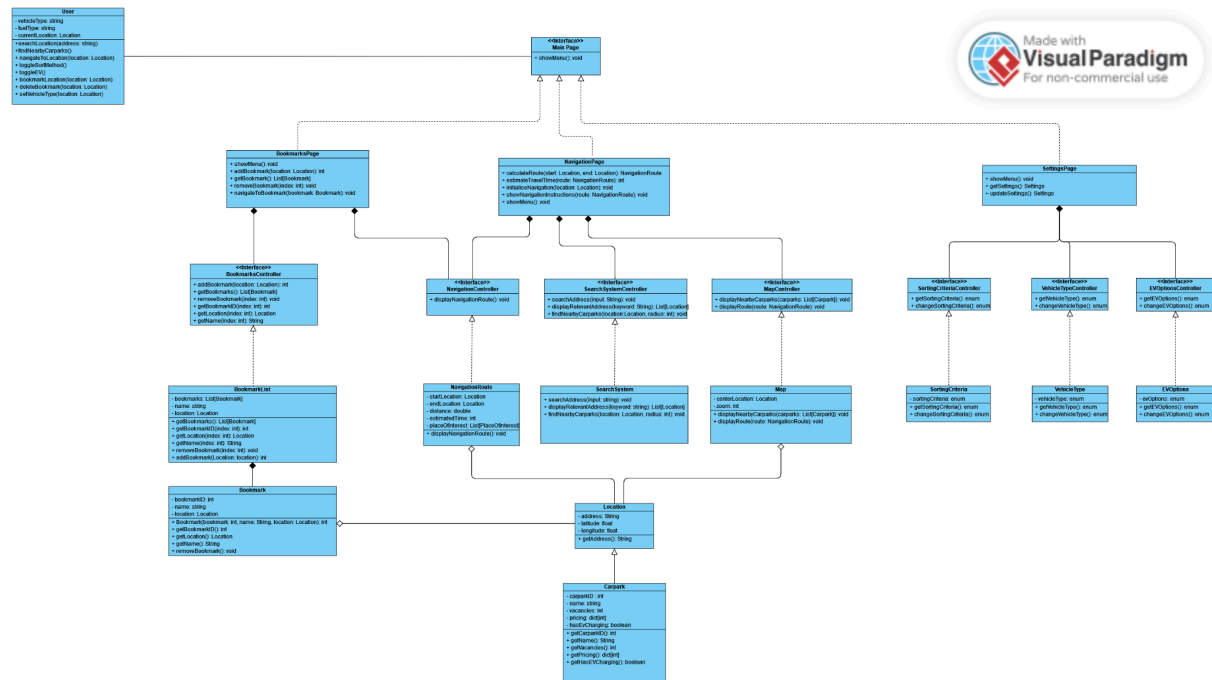
Term	Definition
Carpark	A designated area for parking vehicles, which may include public parking spaces and/or private parking spaces
Vacancy	An available parking space within a carpark
Electric Vehicle (EV)	A vehicle that uses one or more electric motors for propulsion, requiring specific charging infrastructure.
User Location	The current geographical position of the app user, which can be automatically detected or manually entered.
User Destination	The intended geographical position the user is heading towards, which may or may not be a carpark
Search Radius	The circular area around a specified location within which the app searches for available carparks. Default is set to 100 metres.
Carpark Capacity	The total number of parking spaces available in a carpark
Carpark Remaining Capacity	The number of vacancies left in a carpark
Navigation Route	A series of directions guiding the user from User Location to a User Destination
Bookmark Page	A page where a list of Bookmarked Locations are shown.
Bookmarked Locations	A place that a user has saved for quick access in future searches

8. Appendix B: Analysis Models

Class Diagram

High Definition image can be found here:

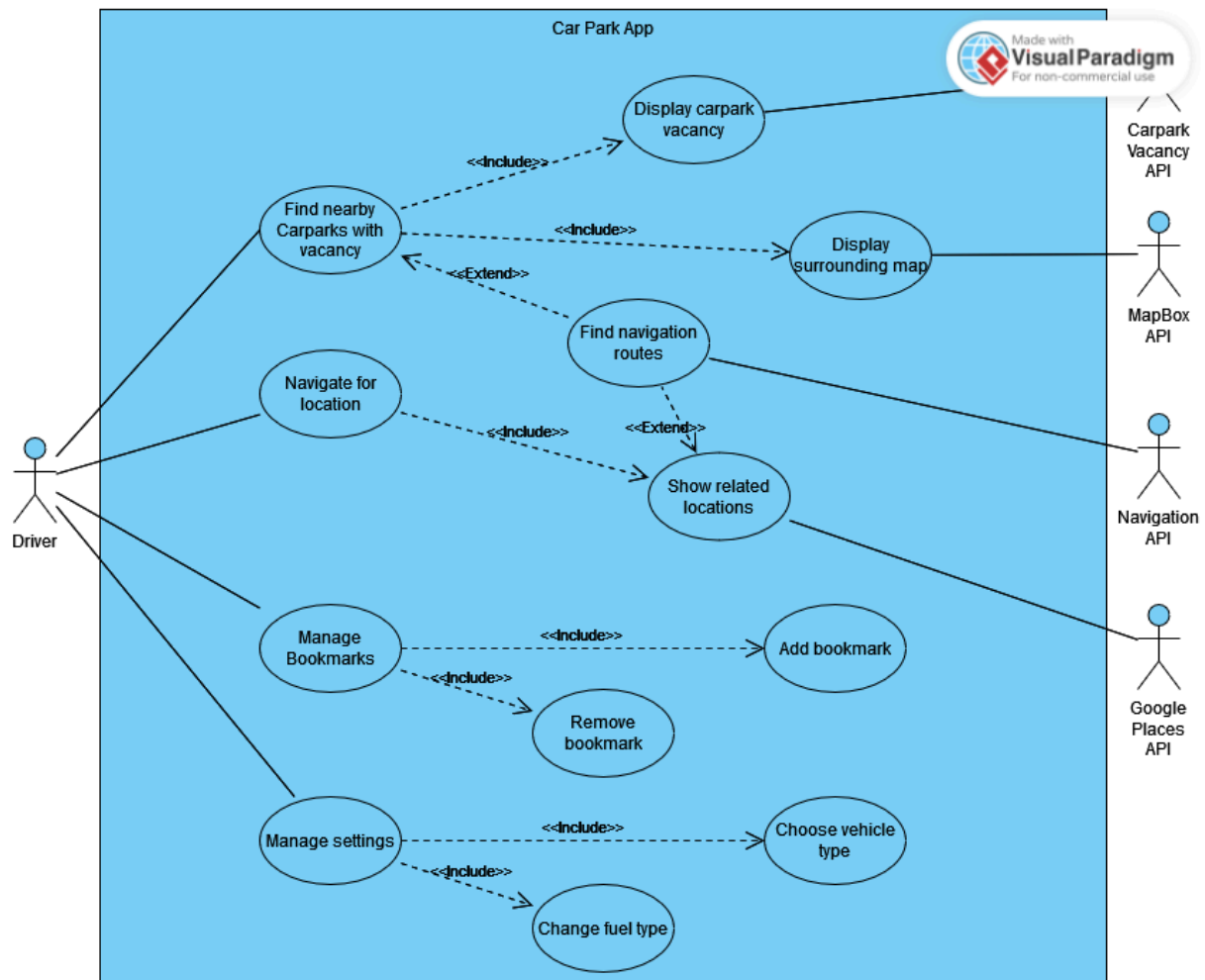
https://drive.google.com/file/d/1YWiMPLfxvkCu-IMx5FAfmHjKSulHxJre/view?usp=drive_link



Use Case Diagram

High Definition image can be found here:

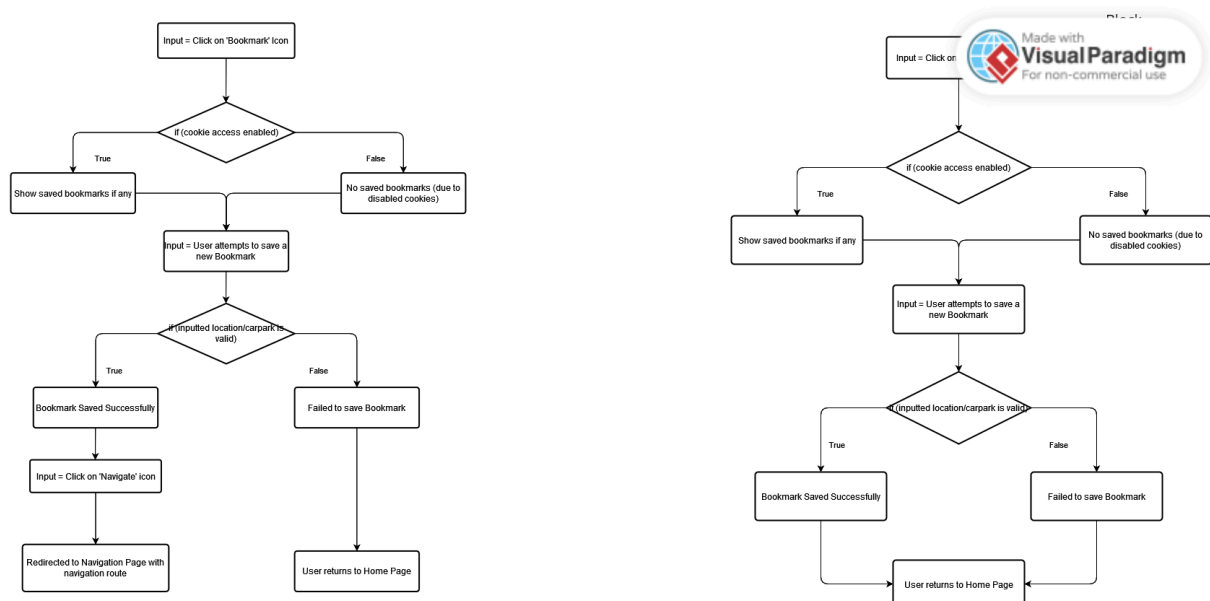
https://drive.google.com/file/d/1S0JGu5DC_3ila8gU26CkXpEh9NKLyMf7/view?usp=drive_link



Control Flow Graph for bookmarks

High Definition image can be found here:

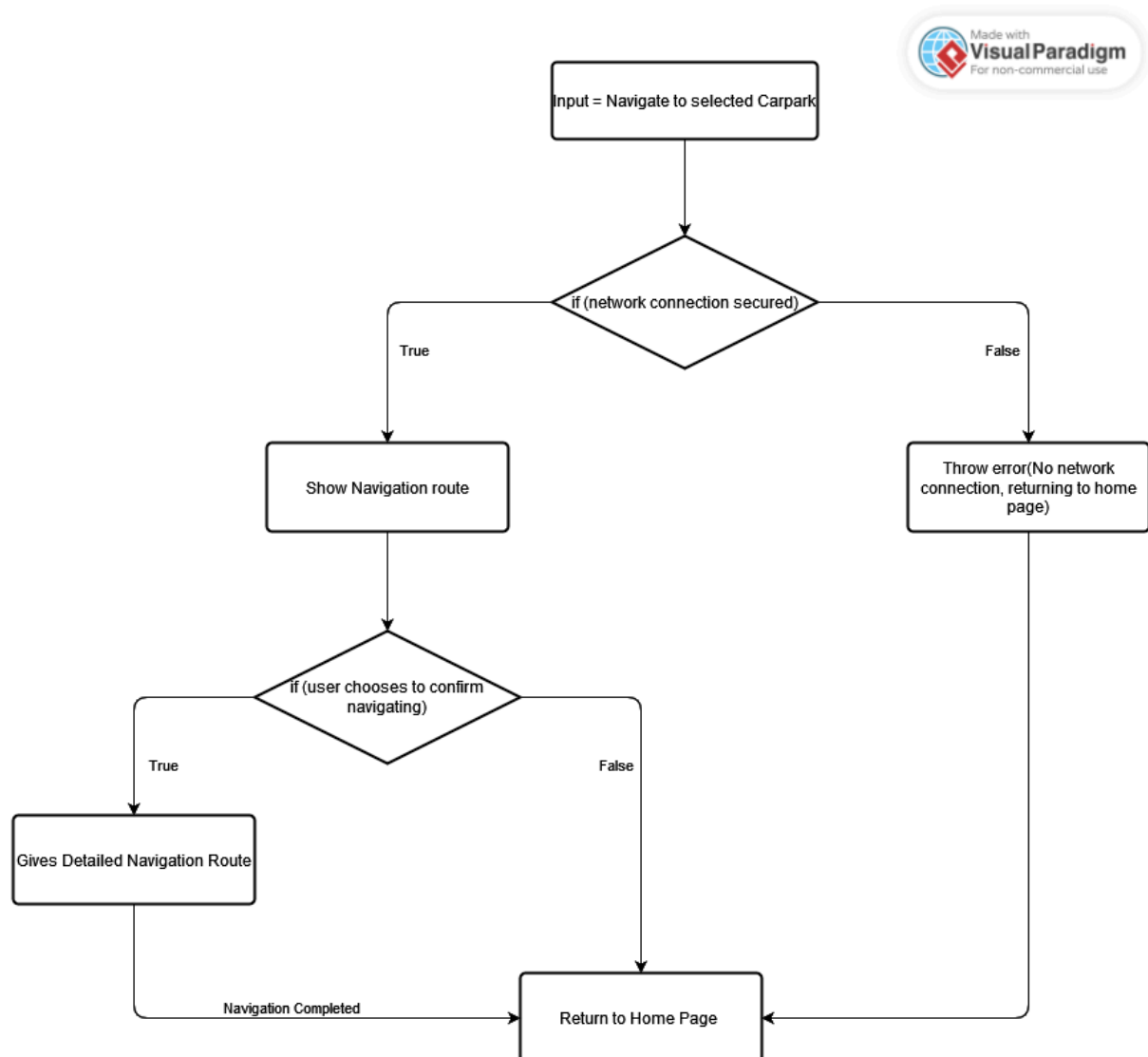
https://drive.google.com/file/d/1OQKJpjyqmnN5FKy62x2ZRmVilU0X5FEo/view?usp=drive_link



Control Flow Graph for navigation

High Definition image can be found here:

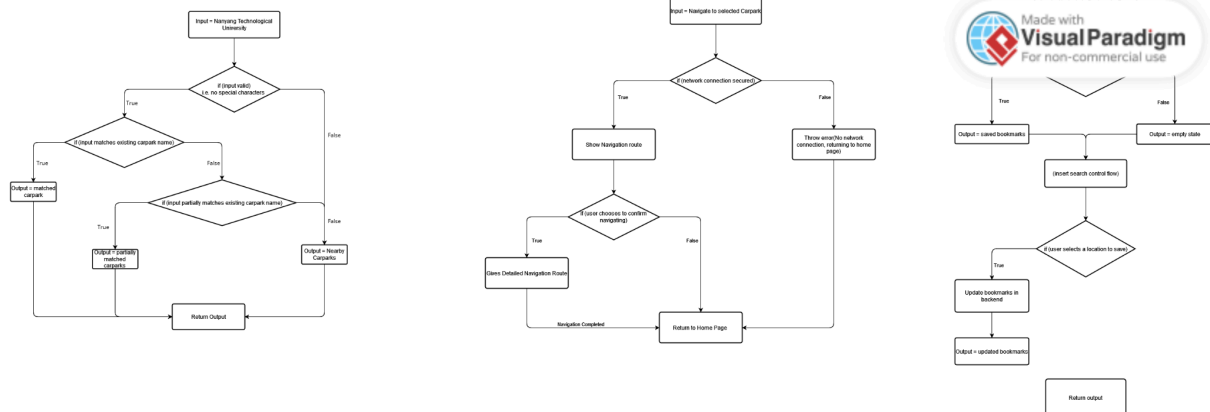
https://drive.google.com/file/d/1szJN5jvaQjmHEiETG3a-lyusb-pS_D6z/view?usp=drive_link



Control Flow Graph for searching

High Definition image can be found here:

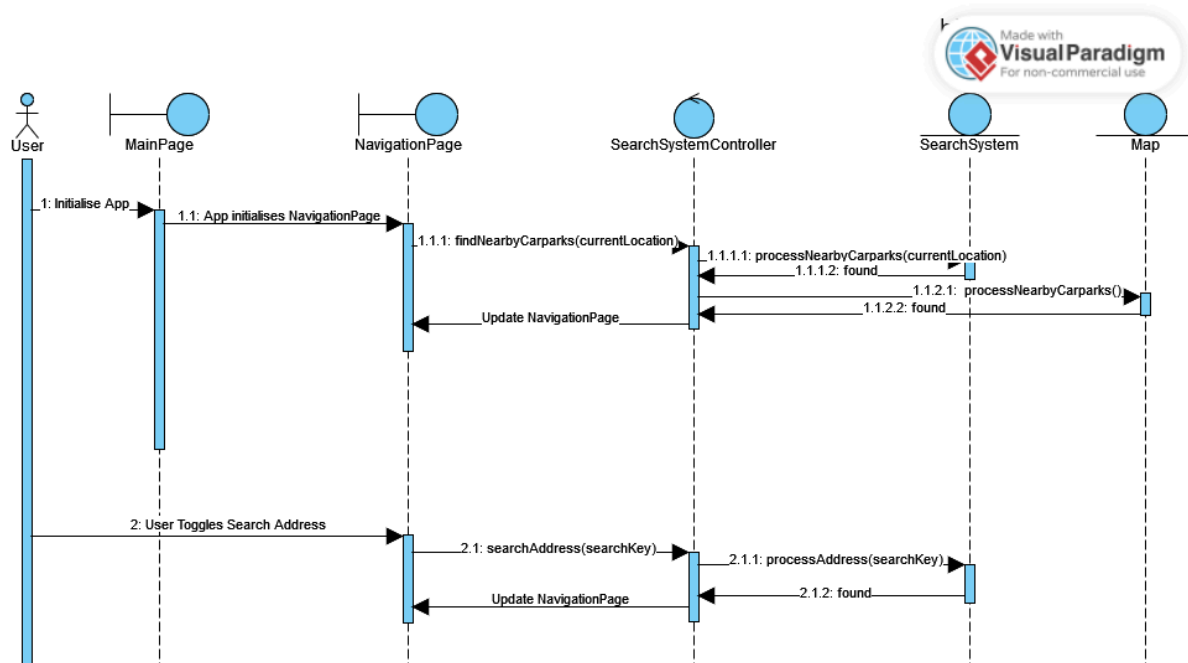
https://drive.google.com/file/d/1CoBcci4H9e2AA3FQsFt0IxZqv3Wtvm2/view?usp=drive_link



Sequence Diagram for searching

High Definition image can be found here:

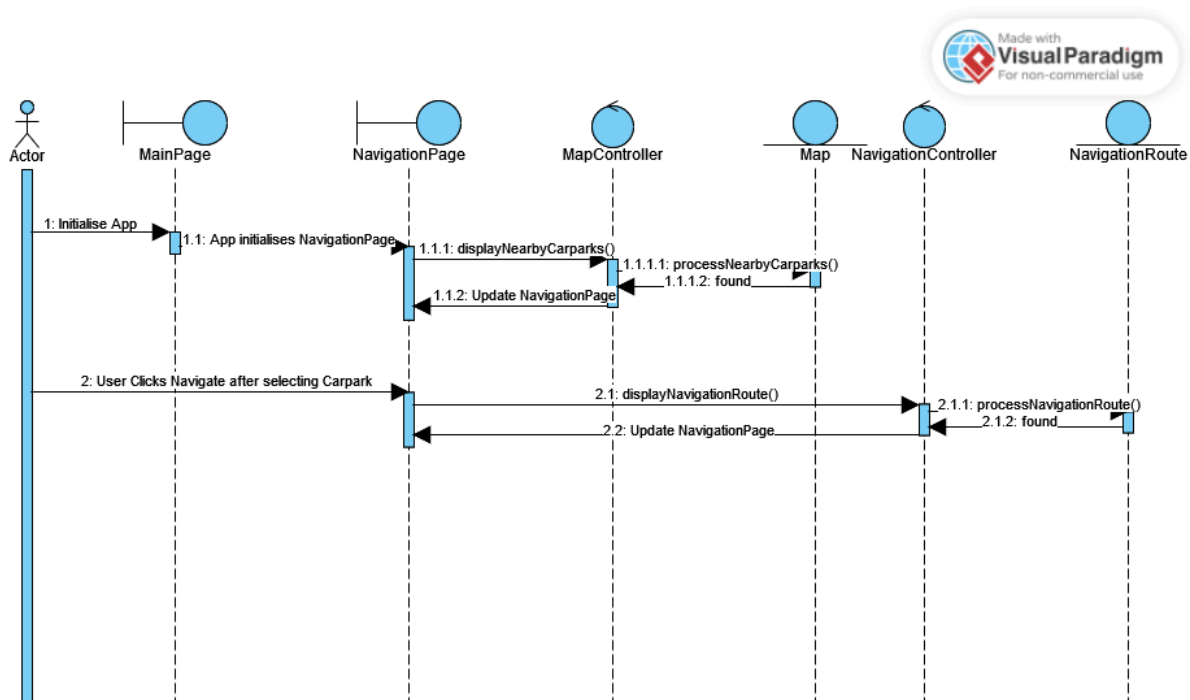
https://drive.google.com/file/d/1-N-tTm9B0mLAXD8ygf1yBSpkGSodrrdR/view?usp=drive_link



Sequence Diagram for navigation

High Definition image can be found here:

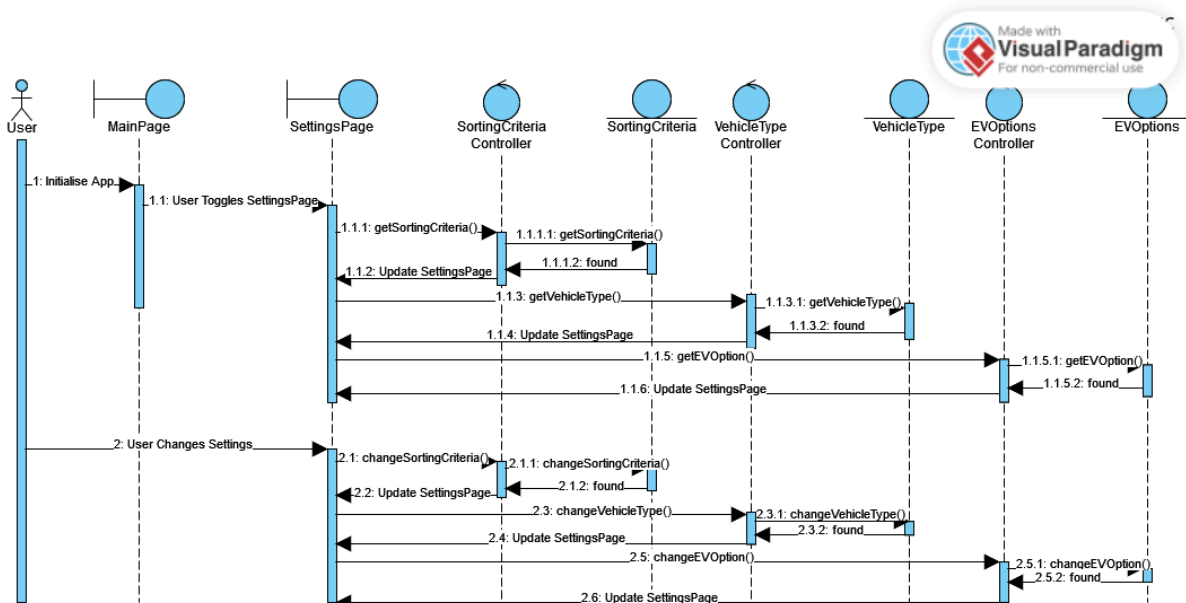
https://drive.google.com/file/d/1KTlwnMhlpq-aJ5kpzg8HhMBApUOiT7SK/view?usp=drive_link



Sequence Diagram for settings

High Definition image can be found here:

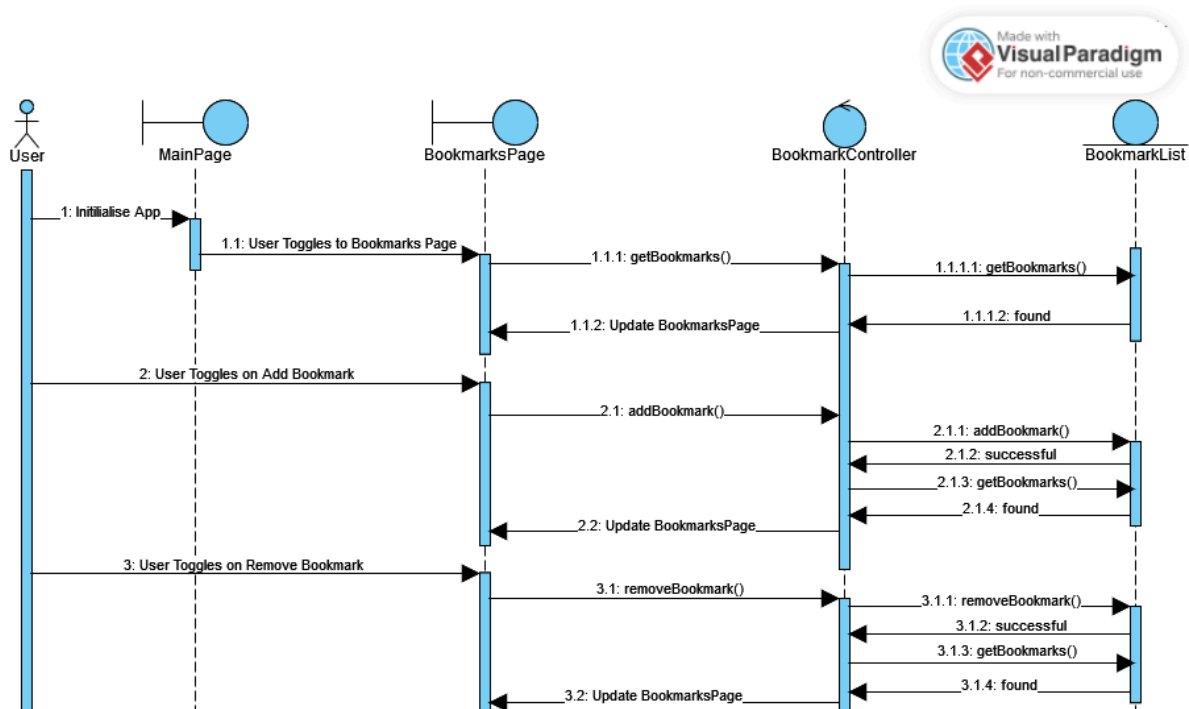
https://drive.google.com/file/d/19UQhe-X0KY25PP40Kg-djOCgL-uqUGTs/view?usp=drive_link



Sequence Diagram for bookmarks

High Definition image can be found here:

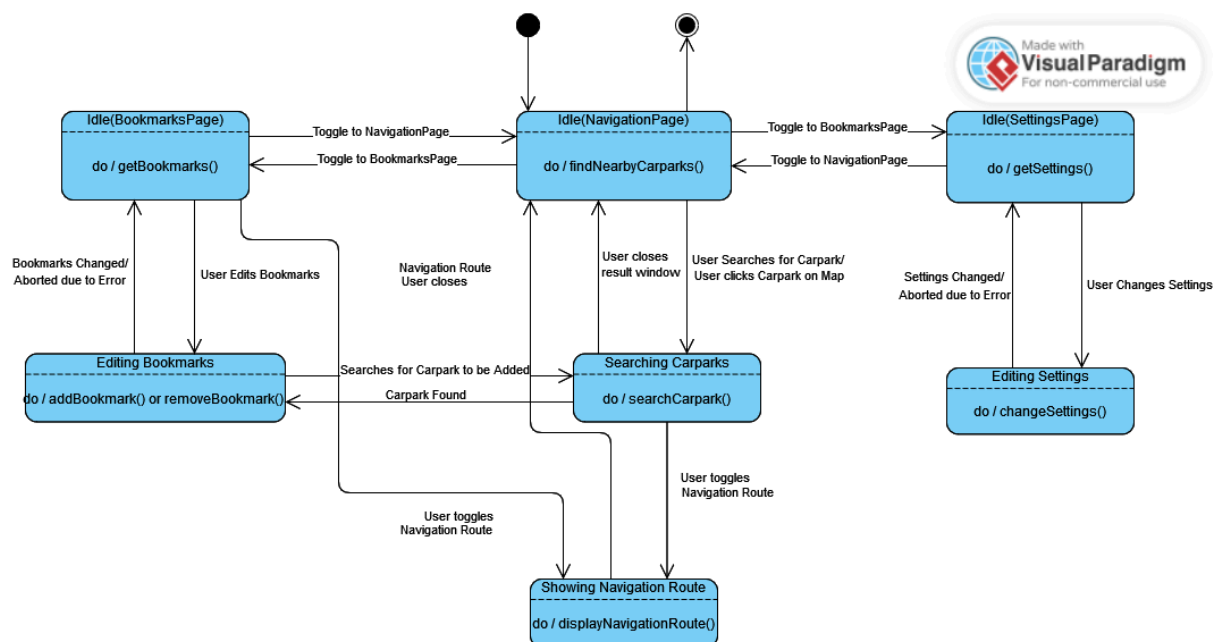
[BookmarkPage Sequence Lab 3.png](#)



Dialog Map

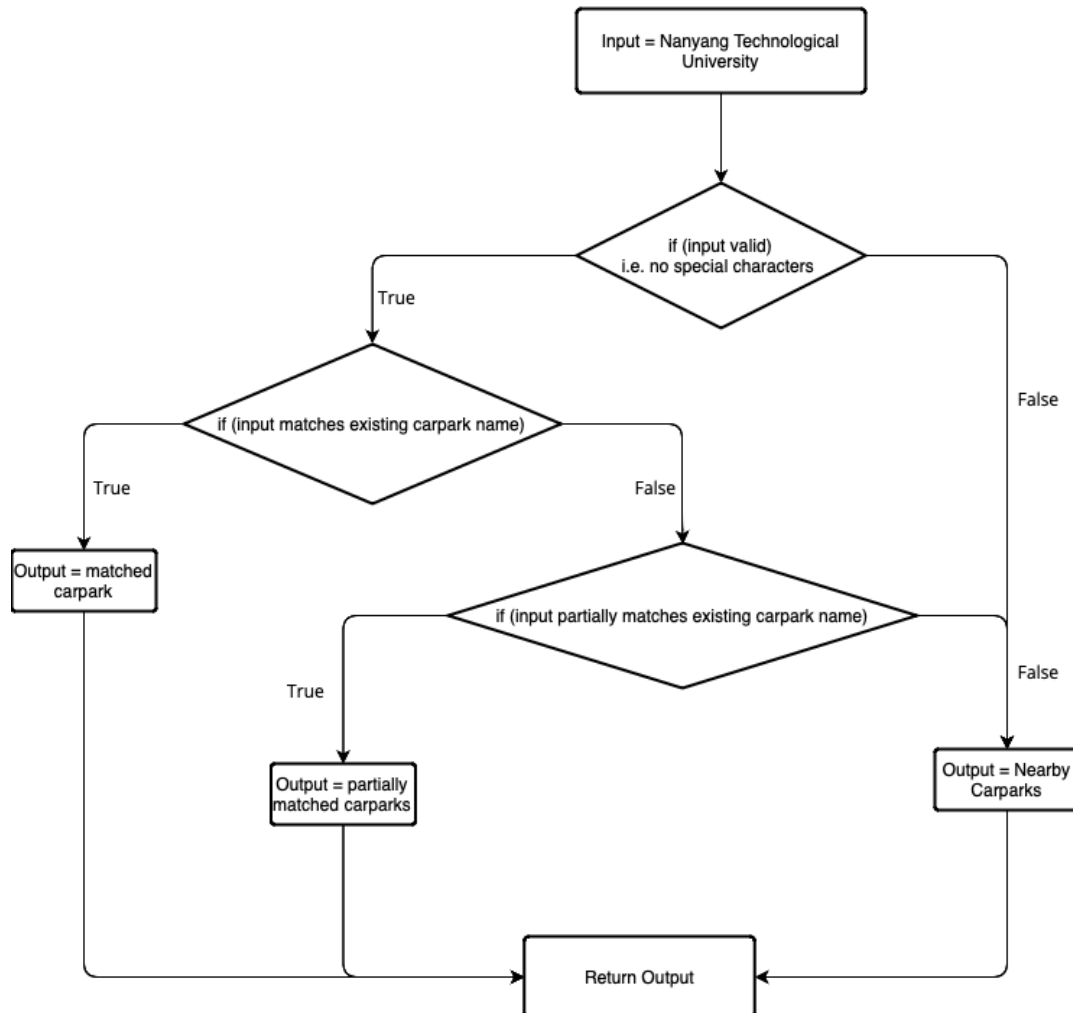
High Definition image can be found here:

https://drive.google.com/file/d/1g5XhXYR2kvW46Lzv7rox-otn5YAvH1Kg/view?usp=drive_link



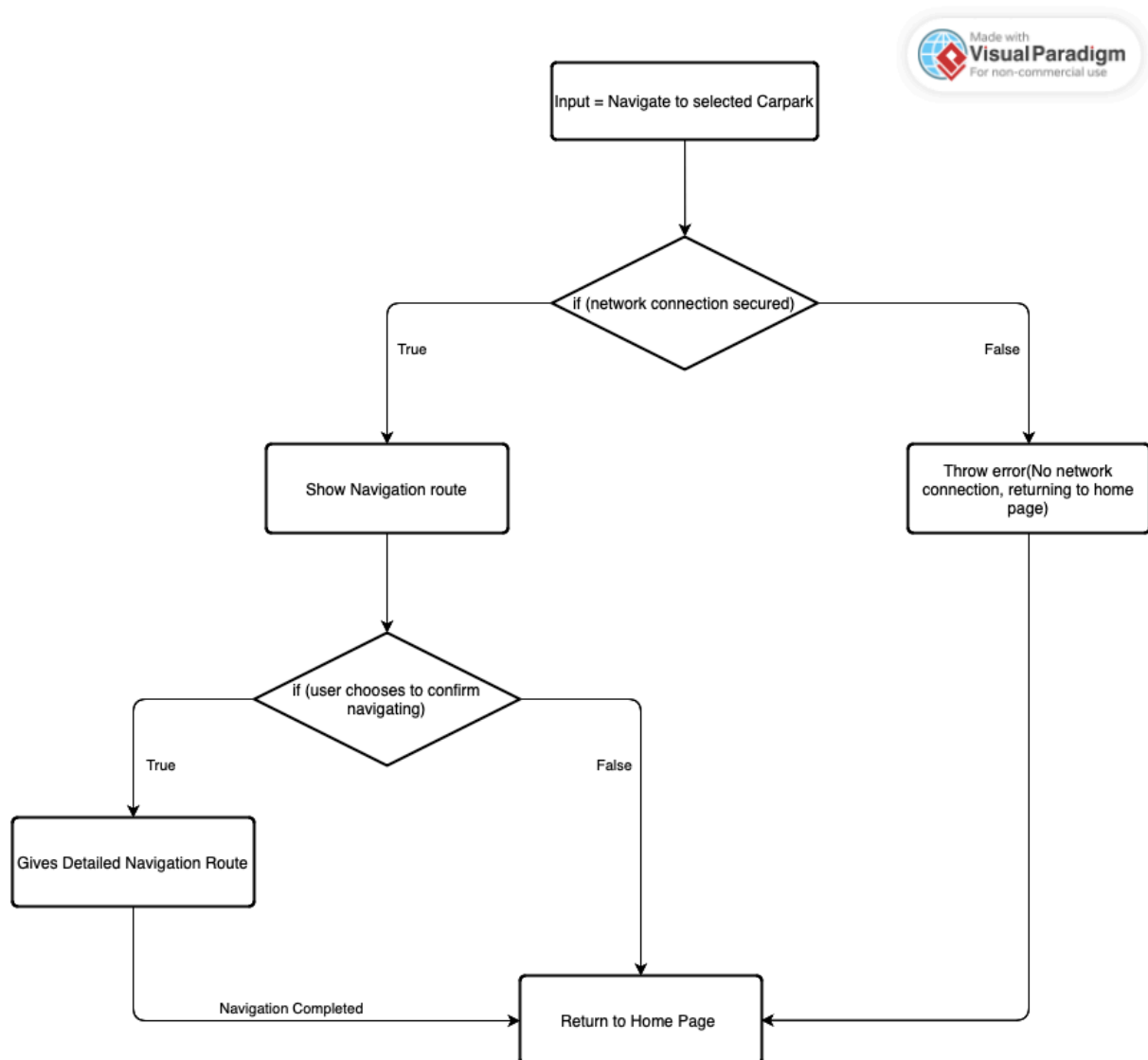
White Box Testing

Search Carparks



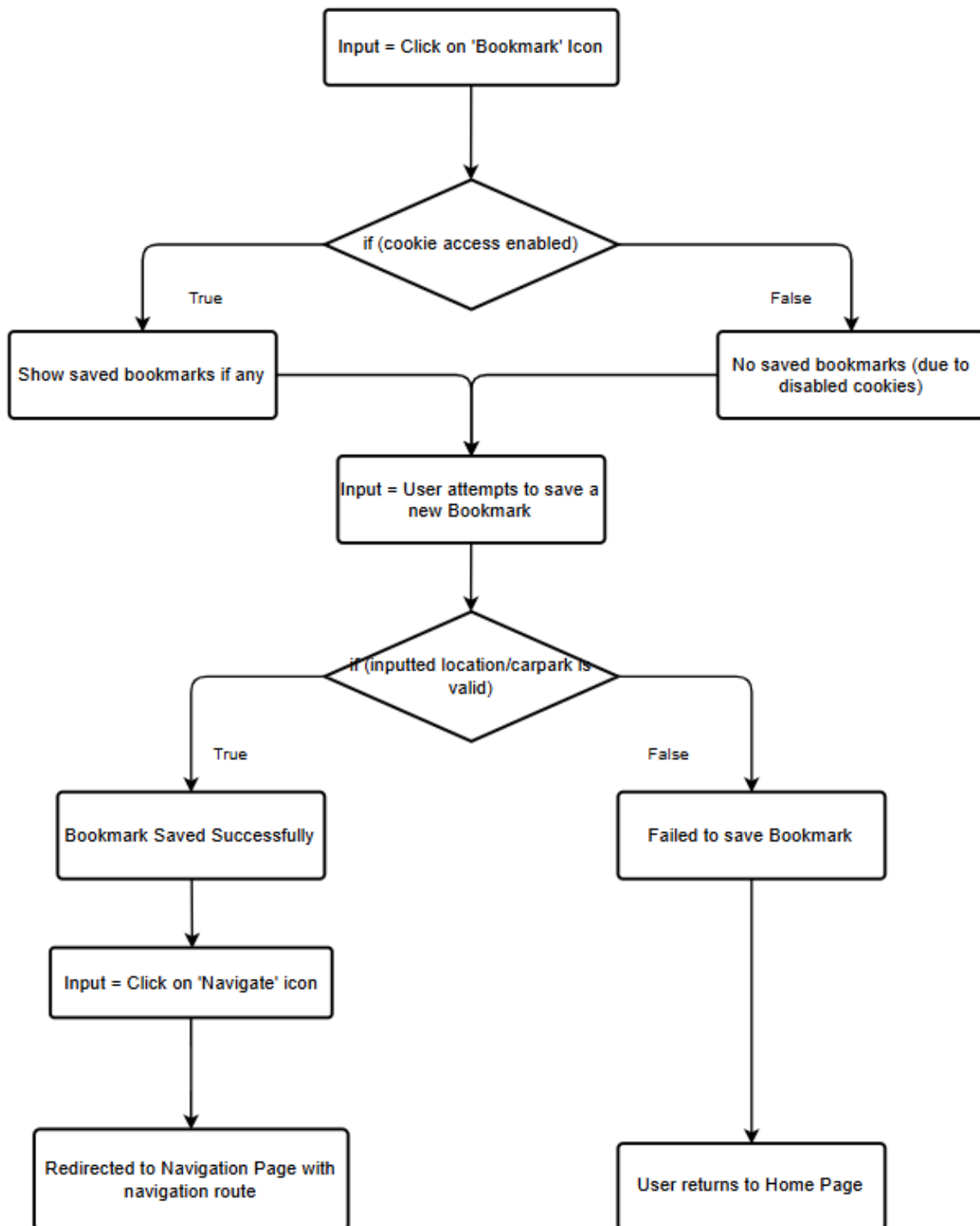
Test Case ID	Search-1		Test Case Priority	High	
Test Case Description	Searching - Positive Test Case				
Prerequisite	Stable Internet Connection		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Launch Application	-	Landing Page	Landing Page	Pass
2	Wait for Home Page to load	-	Home Page	Home Page	Pass
3	Enter location/ carpark into searchbox	Valid location input	Searched location on Map	Searched location on Map	Pass
Test Case Status					
Comments					

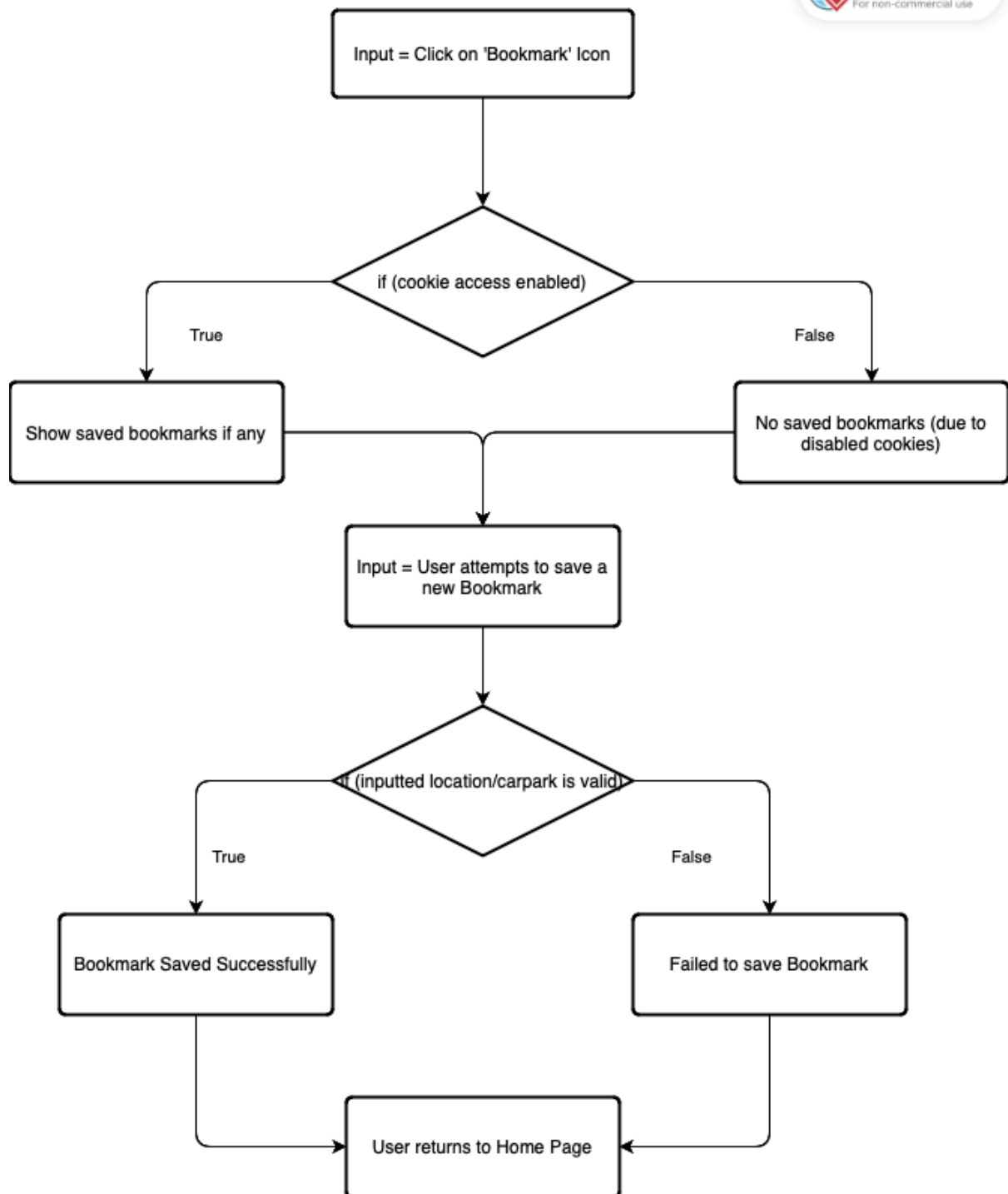
Test Case ID	Search-2		Test Case Priority	High	
Test Case Description	Searching - Negative Test Case				
Prerequisite	Stable Internet Connection		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Launch Application	-	Landing Page	Landing Page	Pass
2	Wait for Home Page to load	-	Home Page	Home Page	Pass
3	Enter invalid location/ carpark into searchbox	Inputs of the type (Empty string, Special characters, Coordinates)	No results	No results	Pass
Test Case Status					
Comments					

Navigate to Carpark

Test Case ID	Navigate-1		Test Case Priority	High	
Test Case Description	Navigating - Positive Test Case				
Prerequisite	Stable Internet Connection and Valid Location/Carpark Input and Location Access Allowed		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Click on Searched Location/Car park	Current Location, Searched Location	Navigation Route to Destination	Navigation Route to Destination	Pass
Test Case Status					
Comments					

Test Case ID	Navigate-2		Test Case Priority	High	
Test Case Description	Navigating - Negative Test Case				
Prerequisite	Stable Internet Connection and Valid Location/Carpark Input		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Deny Location Access when Prompted	-	Home Page	Home Page	Pass
2	Click on Searched Location/Carpark	Searched Location	No Route	No Route	Pass
Test Case Status					
Comments					

Bookmarks



Test Case ID	Bookmark-1		Test Case Priority	High	
Test Case Description	Bookmarking - Positive Test Case				
Prerequisite	Stable Internet Connection and Cookies Enabled and Valid Location/Carpark		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Click on 'Bookmark' Icon	-	Bookmark Page	Bookmark Page	Pass
2	Automaticall y Fetch Saved Bookmarks	Server cookie	Saved Bookmarks (if any)	Saved Bookmarks (if any)	Pass
3	Save New Bookmark	Valid Location/Carpark	Bookmark saved successfully	Bookmark saved successfully	Pass
4	Click on 'Navigate' Icon for Saved Bookmark	-	Navigation Page with Navigation Route	Navigation Page with Navigation Route	Pass
Test Case Status					
Comments					

Test Case ID	Bookmark-2		Test Case Priority	High	
Test Case Description	Bookmarking - Negative Case				
Prerequisite	Stable Internet Connection and Valid Location/Carpark		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Click on 'Bookmark' Icon	-	Bookmark Page	Bookmark Page	Pass
2	Deny cookie access	-	Bookmark Page	Bookmark Page	Pass
3	Automaticaly Fetch Saved Bookmarks	-	No saved bookmarks	No saved bookmarks	Pass
3	Save New Bookmark	Valid Location/Car park	Bookmark saved successfully	Bookmark saved successfully	Pass
Test Case Status					
Comments					

Test Case ID	Bookmark-3		Test Case Priority	High	
Test Case Description	Bookmarking - Negative Test Case				
Prerequisite	Stable Internet Connection and Cookies Enabled		Postrequisite	NA	
Test Execution Steps					
Step No.	Action	Input	Expected Output	Actual Output	Test Result
1	Click on 'Bookmark' Icon	-	Bookmark Page	Bookmark Page	Pass
2	Automaticaly Fetch Saved Bookmarks	Server cookie	Saved Bookmarks (if any)	Saved Bookmarks (if any)	Pass
3	Save New Bookmark	Invalid Location/Carpark of the type <ul style="list-style-type: none">- Empty string- Special characters- Coordinates	Failed to save bookmark	Failed to save bookmark	Pass
Test Case Status					
Comments					

9. References

I. *IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications.*

IEEE Standards Association. (n.d.). Retrieved 17 November, 2024 from

<https://standards.ieee.org/ieee/830/1222/>

II. LTA DataMall, an LTA Open Data Initiative API User Guide and Documentation Version

6.1.1. Land Transport Authority Retrieved 17 November 2024 from

https://datamall.lta.gov.sg/content/dam/datamall/datasets/LTA_DataMall_API_User_Guide.pdf