Software Requirements Specification

for

ParkIT!

Version 1.8 approved

Prepared by

FONG ZHENG FENG VICTOR

JERRAL CHUA FUNG YONG

KEE YU KAI

LIANG JIANPENG

RESWARA ANARGYA DZAKIRULLAH

TIO SHER MIN

Group 20, SCS1, Nanyang Technological University

9 November 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Fong Zheng Feng Victor	26 October 2024	Initial write up for Testing	1.0
Jerral Chua Fung Yong	28 October 2024	Initial write up for Introduction and Overall Description	1.1
Liang Jianpeng	29 October 2024	Initial write up for External Interface Requirements, Functional Requirements, Non-Functional Requirements and Other Requirements	1.2
Tio Sher Min	29 October 2024	Initial write up for Appendix A: Data Dictionary and Appendix B: Analysis Models	1.3
Tio Sher Min	31 October 2024	Update 2.1 Product Perspective	1.4
Jerral Chua Fung Yong	2 November 2024	Update 1.2 Document Conventions, 2.3 User Classes and Characteristics and 2.4 Operating Environment	1.5
Fong Zheng Feng Victor	5 November 2024	Update Other Requirements	1.6
Tio Sher Min	5 November 2024	Update 3.1 User Interfaces	1.6
Kee Yu Kai	5 November 2024	Update External Interface Requirements	1.6
Reswara Anargya Dzakirullah	8 November 2024	Update External Interface Requirements and 2.5 Design Implementation and Constraints	1.7
Tio Sher Min	9 November 2024	Update 2.6 User Documentation	1.8

1. Introduction

1.1 Purpose

The purpose of this Software Requirement Specification (SRS) is to serve as documentation of the web application, ParkIT!. To facilitate the development process among the development team, relevant stakeholders, and clients, this SRS document describes the requirements specifications for ParkIT! in detail, including but not limited to web application features, functional and non-functional requirements, interface design, and limitations. Diagrams are used to help describe the requirements specifications, such as use case models, class diagrams, sequence diagrams, state machine diagram, system architecture diagram.

1.2 Document Conventions

This Software Requirements Specification (SRS) document follows standard typographical conventions to ensure clarity and consistency.

Font:	Times New Roman
Heading 1:	Size 18, Bold
Heading 2:	Size 14, Bold
Heading 3:	Size 12, Bold
Heading 4:	Size 11, Bold
Content:	Size 11
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 Software Specification Requirement (SRS) document is designed to cater to the needs of various stakeholders, including the development team, project managers, marketing team, testers, documentation writers, and end-users of ParkIT!.

Although it is recommended that all parties involved read the full document in order to obtain a comprehensive understanding of the project, we identified key components that are especially important to each group:

Software Developers
Should focus on the overall system architecture and detailed requirement specifications to understand the technical implementation.

Project Managers
Can review the entire document, paying special attention to the product scope, objectives, and high-level requirements to ensure alignment with project goals.

Marketing Personnel
Should concentrate on the product scope and user interface sections to align marketing strategies with product capabilities.

Documentation Writers
Can utilise the entire document to create user manuals and help documentation.

Users
Are encouraged to read the overview and user interaction sections to familiarise themselves with the system's functionalities and user interface.

Testers
Should focus on the detailed requirements and exception handling to devise comprehensive test plans.

Documentation Writers
Can utilise the entire document to create user manuals and help documentation.

1.4 Product Scope

In line with the advancements of the digital era and the push for smart urban solutions, ParkIT! is a user-centric application designed to simplify the parking experience for drivers. Built using MongoDB, Node.js, and React, ParkIT! allows users to locate and secure parking spaces conveniently near their destinations. This app empowers users by reducing time spent searching for parking, enhancing trip planning, and improving cost transparency. Through ParkIT!, users can search for available parking in a given area, filter options based on individual preferences, select a carpark, and receive seamless navigation support through Google Maps integration. Users can also check in upon arrival and check out when leaving, with the application calculating parking fees based on duration.

1. **Destination-Based Parking Search**
   ParkIT! enables users to search for nearby parking spaces by simply entering their destination. The app provides an intuitive list of available parking options in the vicinity, helping users save time and avoid the hassle of manual searches.

2. **Parking Filter and Selection**
   Users can filter parking options based on their priorities, such as proximity or cost. This allows users to choose car parks that best align with their needs, providing a more tailored experience and supporting a stress-free journey.

3. **Routing to Destination**
   If needed, ParkIT! redirects users to Google Maps, enabling users to navigate effortlessly to their selected carpark. This integration helps streamline travel, ensuring users have access to the fastest routes to their chosen parking spot.

4. **Check-In and Check-Out for Fare Calculation**
   The app features a straightforward check-in and check-out system. When users arrive, they can check in, and upon leaving, they can check out, prompting the app to calculate parking fees based on the duration.

1.5 References

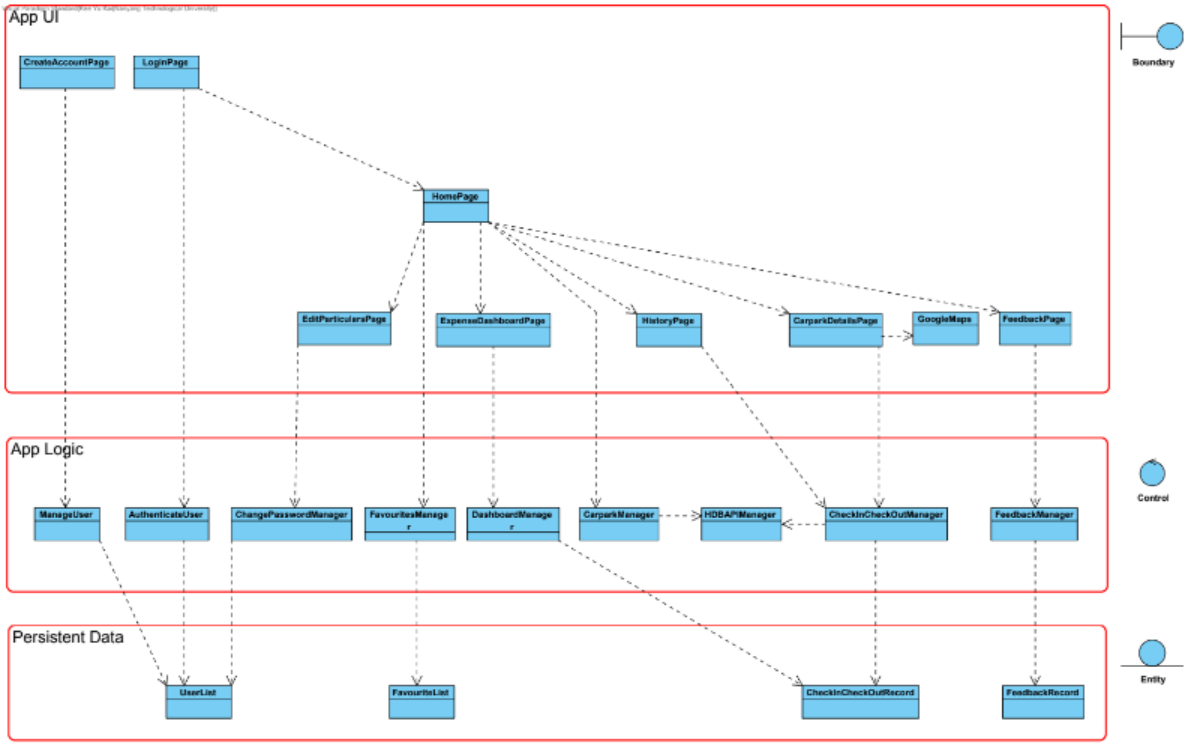
- Google Maps API: https://developers.google.com/maps/documentation/javascript
- Dataset API: guide.data.gov.sg/developer-guide
- MongoDB Documentation: https://www.mongodb.com/docs/
- React JS Documentation: https://reactjs.org/docs/getting-started.html
- Node JS Documentation: https://nodejs.org/en/docs

2. Overall Description

2.1 Product Perspective

The ParkIT! web application was developed to assist users in finding parking lots close to a particular location. To help them choose a preferred parking lot, our app also displays the prices and quantity of lots available at these parking lots. It also aids in keeping track of costs derived from the time spent in each parking lot.

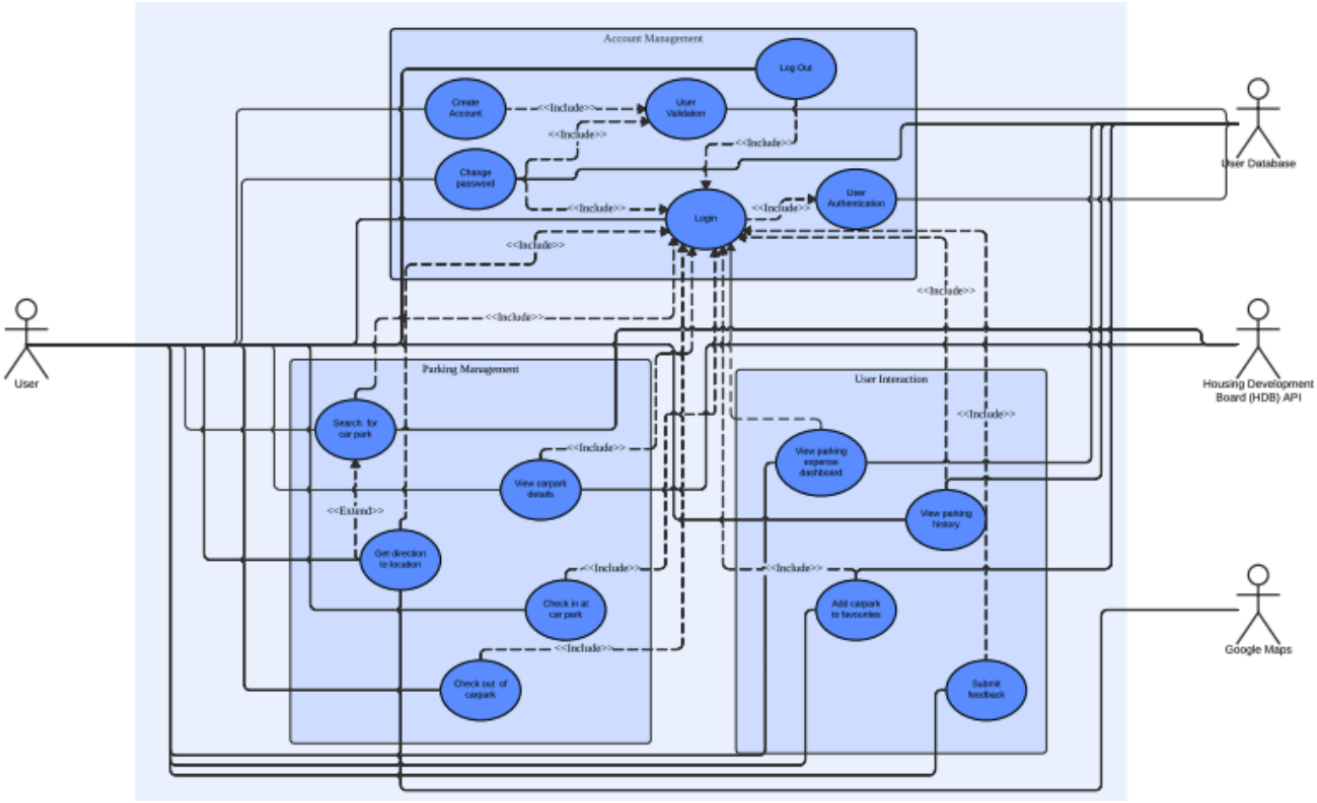
A simplified system architecture diagram is shown below for a brief overview of the operation of ParkIT!.



(High-definition image can be accessed via link in Appendix B: Analysis Models)

ParkIT! makes use of Application Programming Interfaces (API) such as Google Maps API for car park details as well as navigation.

2.2 Product Functions



(High-definition image can be accessed via link in Appendix B: Analysis Models)

ParkIT! must minimally be able to:

- Register users with a new account
- Allow users to login to their account with their password
- Allow users to search for a location and display the details of car parks within a 3km radius around it
- Allow users to reset their password for their account
- Allow users to logout of their accounts

2.3 User Classes and Characteristics

The user classes and their characteristics can be defined as follows:

Age of all User Class is assumed to be 18 years and older and hold a valid licence for driving.

2.3.1 Casual Users

Aspect	**Description**
Frequency of Use	Occasional
Subset of Product Functions Used	All
Technical Expertise	- Owns a personal device that is able to connect to the internet - Have navigated through a web application
Characteristics	Individuals that want to find a suitable parking lot around the destination and to track their parking expenses

2.3.2 Cost Conscious Users

Aspect	**Description**
Frequency of Use	Whenever the user drives
Subset of Product Functions Used	All
Technical Expertise	- Owns a personal device that is able to connect to the internet - Have navigated through a web application
Characteristics	Want to reduce the amount they spend on parking

2.4 Operating Environment

2.4.1 Front-end Development: React.js (Version 18.2.0)

React.js is an open-source JavaScript library designed for building user interfaces, particularly single-page applications where data changes frequently over time. It allows developers to create large web applications that can update and render efficiently by managing a virtual DOM, which optimises updates to the actual DOM. React follows a component-based architecture, enabling reusable and modular code where each component manages its own state and can interact with other components, making it ideal for building complex, dynamic web interfaces.

2.4.2 Back-end Development: Node.js (Version 23.1.0)

Node.js is an open-source, cross-platform JavaScript runtime designed for server-side scripting and allowing developers to use JavaScript for backend development. Known for its asynchronous, non-blocking I/O model, Node.js is highly efficient and scalable, making it ideal for real-time applications, such as chat applications and APIs, that need to handle many concurrent connections.

2.4.3 Database: MongoDB (Version 8.0.1)

MongoDB is a NoSQL, document-oriented database that stores data in flexible, JSON-like documents. This schema-less structure allows for rapid development and scaling, as well as easy handling of complex, hierarchical data structures. MongoDB's design fits well with Node.js applications, providing a natural pairing for full-stack JavaScript development, where data can flow seamlessly between the front end and back end.

2.4.4 Map UI: Google Maps API

The Google Maps API is a versatile set of tools provided by Google, enabling developers to integrate interactive maps directly into their web applications. By using this API, developers can embed customizable map views, complete with zoom controls, map types (like satellite and terrain), and real-time data overlays. It also supports advanced features such as markers, info windows, and custom styling, allowing for a tailored user experience. Ideal for location-based applications, the Google Maps API allows users to interact with geographic data visually, making it easy to create dynamic, responsive map UIs that enhance navigation, location display, and real-world context for front-end applications.

2.4.5 Dataset

Car Park Information: https://beta.data.gov.sg/datasets/d_23f946fa557947f93a8043bbef41dd09/view

Car Park Rates: https://beta.data.gov.sg/datasets/d_9f6056bdb6b1dfba57f063593e4f34ae/view

Car Park Availability: https://beta.data.gov.sg/datasets/d_ca933a644e55d34fe21f28b8052fac63/view

2.5 Design and Implementation Constraints

ParkIT! is currently only available in English and thus might not be usable for users who are unfamiliar with English. Similarly, it is not visually impaired friendly as well. Secondly, ParkIT! depends on the Data.gov API to provide real-time information on the car parks. Thus, any availability or accuracy issues with the API will result in the same issues on ParkIT!. Thirdly, ParkIT! uses free planning for cloud database and hosting services, therefore the response of the application would need more time. This is because the application cannot store all of the data due to the limitation of the database, and must fetch that data continuously from outside of the database whenever needed. Lastly, ParkIT! uses Google Maps API. Although being very reliable, it may not accurately represent the entire Singapore, and any changes to the landscape of Singapore may take a while before Google Maps displays these changes.

2.6 User Documentation

ParkIT!'s source code has been made available at https://github.com/reswaraa/ParkIT for public access. A README.md file has been provided to guide developers on cloning the app if they wish to further develop it.

A demonstration of our web application can be viewed at the following link:
https://drive.google.com/file/d/1KGpLJQa8n2LZJ0T8-SGTgpL991XwY1fe/view?usp=drive_link

2.7 Assumptions and Dependencies

Assumptions:

- Continuous availability of third-party services for car park data.
- Steady internet connectivity for end-users to access the web-based application.
- Assumed that users will provide accurate and valid input while using the application.

Dependencies:

- Reliance on third-party APIs for map UI and routes to destination.
- The application's performance is contingent on the chosen cloud hosting service's reliability and uptime.

3. External Interface Requirements

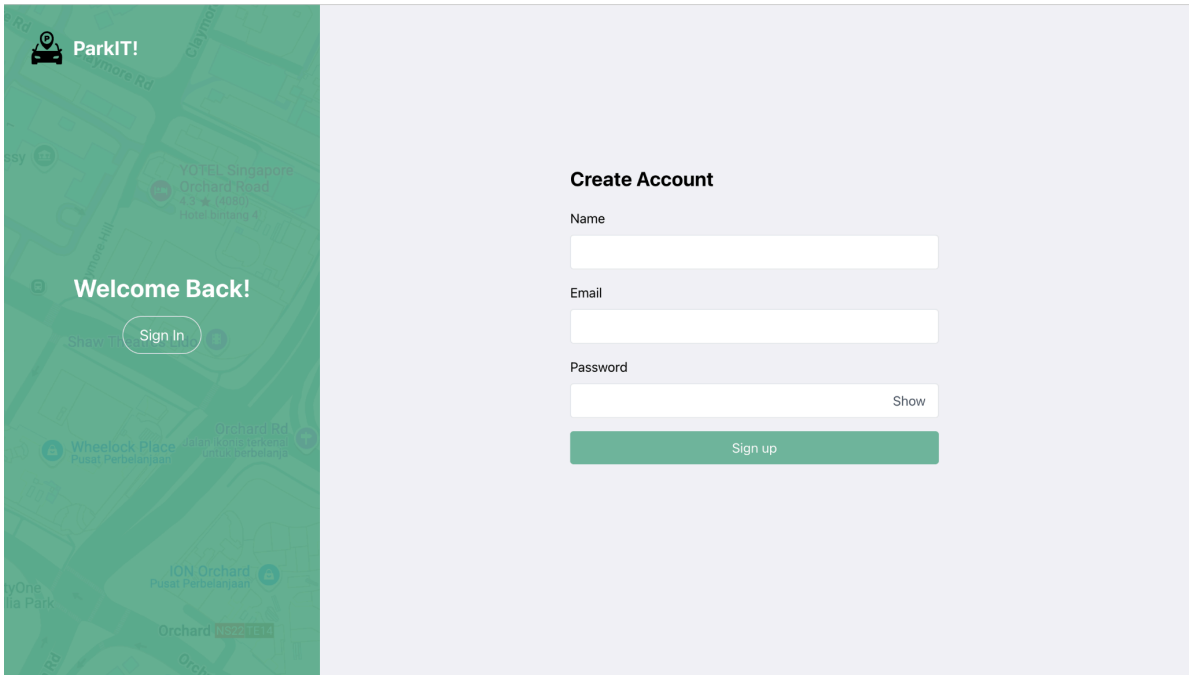
3.1 User Interfaces

Our project provides a variety of functions for users. First-time users are able to create new accounts with username, email and password. Such registered credentials are used to login. After the first successful account creation and login, users will be provided with a visual and text-based tutorial to gain an overall view of our app.

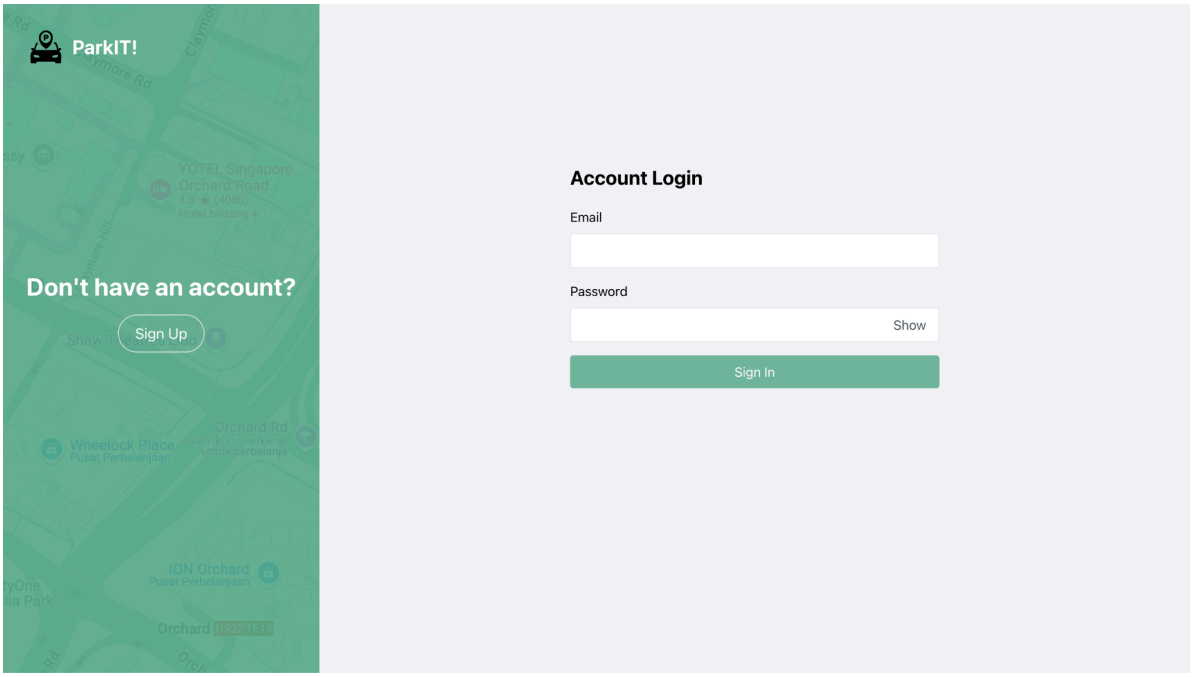
Next, users can input destinations and find a suitable nearby car park by using the filter. The filter options include car park availability, parking rates and distance from the destination. When safely parked, the user can check in and check out when leaving so that car park expenses and history could be seen on "Expense Dashboard" and "History" Dashboard respectively.

Users can also add car parks to a list of favourites by clicking on the "Add to Favourites" button. If users wish to change their password, they may do so at "Edit Particulars". Last but not least, our project also enables users to submit feedback with text, images and videos.

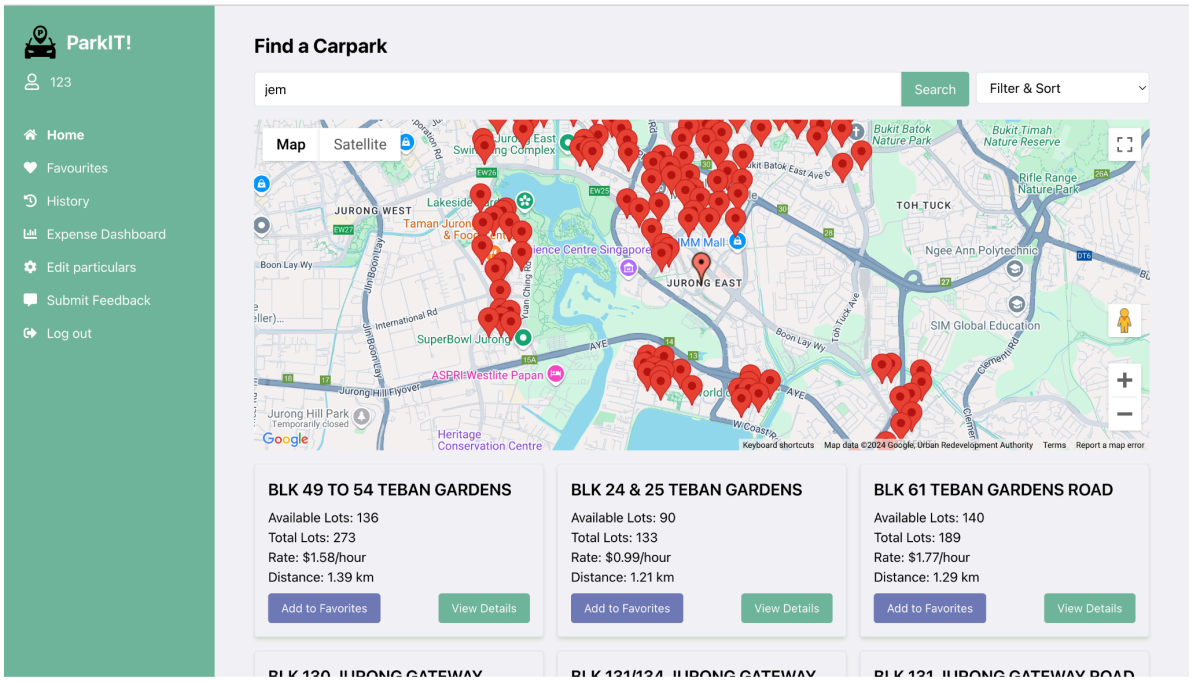
3.1.1 Create Account Page



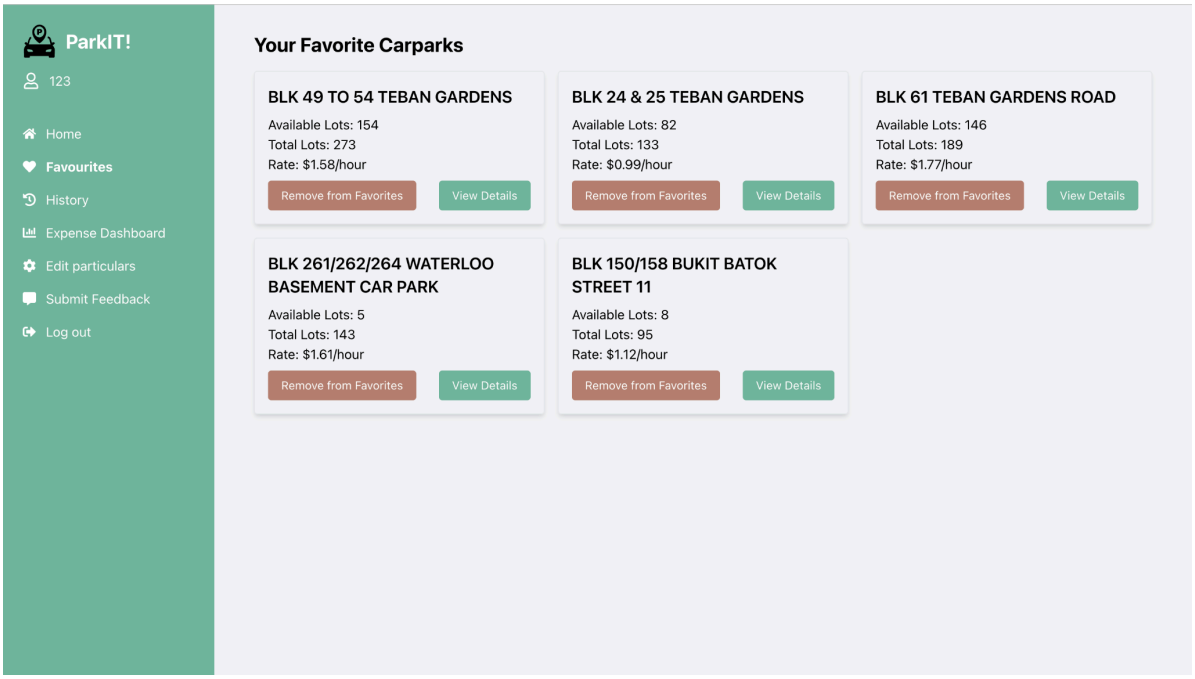
3.1.2 Login Page



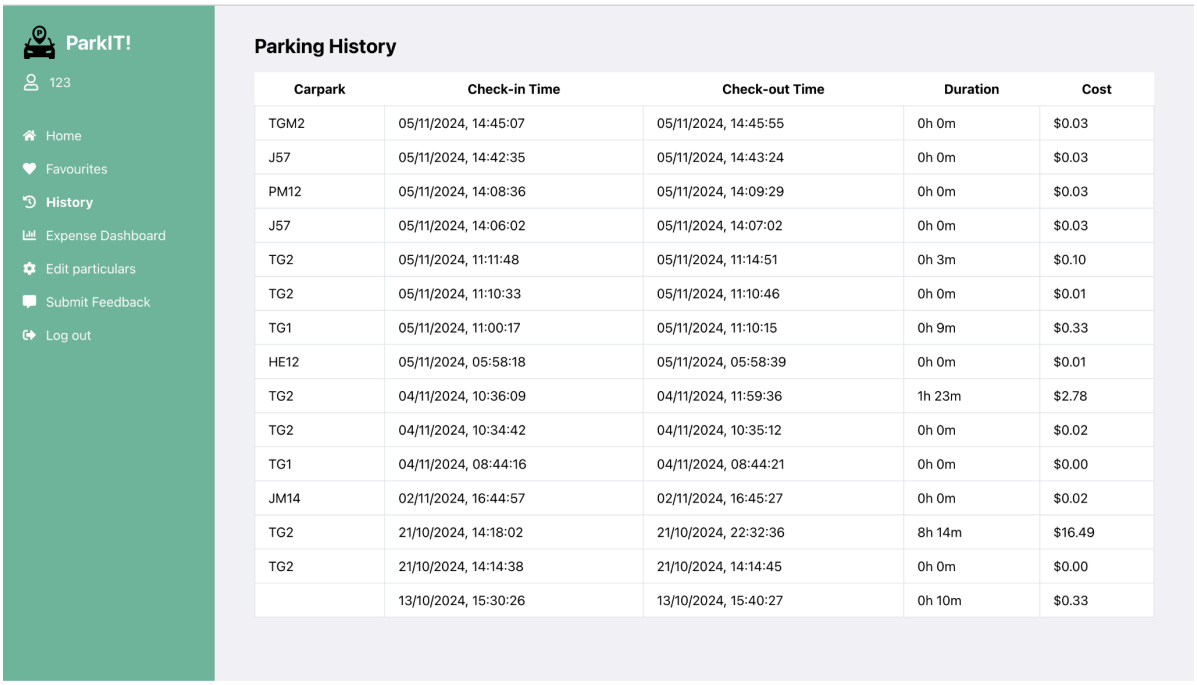
3.1.3 Home Page



3.1.4 Favourites Page

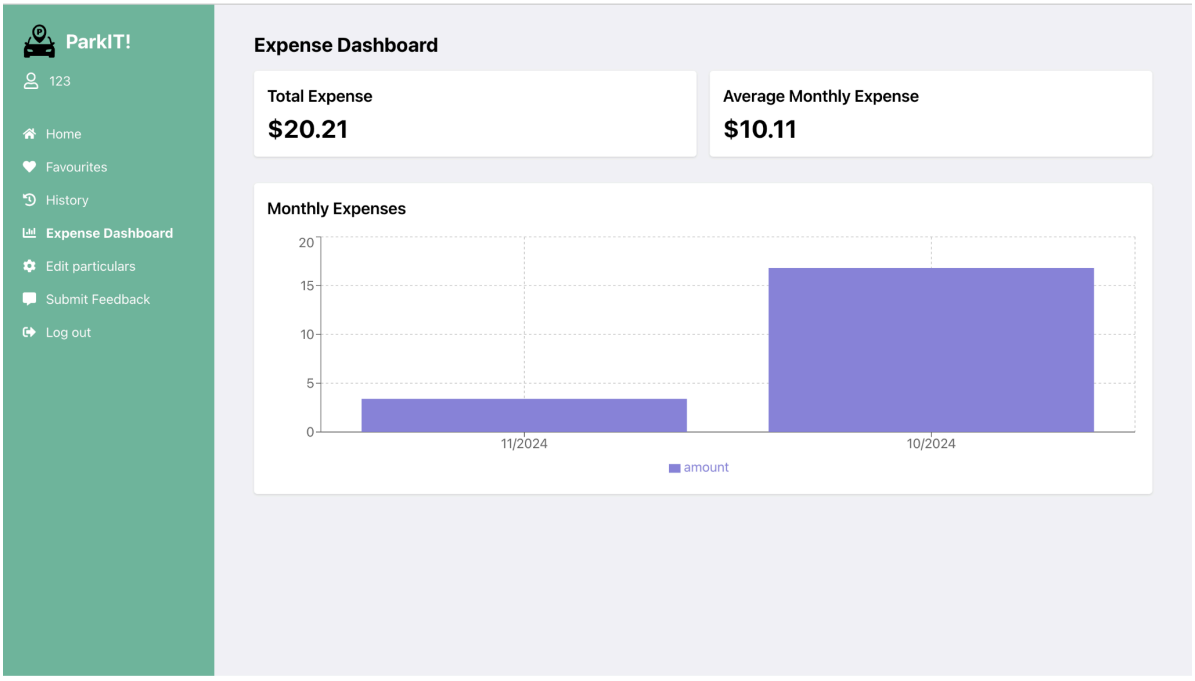


3.1.5 History Page

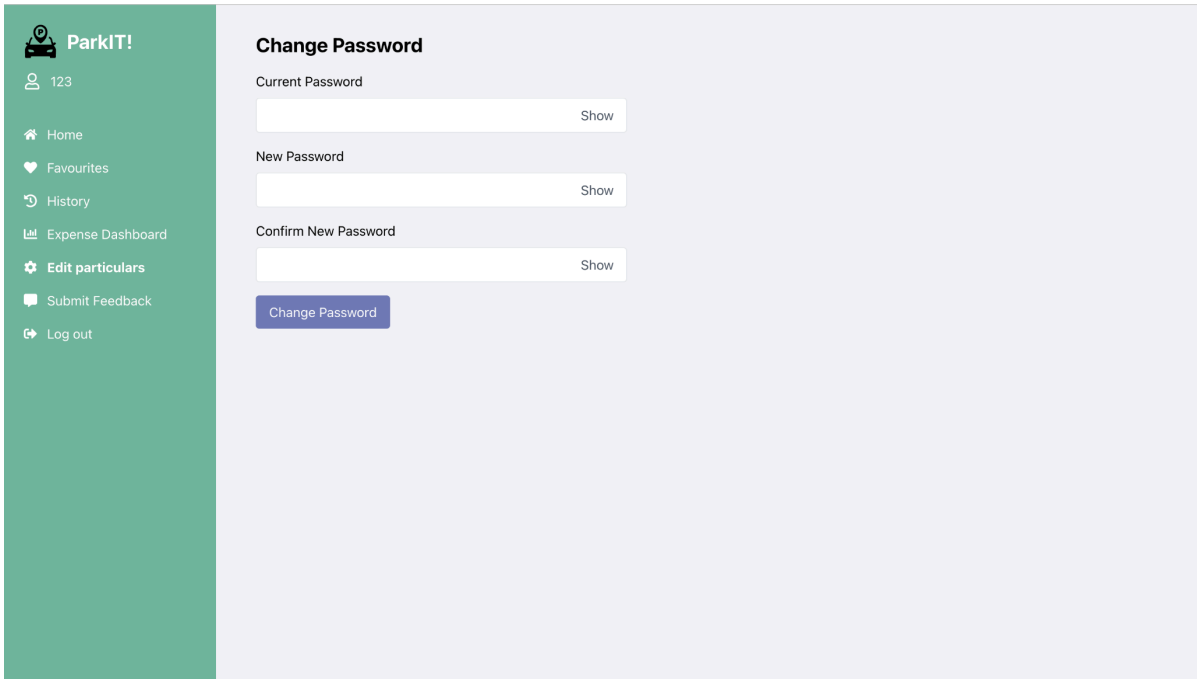


Carpark	Check-in Time	Check-out Time	Duration	Cost
TGM2	05/11/2024, 14:45:07	05/11/2024, 14:45:55	0h 0m	$0.03
J57	05/11/2024, 14:42:35	05/11/2024, 14:43:24	0h 0m	$0.03
PM12	05/11/2024, 14:08:36	05/11/2024, 14:09:29	0h 0m	$0.03
J57	05/11/2024, 14:06:02	05/11/2024, 14:07:02	0h 0m	$0.03
TG2	05/11/2024, 11:11:48	05/11/2024, 11:14:51	0h 3m	$0.10
TG2	05/11/2024, 11:10:33	05/11/2024, 11:10:46	0h 0m	$0.01
TG1	05/11/2024, 11:00:17	05/11/2024, 11:10:15	0h 9m	$0.33
HE12	05/11/2024, 05:58:18	05/11/2024, 05:58:39	0h 0m	$0.01
TG2	04/11/2024, 10:36:09	04/11/2024, 11:59:36	1h 23m	$2.78
TG2	04/11/2024, 10:34:42	04/11/2024, 10:35:12	0h 0m	$0.02
TG1	04/11/2024, 08:44:16	04/11/2024, 08:44:21	0h 0m	$0.00
JM14	02/11/2024, 16:44:57	02/11/2024, 16:45:27	0h 0m	$0.02
TG2	21/10/2024, 14:18:02	21/10/2024, 22:32:36	8h 14m	$16.49
TG2	21/10/2024, 14:14:38	21/10/2024, 14:14:45	0h 0m	$0.00
	13/10/2024, 15:30:26	13/10/2024, 15:40:27	0h 10m	$0.33

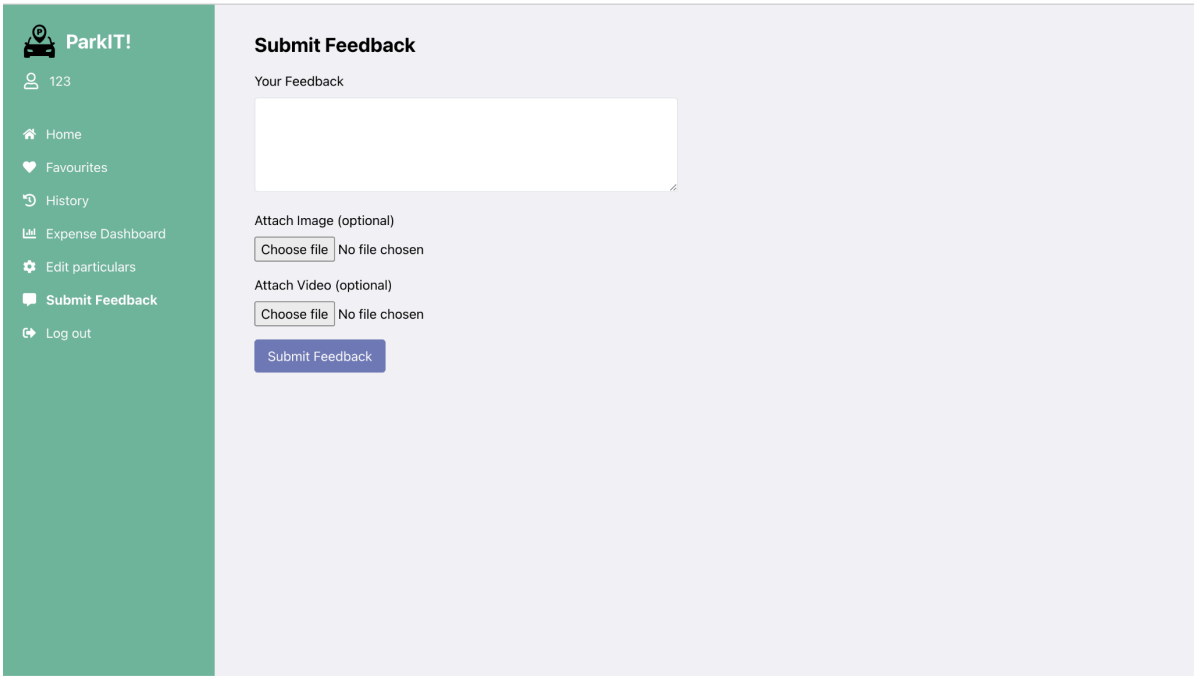
3.1.6 Expense Dashboard Page



3.1.7 Edit Particulars Page



3.1.8 Submit Feedback Page



3.2 Hardware Interfaces

This section describes the hardware interfaces required for the ParkIT! website to perform effectively and reliably.

Requirements	**Description**
Operating System	Devices with Windows 11 or Android 14 or iOS 17 or MacOS 14 Sonoma onwards.
Network Connection	Wireless Network Interface Card (WNIC) or a modem chip with a cellular modem.
Interaction	A functional touchpad or mouse to navigate through the website.

3.3 Software Interfaces

This section shows the software interfaces required for the ParkIT! website to run as intended.

Software Components	**Provider**	**Description**
Operating System	Windows 11 or MacOS 14 Sonoma	Latest Operating System
Tools	React js	A popular JavaScript library for building fast, interactive user interfaces, by enabling efficient component-based architecture and state management.
	Node js	A runtime environment that allows developers to run JavaScript server-side, enabling the creation of scalable, fast, and lightweight backend applications.
	Express	A lightweight, flexible Node.js web application framework that simplifies building server-side applications and APIs with robust routing, middleware support, and HTTP utility methods.
	GithubAction	An automation tool that allows the creation of custom workflows for continuous integration and continuous deployment (CI/CD) directly in the GitHub repository.
Database	MongoDB	A NoSQL database known for its flexibility and scalability, storing data in JSON-like documents, making it ideal for handling large volumes of

		unstructured or semi-structured data.
Libraries	Tailwind CSS	A utility-first CSS framework that provides low-level, reusable styles to quickly build custom designs directly in your HTML without writing traditional CSS.
	React-toastify	A lightweight JavaScript library for React that simplifies displaying customizable and responsive toast notifications with ease.
	Recharts	A composable charting library for React that makes it easy to create responsive, customisable and visually appealing data visualisations.
Cloud Services	Vercel (frontend)	A cloud platform for deploying, hosting, and optimising web applications, offering fast, serverless deployment with automatic scaling and global CDN.
	Render (backend)	A cloud platform that provides developers with scalable and easy-to-use infrastructure to deploy and manage web applications, APIs, and static sites with automatic scaling and continuous deployment.
APIs	Google Maps API	Allows the app to embed customizable maps, geolocation, and spatial data features, enhancing user experiences with interactive maps and location-based services.
	Data Gov API	Provides access to a vast collection of Singapore's government datasets including car park information, availability, and parking rates, enabling to query, retrieve, and integrate public data into applications.

3.4 Communications Interfaces

A stable internet connection is required for data to be fetched from API and user database since communications via HTTP GET, PATCH and DELETE are required. Otherwise, data from APIs and user databases will not be fetched causing the website to not be fully functional.

This section describes the communication requirements for ParkIT!, focusing on how the application interacts with various services and components through network communications.

RESTful API Calls: Communication between frontend and backend API

1. Authentication
    - POST /api/auth/register - Register new user

- POST /api/auth/login - User login

2. User management
   - PUT /users/:id/change-password - Change user password
   - POST /users/:id/complete-tutorial - Tutorial completion

3. Car Park Services
   - GET /api/carparks - Get all car parks
   - GET /api/carparks/:id - Get specific car park

4. Check-in check-out carpark
   - POST /api/checkin - Check-in the user to the carpark
   - POST /api/checkout - Check-out the user to the carpark
   - GET /checkins/:userId - Get check-in history of the user
   - GET /checkin/:userId/:carparkId - Get check-in status of the user

5. Favourites
   - POST /favourites/add - Add to favourites
   - POST /favourites/remove - Remove to favourites
   - GET /favourites/:userId - Get user's favourites

6. Expenses
   - GET /api/expenses/:userId - Get user parking expenses

7. Feedback
   - POST /feedbacks - User feedbacks

Communication Standards and Protocols

1. HTTP/HTTPS: All communications are conducted over HTTP/HTTPS, ensuring reliable and standardised data transfer.

2. Message Formatting: Data exchanged between the frontend, backend, and external services is formatted in JSON, providing a lightweight data interchange format.

4. Functional Requirements

4.1 Create New Account for First-Time Users

4.1.1	The web application must require the user’s credentials for account creation.
4.1.1.1	The web application must require the user's name.
4.1.1.2	The web application must require the user’s email.

4.1.1.3	The web application must require the user's password.
4.1.2	The web application must validate the uniqueness of the email address to ensure that each email is associated with only one account.
4.1.3	The web application must ensure that the password entered meets security criteria.
4.1.3.1	The password must have a minimum of 8 characters.
4.1.3.2	The password must have at least 1 lowercase letter.
4.1.3.3	The password must have at least 1 uppercase letter.
4.1.3.4	The password must have at least 1 special character.
4.1.4	Once an account is created, the web application must log the user in.

4.2 Login

4.2.1	The web application must allow existing users to log in using their registered credentials.
4.2.1.1	The web application must require the user's registered email.
4.2.1.2	The web application must require the user's registered password.
4.2.2	The web application must validate the login credentials and grant access provided email and password are correct.
4.2.2.1	The web application must display an error message when the login credentials are incorrect.
4.2.3	The web application must allow the user to log out of their account.
4.2.3.1	The web application must redirect them to the login page upon logout.

4.3 Present Tutorial to New User upon First Successful Account Creation and Login

4.3.1	The web application must provide an option to skip the tutorial for the user who does not wish to view it.
4.3.2	The tutorial must provide visual and text-based instructions to help new users navigate the application.
4.3.3	The web application must track whether the user has completed the tutorial.

4.3.3.1	The web application must not prompt the user to view the tutorial again upon subsequent logins after the completion of the tutorial.
4.3.3.2	The user must be directed to the home page after skipping or finishing the tutorial.

4.4 Input Destination

4.4.1	The web application must use destination and search for car parks within a set distance (3km by default).
4.4.1.1	The web application must query the details of car parks within the set distance.
4.4.2	The web application must display information about the nearby car parks.
4.4.2.1	The web application must show car parks available as icons on the map, and details must be shown when selected.
4.4.2.1.1	The car park details displayed must consist of car park availability.
4.4.2.1.2	The car park details displayed must consist of parking rates.
4.4.2.1.3	The car park details displayed must consist of the distance from the destination.
4.4.2.1.4	The car park details displayed must consist of service time.
4.4.2.1.5	The car park details displayed must consist of vacancy.
4.4.3	Users can find a suitable car park by using the filter.
4.4.3.1	Users can select one, some, all, or none of the filter options.
4.4.3.1.1	One of the filter options must include car park availability.
4.4.3.1.2	One of the filter options must include parking rates.
4.4.3.1.3	One of the filter options must include distance from the destination.
4.4.3.1.4	One of the filter options must include service time.
4.4.3.1.5	One of the filter options must include vacancy.
4.4.4	The icons of each car park must display a different colour based on its vacancy.
4.4.4.1	A green icon indicates that the car park still has vacancies.
4.4.4.2	A red icon means that the car park is full.
4.4.4.3	Vacancies of car parks are tracked live.

4.4.5	The user can select their preferred choice of car park by clicking on that car park.
4.4.5.1	The web application will redirect the user to Google Maps to find the best route to the car park.

4.5 Check In and Out of A Car Park

4.5.1	When safely parked, the user can check in the web application to create a time log.
4.5.1.1	The web application must add a new parking detail on “history” with the button “checkout” which indicates that this parking event has not been checked out.
4.5.2	If the vacancy of the car park is full, a notification must appear recommending another car park.
4.5.2.1	Clicking on the notification must redirect the user to Google Maps to find the best route from the current car park to the recommended one.
4.5.3	The user can check out of the web application when exiting the car park to end the time log.
4.5.3.1	The web application must calculate the expense based on duration (Time In and Time Out) and car park rate when the user clicks on “checkout”.
4.5.3.2	The web application must save expenses in the “Expense Dashboard”.
4.5.3.3	The web application must update the “checkout” button in the “history” to the time the parking event was checked out.

4.6 View Individual Car Park Expenses on “Expense Dashboard”

4.6.1	The dashboard must display the total current expenditure up to date.
4.6.2	The dashboard must display the average expenditure for some range of time.
4.6.2.1	One option is to display the average expenditure in a day.
4.6.2.2	One option is to display the average expenditure in a week.
4.6.2.3	One option is to display the average expenditure year.

4.7 View Individual Car Park History on “History” Dashboard

4.7.1	The “history” dashboard must keep records of the parking event details for a year.
4.7.1.1	Each event detail on the “history” dashboard must contain the name of the car park
4.7.1.2	Each event detail on the “history” dashboard must contain the address of the car park
4.7.1.3	Each event detail on the “history” dashboard must contain a “check in” time.
4.7.1.4	Each event detail on the “history” dashboard must contain a “check out” time.
4.7.1.4.1	If the parking event has not checked out yet, the event shows a “check out” button for the user to check out.
4.7.1.5	Each event detail on the “history” dashboard must contain expenses.

4.8 Add Car Parks to A List of Favourites

4.8.1	The user must be able to see a list of car parks when they search for the desired car park.
4.8.2	The web application must display an “Add to Favourites” icon for each car park in the list.
4.8.3	The user must be able to select the "Add to Favourites" icon for a car park.
4.8.4	The web application must save the car park to the user's favourites list if the user selects the "Add to Favourites" icon for a car park.

4.9 Submit Feedback

4.9.1	The user must be able to click on the “Feedback” icon to submit feedback.
4.9.2	The web application must direct the user to a page to submit feedback.
4.9.3	The user must be able to go back to the previous page even if they do not submit feedback.
4.9.3.1	The web application must display a “go back” icon on the feedback submission page
4.9.3.2	If the user clicks on the “go back” icon, the web application must redirect the user back

	to the previous page.
4.9.4	The web application must display a feedback form on the feedback page for the user.
4.9.4.1	The user must be able to type in characters in the form.
4.9.4.2	The user must be able to attach images in the form.
4.9.4.3	The user must be able to attach video files in the form.
4.9.5	The user must be able to click on “Submit” to submit the feedback.
4.9.6	The user must be able to go back to the web application after submitting feedback.
4.9.6.1	After submission, the web application must display a “Feedback Successfully Submitted!” message upon successful submission.
4.9.6.2	The web application must display a “go back” icon to redirect the user back to the web application page after submission.

5. Non-Functional Requirements

5.1 Usability

5.1.1	80% of first-time users must be able to view information of the car park and its vacancies within 2 minutes of starting to use the web application.
5.1.2	The user must be able to operate the web application safely with one hand, while on the road.

5.2 Performance

5.2.1	When the user opens the web application or inputs their destination or redirects the user to Google Maps, the system must respond and/or display the correct information within 2 seconds.

5.3 Reliability

5.3.1	The web application must provide accurate and up-to-date information on parking lot vacancies, parking rates, service times, distance and duration of user to car park.

5.3.2	The web application must have an uptime of 99.9%, ensuring that it is available to users at all times.

5.4 Accuracy

5.4.1	The web application must update the information on car park vacancies and duration to car park within 20 seconds of the API updating its information, even when a user is currently viewing an image
5.4.2	The web application must display parking rates accurate to the current hour.

5.5 Security

5.5.1	The web application must encrypt user data into the database.
5.5.2	The web application must follow best practices for secure authentication.

5.6 Supportability

5.6.1	The web application must support multiple languages based on the user's locale.
5.6.2	The web application must be compatible with the latest versions of web browsers.

5.7 Maintainability

5.7.1	The web's codebase must be modular and well-documented to facilitate easy updates and bug fixes.
5.7.2	The web is designed so that testing can be performed easily and efficiently which does not introduce new bugs when any changes are made.

6. Other Requirements

6.1 Installations

6.1.1 To access the web application, visit this link: https://park-it-ashy.vercel.app/login

7. Testing

7.1 Black Box Testing

7.1.1 Create Account

Test ID	Scenario	Name	Email	Password	Expected Result	Actual Result
1	User key in valid name, email, password	testuser1	testuser3@email.com	12345678aA!	System will display a “Account created successfully ” message and redirects to login page	System will display a “Account created successfully ” message and redirects to login page
2	User keys in invalid email format	testuser2	testuser2@	12345678aA!	System will display a “Email is incomplete” message	System will display a “Email is incomplete” message
3	User keys in password of length less than 8	testuser3	testuser3@email.com	1234aA!	System will display a weak password warning	System will display a weak password warning
4	User keys in password with no lowercase letters	testuser4	testuser4@email.com	12345678A!	System will display a weak password warning	System will display a weak password warning .
5	User keys in password with no uppercase letters	testuser5	testuser5@email.com	12345678a!	System will display a weak password warning	System will display a weak password warning .

6	User keys in password with no special characters	testuser6	testuser6@email.com	12345678aA	System will display a weak password warning	System will display a weak password warning
7	User keys in an email that already exists in the system	testuser1	testuser1@email.com	12345678aA!	System will display “user already exist” message	System will display “user already exist” message
8	User leave the name field empty and click on create account button		testuser7@email.com	12345678aA!	System will display “Fill in all fields” message	System display “Fill in all fields” message
9	User leave the email field empty and click on create account button	testUser7		12345678aA!	System will display “Fill in all fields” message	System display “Fill in all fields” message
10	User leave the password field empty and click on create account button	testUser7	testuser7@email.com		System will display “Fill in all fields” message	System display “Fill in all fields” message

7.1.2 Login

Test ID	Scenario	Email	Password	Expected Result	Actual Result
1	User keys in valid email and password	testuser1@email.com	12345678aA!	Redirect to home page	Redirect to home page
2	User key in wrong password	testuser1@email.com	12345678	“Invalid credentials” message	“Invalid credentials” message
3	User key in invalid email	invalid@email.com	12345678aA!	“Invalid credentials” message	“Invalid credentials” message
4	User leave the email field empty and		12345678aA!	“Invalid credentials” message	“Invalid credentials” message

	click on login button				
5	User leave the password field empty and click on login button	testuser1@email.com		"Invalid credentials" message	"Invalid credentials" message

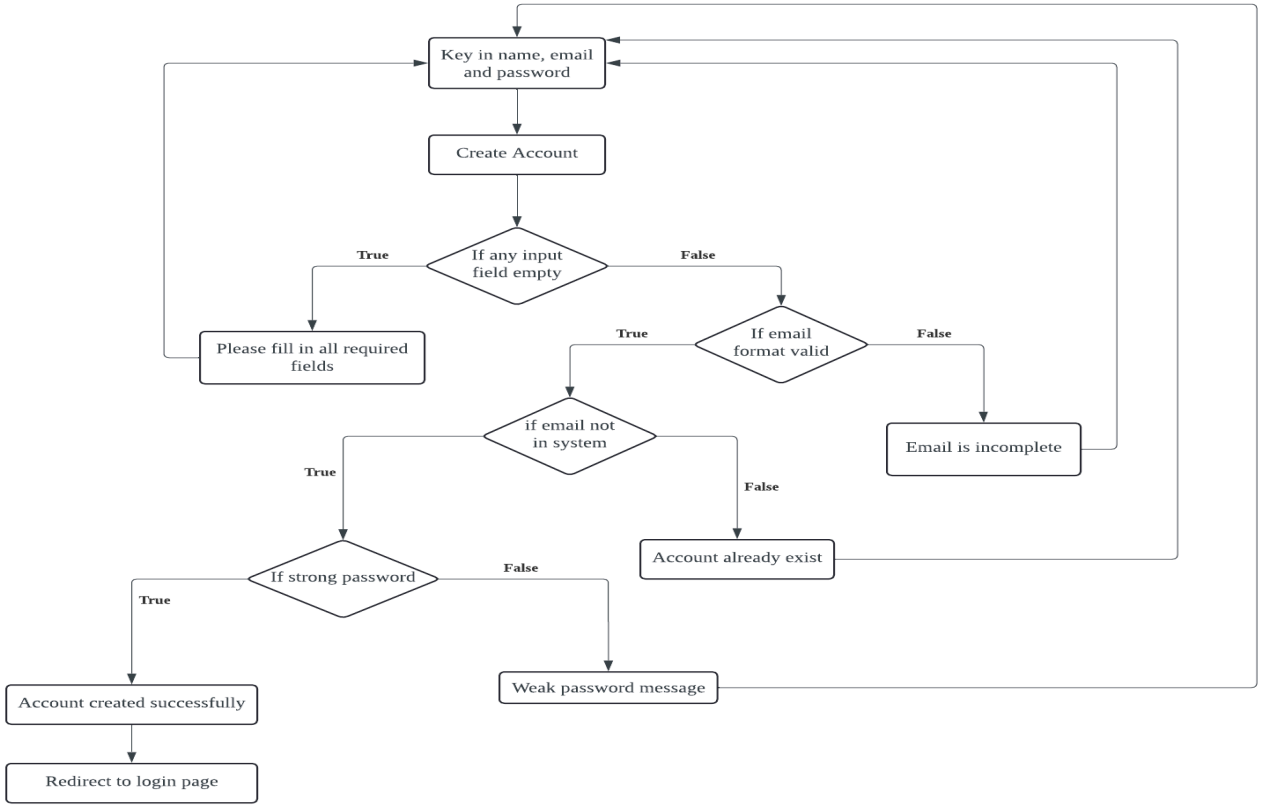
7.1.3 Change Password

Test ID	Scenario	Old Password	New Password	Confirm Password	Expected Result	Actual Result
1	User keys in valid old password, new password and confirmation password	12345678aA!	23456789aA!	23456789aA!	Password changed successfully	Password changed successfully
2	User key in wrong old password	12345678	23456789aA!	23456789aA!	Wrong password message	Wrong password message
3	User key in invalid new password	12345678aA!	23456789aA	23456789aA	"New password is weak" message	"New password is weak" message
4	Confirm password and new password not the same	12345678aA!	23456789aA!	23456789a	New password and confirm password does not match message	New password and confirm password does not match message
5	New password field empty	12345678aA!		23456789aA!	Please fill in all the required fields message	Please fill in all the required fields message
6	Confirm password field empty	12345678aA!	23456789aA!		Please fill in all the required fields message	Please fill in all the required fields message

7.2 White Box Testing

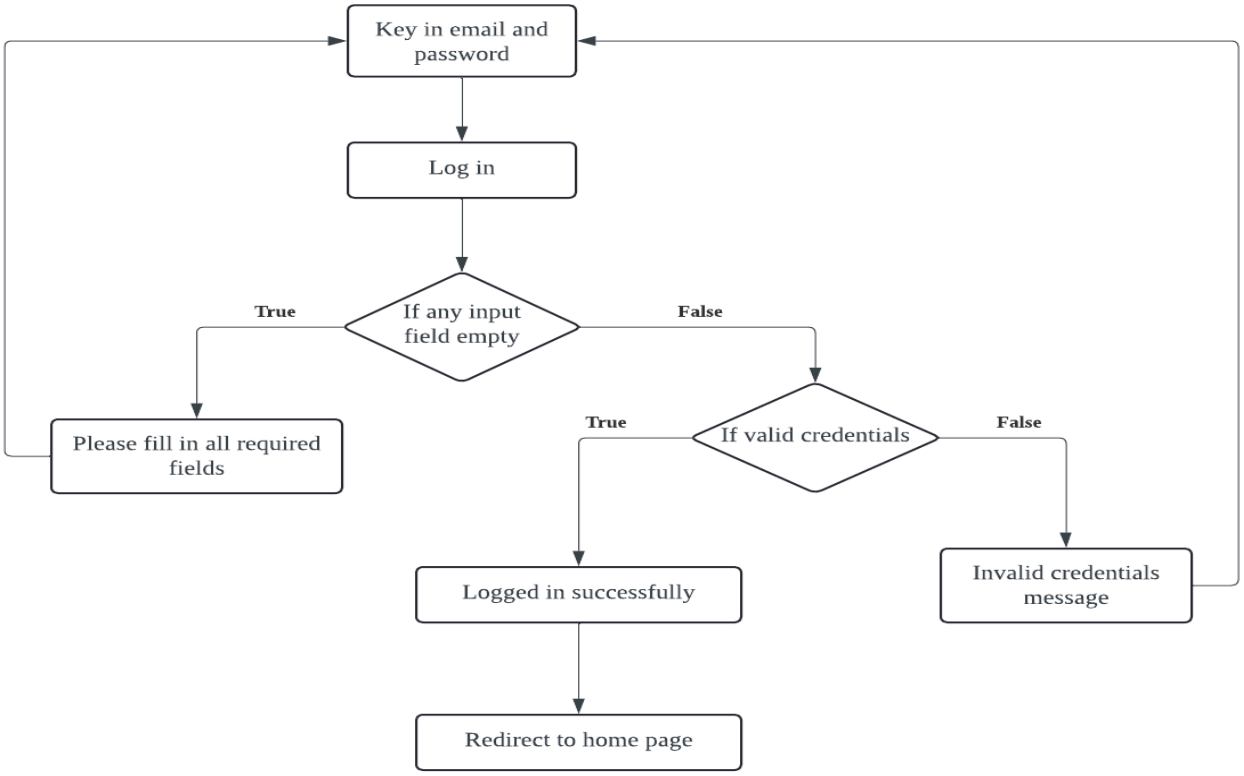
7.2.1 Create Account

High-definition image can be accessed via
https://drive.google.com/file/d/1jY4Ofb4XK7Lf-wCdBQv_rreIgWc8bMJ_/view?usp=drive_link



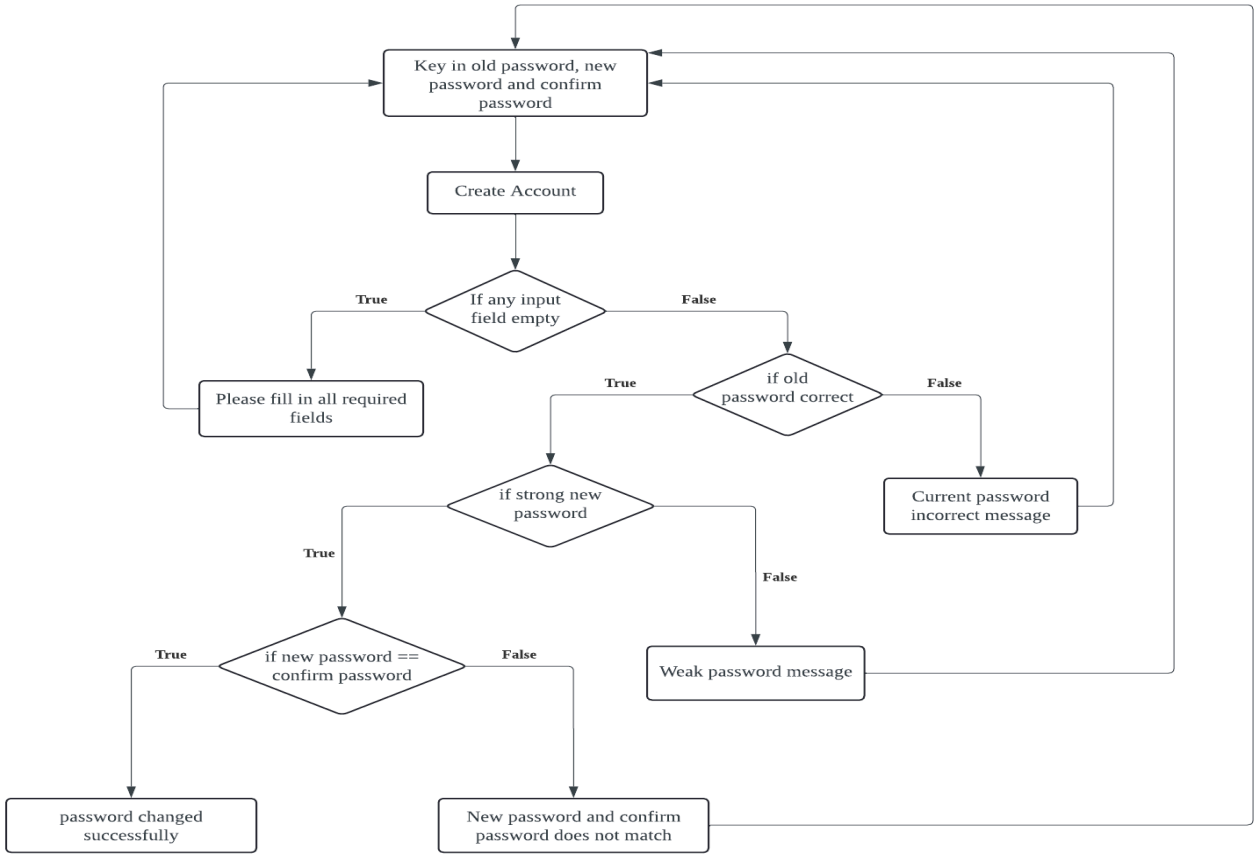
7.2.2 Login

High-definition image can be accessed via
https://drive.google.com/file/d/1RcLOENLvcf8CfjCWn_NOztjK1xT3FEQR/view?usp=drive_link



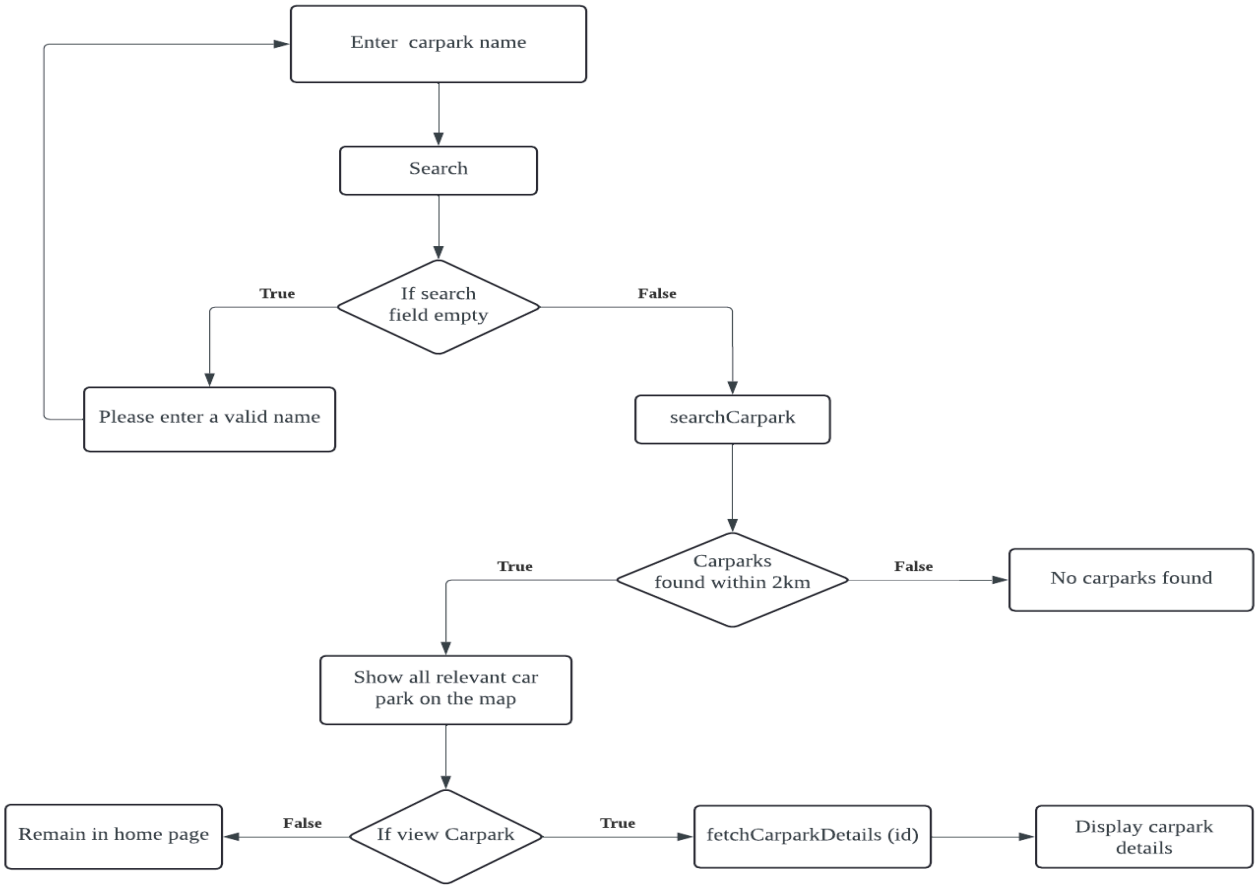
7.2.3 Change Password

High-definition image can be accessed via
https://drive.google.com/file/d/1cJEinmXFEQKtKeE3_9tWI96S7bqS4ymN/view?usp=drive_link



7.2.4 Search Car Park

High-definition image can be accessed via
https://drive.google.com/file/d/1zkDQ5VLCQvqw9L0W1Gqdys_SPF-3r79o/view?usp=drive_link



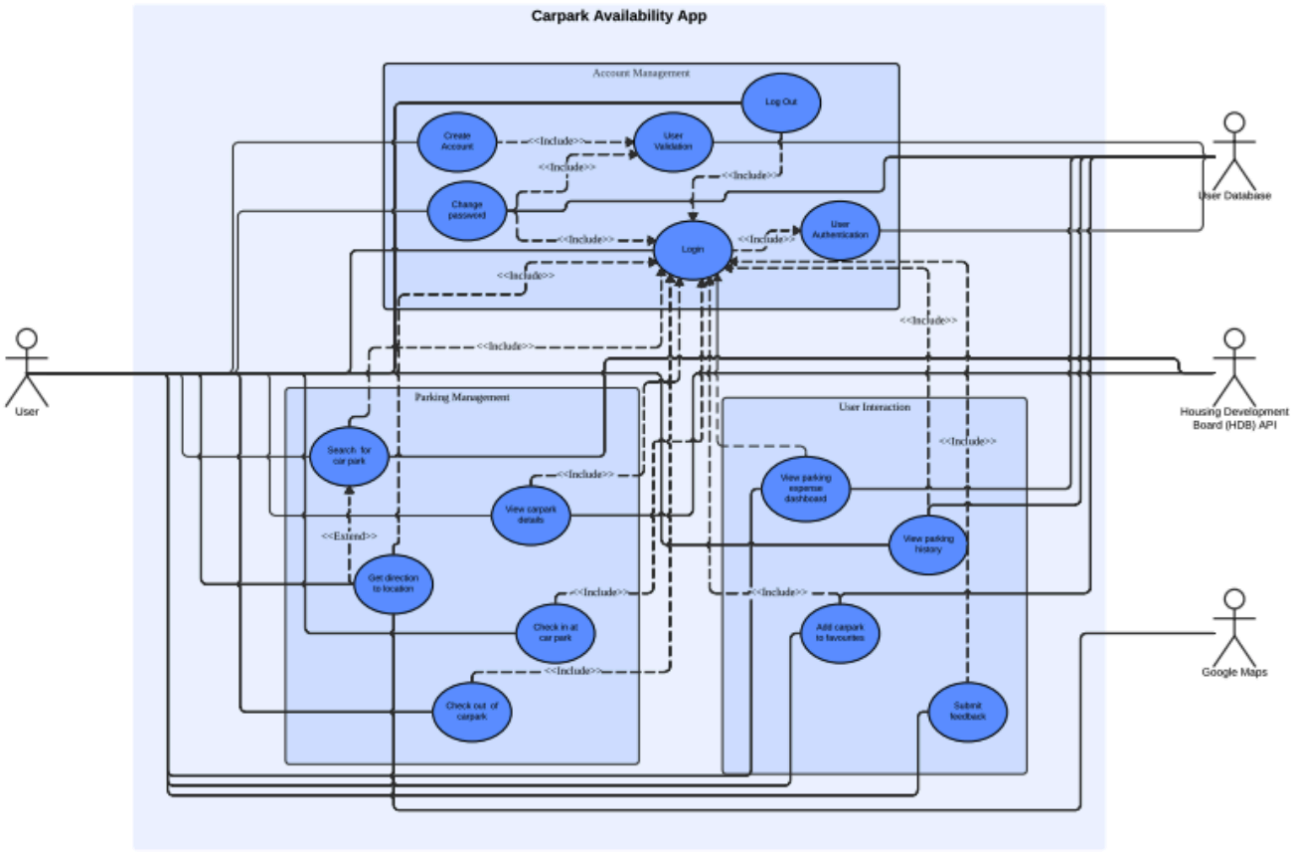
8. Appendix A: Data Dictionary

Term	Definition
User	An individual who owns a vehicle in Singapore and is making use of the web application to find availability of parking slots.
Account	A unique identity created by a user consisting of a name, email, and password. No two accounts can have the same email.
HDB	Housing Development Board. It is a statutory board under the Ministry of National Development responsible for public housing in Singapore. They provide a real-time API for the availability of the parking slots updated every minute.
Car Park	A location that is legally designated for parking.
Car Park Details	Information about a car park, including address, parking lot availability, parking rates, distance from current location, and service time.
Check In	The action a user takes to log their parking start time at a car park.
Check Out	The action a user takes to log their parking end time at a car park, which calculates the parking expense.
Favourites	A list of car parks that a user has marked for easy access and future reference.
Parking Rate	The fee that must be paid upon exit of a parking space. This fee is usually metered based on the duration of use and varies between parking spaces and days of the week.
Parking Expense Dashboard	A visual display showing the user's total expenses up till date, and average monthly spending. Bar charts must be shown for easier visualisation for trends.
Login Credentials	A User ID and password combination that allows users to access a website or application.
Query	A request for data from database.

9. Appendix B: Analysis Models

9.1 Use Case Diagram and Description

High-definition image can be accessed via
https://drive.google.com/file/d/1ZCYYIzUfw8KHSB9e5ebWeHscEbustrmH/view?usp=drive_link



Use Case ID:	1		
Use Case Name:	Create Account		
Created By:	Victor	Last Updated By:	Resa
Date Created:	26/08/2024	Date Last Updated:	15/09/2024

Actor:	User (Initiating actor)
Description:	The user can create an account using their name, email and password. With an account created, the user can log in to the app to access its features
Preconditions:	1. Device has an active internet connection 2. The user has not registered an account before

Postconditions:	1. The user's account is created and stored in the User Database. 2. The user is logged in automatically and redirected to the home page.
Priority:	-
Frequency of Use:	Once for every user
Flow of Events:	1. User selects "Create New Account" 2. User enters their name, email and password. 3. User press the "Create New Account" button. 4. System performs User Validation: - User Validation process checks the uniqueness of the email by querying the User Database. - The system verifies that the password meets security requirements. 5. If validation succeeds, the system creates the account and stores the user's credentials in the User Database. 6. The system logs the user in automatically and redirects them to the home page.
Alternative Flows:	-
Exceptions:	E1: Email is already in use. Application displays an "Email in use" message. The system prompts the user to enter a different email. E2: Password does not meet security requirements. Application displays a "Password does not meet security requirements" message.
Includes:	-
Special Requirements:	The system must encrypt the password before storing it.
Assumptions:	User has a valid email account
Notes and Issues:	-

Use Case ID:	2		
Use Case Name:	User Validation		
Created By:	Resa	Last Updated By:	Resa
Date Created:	15/09/2024	Date Last Updated:	15/09/2024

Actor:	User (Initiating actor), User database
Description:	The system validates user input during account creation or password updates to ensure data accuracy, uniqueness, and conformity to security rules.
Preconditions:	1. The system is connected to the User Database. 2. The user is in the process of creating an account or updating their password.

Postconditions:	1. The user's email is validated and checked for uniqueness in the database. 2. The user's password is validated based on security rules (length, complexity).
Priority:	-
Frequency of Use:	Multiple times for each user during account creation, login, or password changes.
Flow of Events:	1. User enters their email and password. 2. System queries the User Database to verify if the email is unique. 3. System checks the password against security rules (minimum length, special characters, etc.). 4. If the email is unique and the password is valid, the system proceeds with the account creation or password change. 5. If any validation fails, the system displays an appropriate error message (e.g., "Email already in use," "Password does not meet security requirements").
Alternative Flows:	A1: User enters a weak password. The system displays a message: "Password does not meet security requirements. Please enter a valid password."
Exceptions:	E1: Email already exists in the database. The system displays a message: "Email already in use. Please use a different email."
Includes:	-
Special Requirements:	-
Assumptions:	The user provides a valid email address during account creation or password update.
Notes and Issues:	-

Use Case ID:	3		
Use Case Name:	Login		
Created By:	Victor	Last Updated By:	Resa
Date Created:	26/08/2024	Date Last Updated:	15/09/2024

Actor:	User (Initiating actor)
Description:	The user can log in using their email and password. The system authenticates the user by verifying the credentials against the User Database.
Preconditions:	1. Device has an active internet connection 2. The user has already created an account with the application 3. The system is connected to the User Database.

Postconditions:	The user will be successfully logged in and redirected to the home page where they can access personalised features.
Priority:	-
Frequency of Use:	Once everyday
Flow of Events:	1. User navigates to the login page. 2. User enters their email and password. 3. System performs User Authentication: - The system queries the User Database to verify the user's credentials. - If the credentials match, the system logs the user in and redirects them to the home page.
Alternative Flows:	-
Exceptions:	E1: Email and password do not match Application displays a "Wrong password" message. E2: Account does not exist Application displays an "Invalid account" message
Includes:	-
Special Requirements:	The application must make sure the password is encrypted when stored in the database.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	4		
Use Case Name:	User Authentication		
Created By:	Resa	Last Updated By:	Resa
Date Created:	15/09/2024	Date Last Updated:	15/09/2024

Actor:	User (Initiating actor), User database
Description:	The system authenticates the user during login by verifying their credentials (email and password) against the stored data in the User Database.
Preconditions:	1. The user has already created an account. 2. The system is connected to the User Database.
Postconditions:	1. The user is successfully authenticated and logged in to the system. 2. The system grants access to personalised features.
Priority:	-

Frequency of Use:	Once every time the user logs in.
Flow of Events:	1. User enters their email and password on the login screen. 2. System queries the User Database to find the email and retrieve the stored credentials. 3. System compares the provided password with the encrypted password stored in the database. 4. If the credentials match, the system logs the user in and redirects them to the home page. 5. If the credentials don't match, the system displays an appropriate error message.
Alternative Flows:	A1: User enters the wrong password The system displays a message: "Incorrect password. Please try again."
Exceptions:	E1: Account does not exist. The system displays a message: "No account found with the provided email. Please create an account."
Includes:	-
Special Requirements:	The system must use encryption when comparing passwords.
Assumptions:	The user is entering valid credentials.
Notes and Issues:	Consider multi-factor authentication for additional security.

Use Case ID:	5		
Use Case Name:	Log out		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor)
Description:	The user logs out of the application. The application terminates the user session and redirects them to the login page
Preconditions:	1. The user is logged in
Postconditions:	The user session is terminated, and the user is redirected to the login page.
Priority:	-
Frequency of Use:	Once everyday
Flow of Events:	1. User selects the “Log out” button 2. The application ends the user session 3. The system redirects the user to the login page

Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	The system must securely terminate the session to prevent unauthorised access.
Assumptions:	-
Notes and Issues:	-

Use Case ID:	6		
Use Case Name:	Search car park by location		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor), HDB API, Google Maps
Description:	The user can search for car parks based on location or address.
Preconditions:	1. Device has an active internet connection 2. The user is already logged in
Postconditions:	1. The user receives a list of the nearby car parks in the specified location. 2. The system displays car park icons on the map and provides details when selected.
Priority:	-
Frequency of Use:	2 to 3 times a day
Flow of Events:	1. User enters a location in the search destination bar. 2. The system processes the search query. 3. The system displays a list of nearby car parks in the location within a set distance. 4. The system displays car park icons on the map
Alternative Flows:	-
Exceptions:	E1: No carpark found in the radius of 3km of the location Application displays a "No results found" message.
Includes:	-
Special Requirements:	-

Assumptions:	The user provides a valid location and the system can access the required data sources.
Notes and Issues:	-

Use Case ID:	7		
Use Case Name:	View car park details		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor), HDB API
Description:	The user can view detailed information about a selected car park, including car park address, total lots, available lots, car park rates, and operating hours.
Preconditions:	The user has selected a car park from the search results or map view.
Postconditions:	The user sees detailed information about the selected car park.
Priority:	-
Frequency of Use:	-
Flow of Events:	1. User searches for car park by location 2. The application will return a list of nearby car parks 3. User selects a car park. 4. The application retrieves detailed information. 5. The application displays car park details to the user.
Alternative Flows:	A1: User filters car park details based on preferences (e.g., lowest rates, availability).
Exceptions:	E1: Details are unavailable. The system displays a message indicating that car park details could not be retrieved.
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	8		
Use Case Name:	Get directions to location		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor), Google Maps
Description:	The user can obtain real-time directions to the selected car park after performing a search by location using google maps.
Preconditions:	The user has performed a search for car parks by location and selected a car park from the results.
Postconditions:	The user will be directed to Google maps with the starting location being the current location the user is in and the end destination set to the selected car park.
Priority:	-
Frequency of Use:	2 - times a day
Flow of Events:	1. Users will search for car parks by location. 2. Application returns a list of car parks. 3. User selects a car park and clicks on the get directions icon . 4. Application retrieves the user's current location using Google Maps. 5. Application generates a route to the selected car park. 6. Application displays the route and step-by-step directions.
Alternative Flows:	A1: User do not require Google Maps for navigation and remains in the app
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	9		
Use Case Name:	Check in at car park		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor)
Description:	The user checks in at a car park to start a parking session, creating a time log.
Preconditions:	The user has arrived at a car park and selected the "Check In" option.
Postconditions:	The system logs the check-in time and begins tracking the parking session.
Priority:	
Frequency of Use:	2 to 3 times a day
Flow of Events:	1. User selects a car park and chooses “Check In” upon reaching the car park 2. The system logs the time and starts tracking the parking session.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	10		
Use Case Name:	Check out of car park		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User (Initiating actor)
Description:	The user checks out of a car park to end a parking session, and the system calculates the expense based on the duration and rates.
Preconditions:	The user has checked in at a car park.
Postconditions:	The system logs the check-out time, calculates the expense, and saves the parking event.
Priority:	
Frequency of Use:	2 to 3 times a day
Flow of Events:	1. User selects the “Check Out” option. 2. The system logs the time and ends the parking session. 3. The system calculates the expense and saves the parking event to history.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	11		
Use Case Name:	Add car park to favourites		
Created By:	Victor	Last Updated By:	Resa
Date Created:	26/08/2024	Date Last Updated:	15/09/2024

Actor:	User
Description:	The user can add a specific car park to their list of favourites for quick access in future sessions.
Preconditions:	1. The user is logged in 2. The System is connected to User Database
Postconditions:	The selected car park is added to the user's list of favourites.
Priority:	-
Frequency of Use:	Occasionally
Flow of Events:	1. User searches for the desired carpark 2. Application returns a list of car parks 3. User selects the "Add to Favourites" icon for a car park. 4. The application add and saves the car park to the user's favourites list in the User Database
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	12		
Use Case Name:	View parking expense dashboard		
Created By:	Victor	Last Updated By:	Resa
Date Created:	26/08/2024	Date Last Updated:	15/09/2024

Actor:	User (initiating actor), User Database
Description:	The user can view a dashboard that plots their parking expenses over a one year period. The dashboard will also include the total current expenditure up till date and also display the average expenditure a month.
Preconditions:	1. The user is logged in 2. The System is connected to User Database
Postconditions:	The user views their parking expense summary on the dashboard.
Priority:	-
Frequency of Use:	-
Flow of Events:	1. User navigates to the expense dashboard. 2. The system retrieves the user's parking transaction data from User Database
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	13		
Use Case Name:	View parking history		
Created By:	Resa	Last Updated By:	Resa
Date Created:	02/09/2024	Date Last Updated:	15/09/2024

Actor:	User (initiating actor), User Database
Description:	Users can view a dashboard containing a history of parking events. Each parking event includes details about the parking name, address, check-in time, check-out time, and cost. If there is a “check out” button on a parking event, it means that the parking event has not been checked out.
Preconditions:	1. The user is logged in 2. The system is connected to User Database
Postconditions:	The user views their parking history on the dashboard.
Priority:	-
Frequency of Use:	-
Flow of Events:	1. User navigates to the expense dashboard. 2. The system retrieves the user's parking history data from User Database.
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

Use Case ID:	14		
Use Case Name:	Change password		
Created By:	Victor	Last Updated By:	Resa
Date Created:	26/08/2024	Date Last Updated:	15/09/2024

Actor:	User
Description:	The user can change their account password after logging into the application.
Preconditions:	The user is logged in
Postconditions:	The user's password is successfully updated, and a “Password changed successfully message” will be shown.

Priority:	-
Frequency of Use:	-
Flow of Events:	1. User navigates to the “Edit particulars” tab. 2. User enters the current password, the new password, and confirms the new password. 3. The system performs User Validation to check the password against security rules 4. The system updates the password and displays a success message
Alternative Flows:	-
Exceptions:	E1: Current password is incorrect. The system displays an error message and prompts the user to re-enter the correct password. E2: New password does not meet security requirements. The application displays a warning message and prompts the user to enter a valid password.
Includes:	-
Special Requirements:	Passwords stored should be encrypted.
Assumptions:	-
Notes and Issues:	-

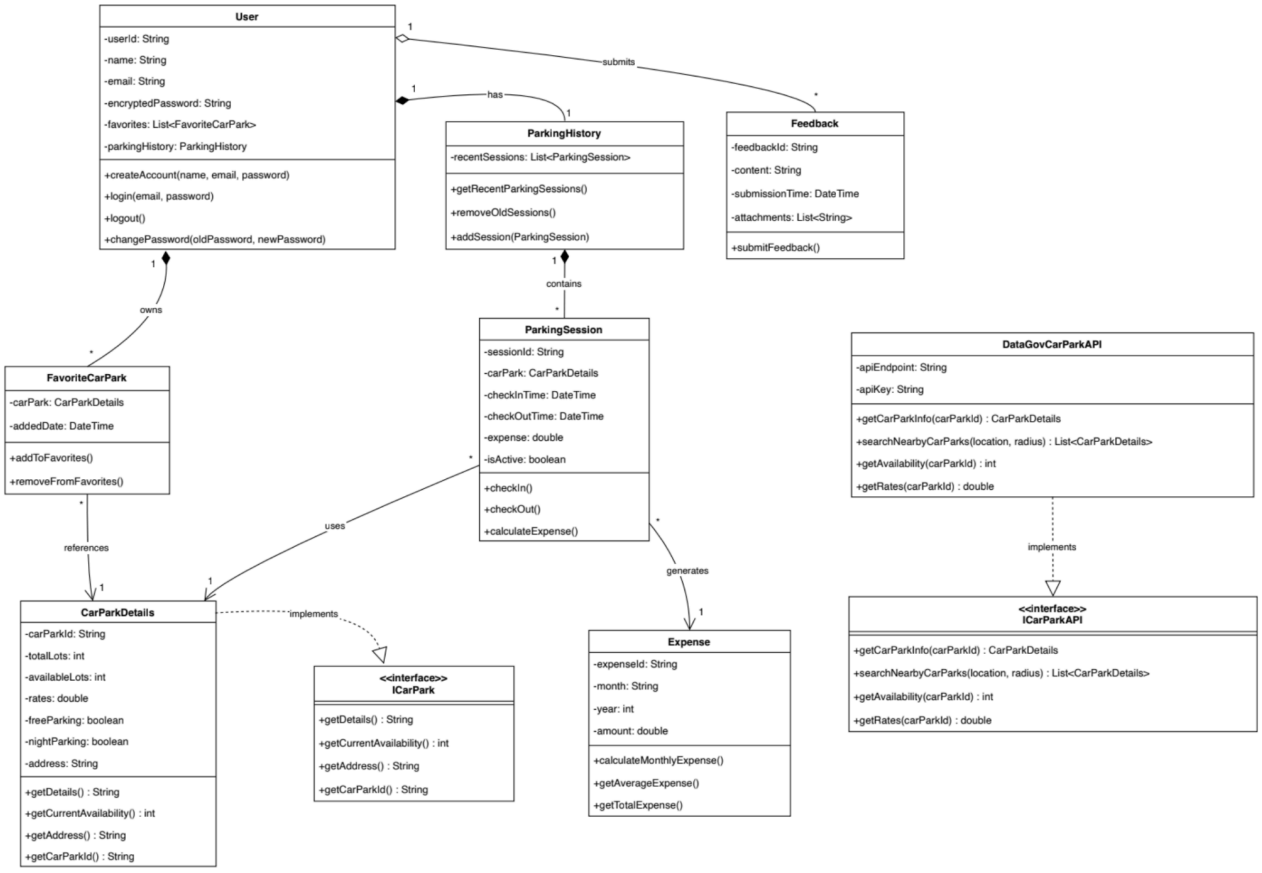
Use Case ID:	15		
Use Case Name:	Submit Feedback		
Created By:	Victor	Last Updated By:	Victor
Date Created:	26/08/2024	Date Last Updated:	26/08/2024

Actor:	User
Description:	The user submits feedback about the application, optionally attaching images or videos.
Preconditions:	The user is logged in
Postconditions:	The system receives and stores the feedback and notifies the user that it has been successfully submitted.
Priority:	-
Frequency of Use:	-

Flow of Events:	1. User selects the "Feedback" option. 2. User enters their feedback and optionally attaches images or videos. 3. User submits the feedback. 4. The system confirms submission and stores the feedback.
Alternative Flows:	A1: User cancels feedback submission and returns to the previous page.
Exceptions:	-
Includes:	-
Special Requirements:	Passwords stored should be encrypted.
Assumptions:	-
Notes and Issues:	-

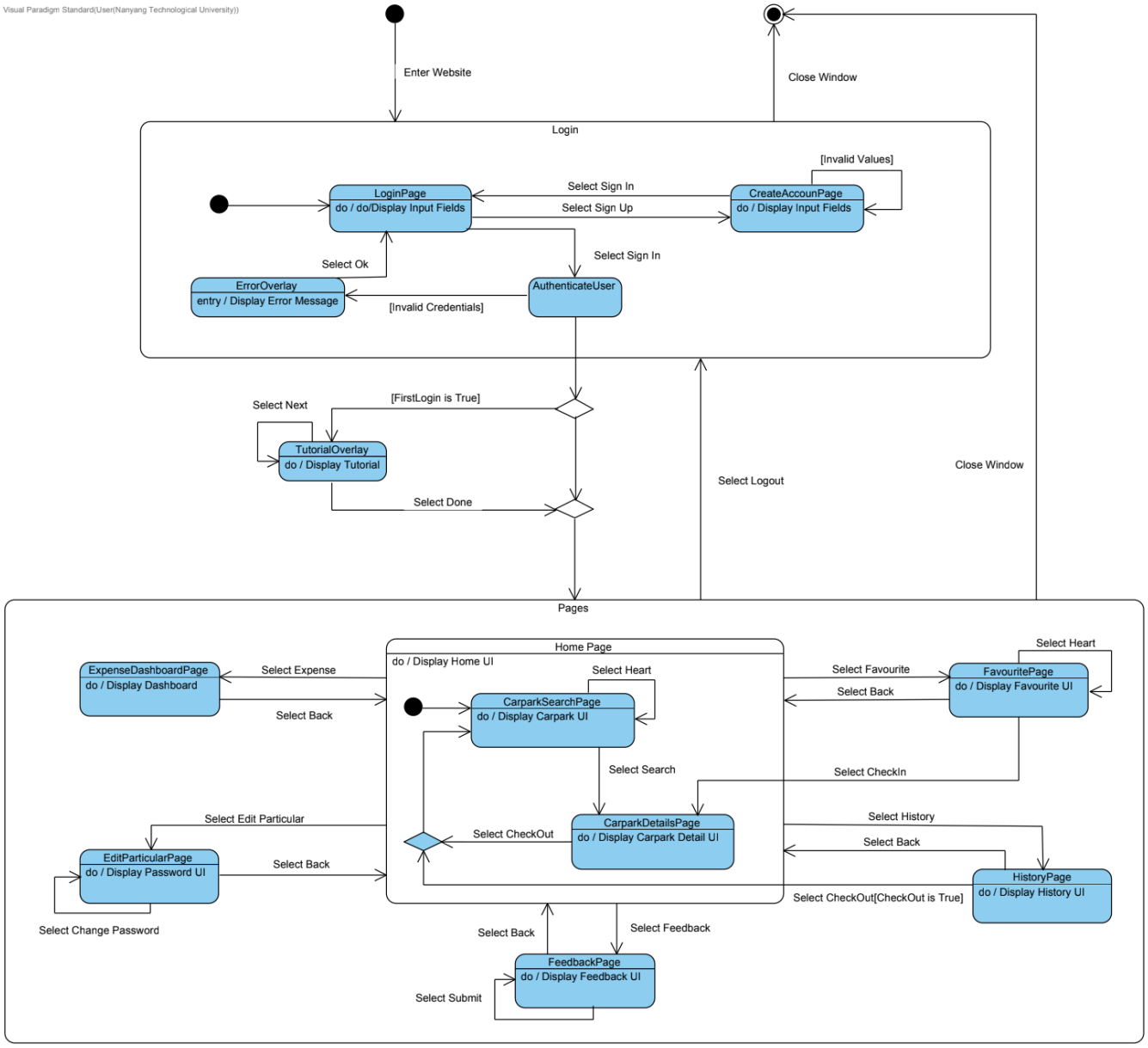
9.2 Class Diagram

High-definition image can be accessed via
https://drive.google.com/file/d/1L4QmYWmETe4czTviqirkAAY8C-vKjlGO/view?usp=drive_link



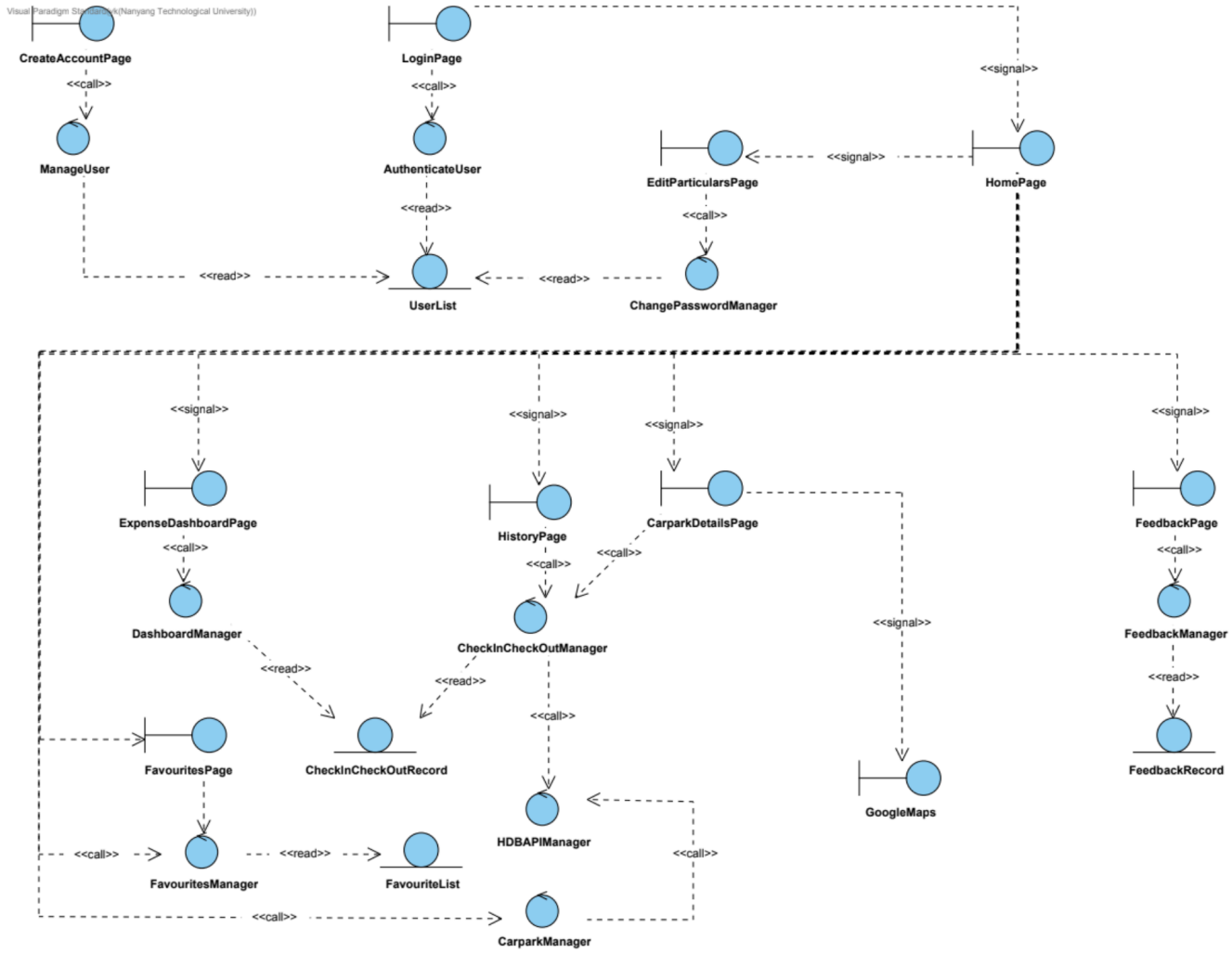
9.3 Dialog Map

High-definition image can be accessed via
https://drive.google.com/file/d/1a-QrUyn1xcjwOtQyCIobtaI0T5S0Jq1V/view?usp=drive_link



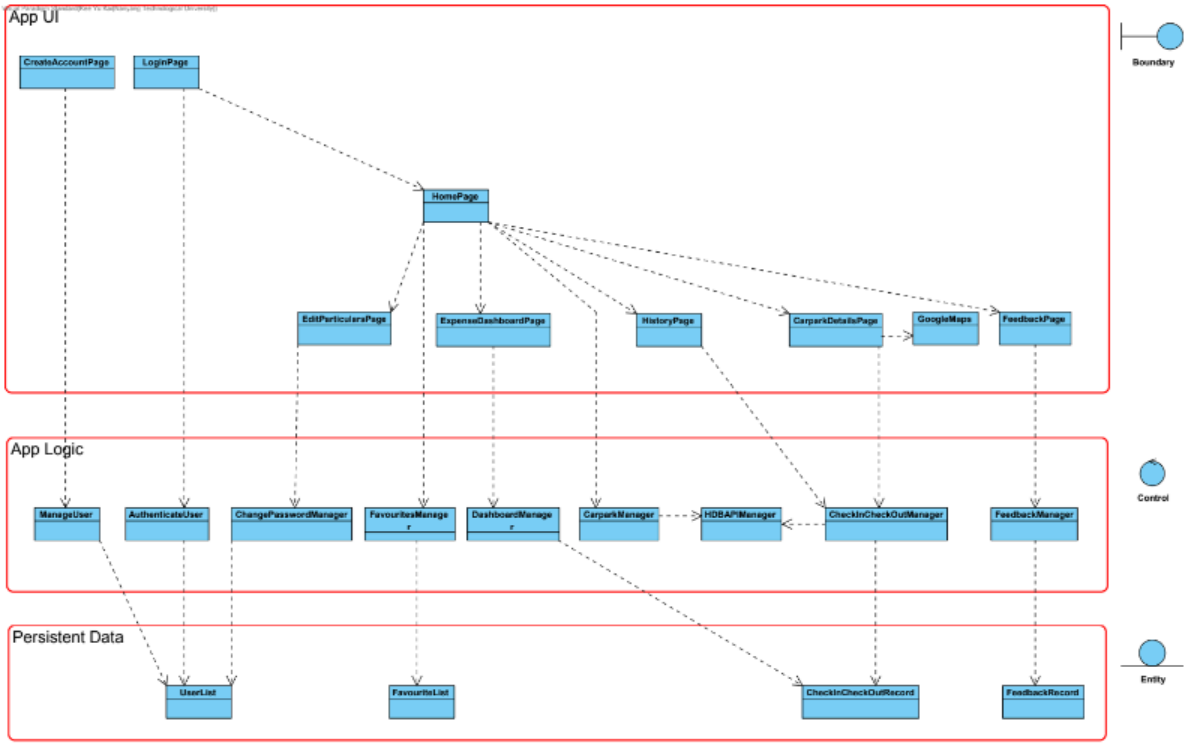
9.4 Key Boundary Class Diagram

High-definition image can be accessed via
https://drive.google.com/file/d/1jzk_-2fKcvqpu62lQaFe8CbXOq1IS41_/view?usp=drive_link



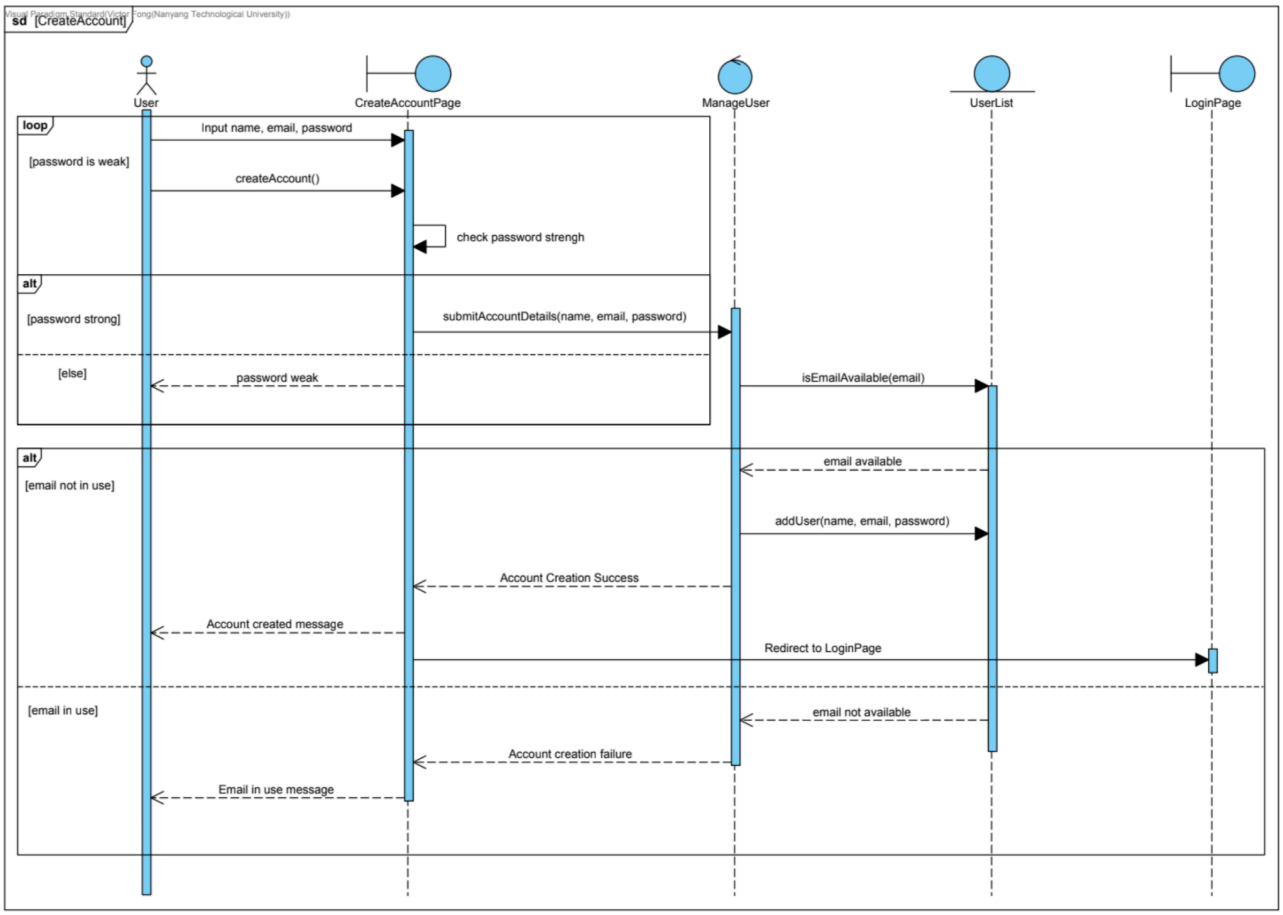
9.5 System Architecture Diagram

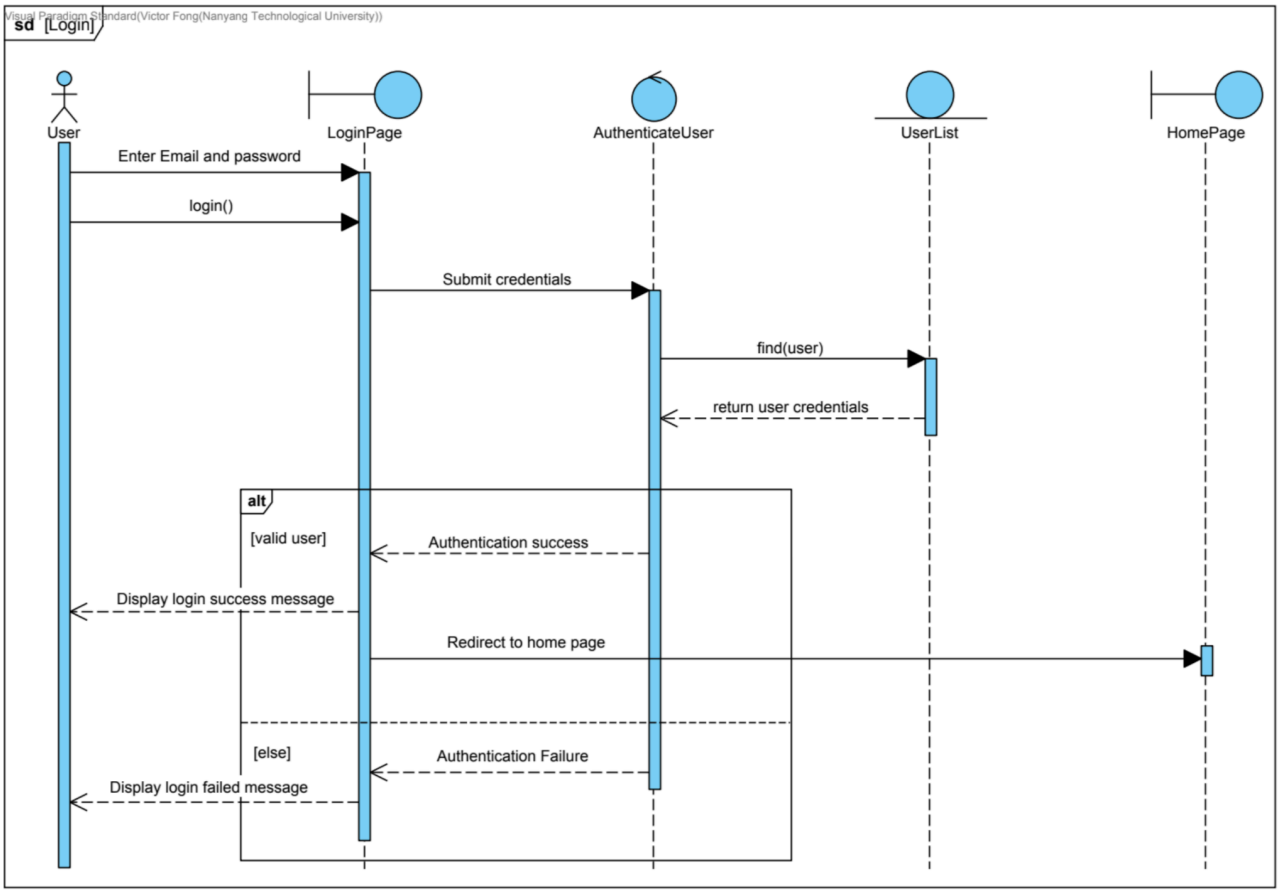
High-definition image can be accessed via
https://drive.google.com/file/d/1KbCruJ4oo_deHPni7V1N_STWQUE50u2y/view?usp=drive_link

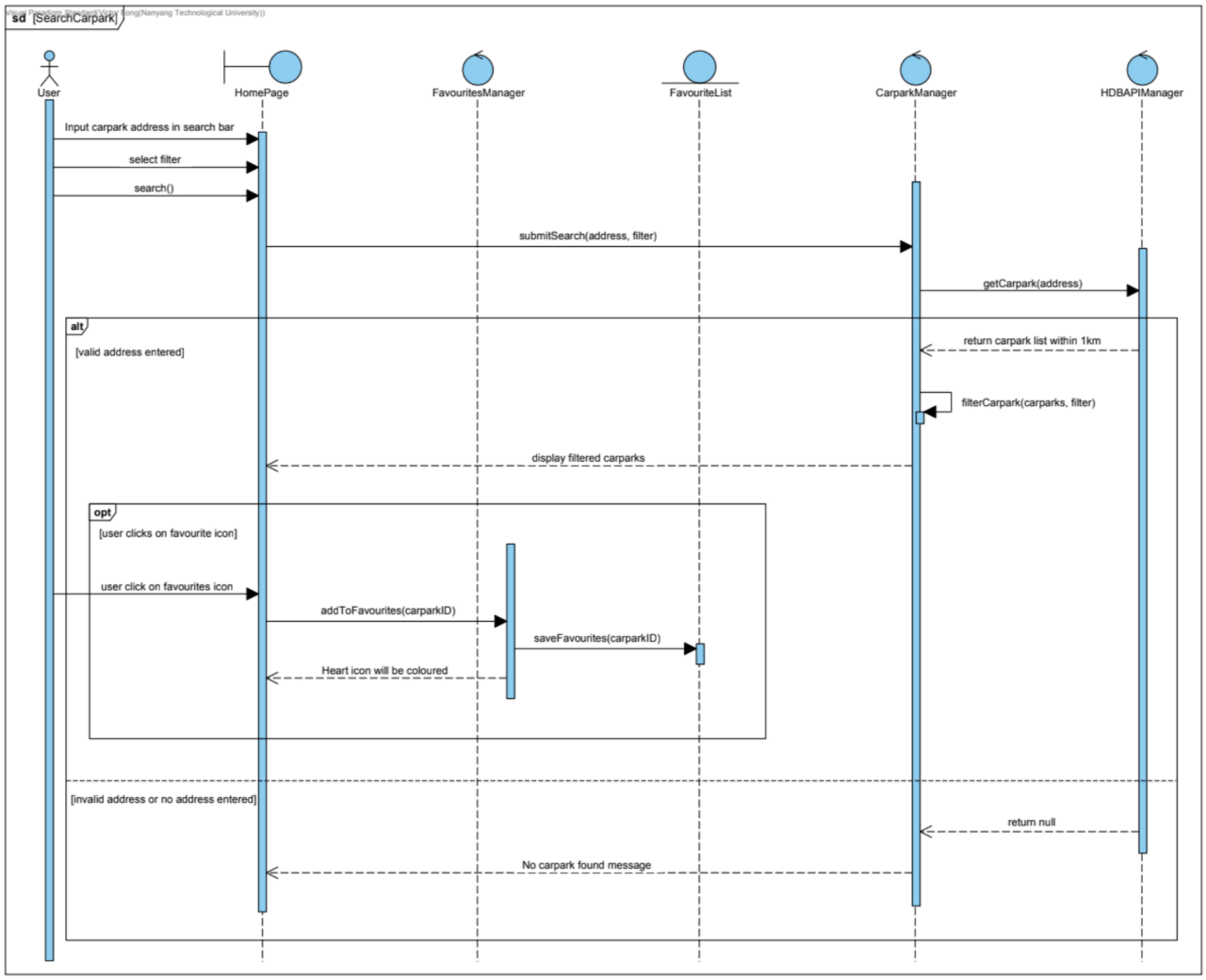


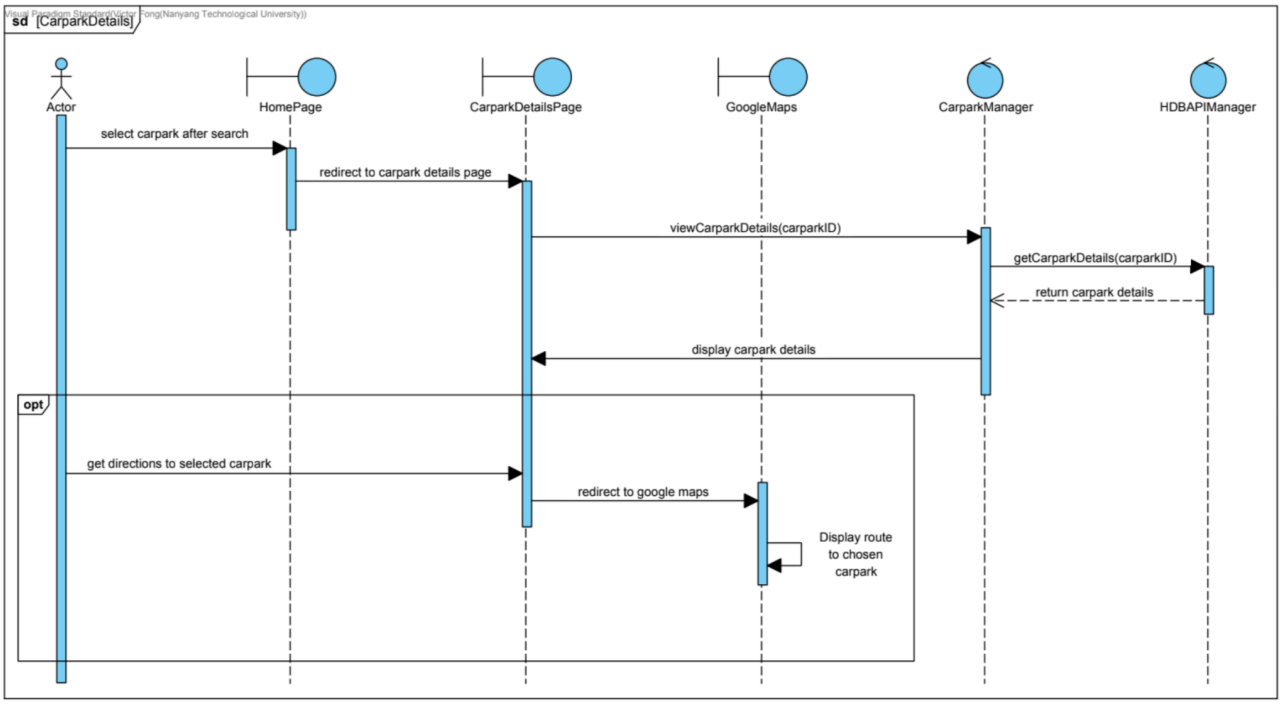
9.6 Sequence Diagrams

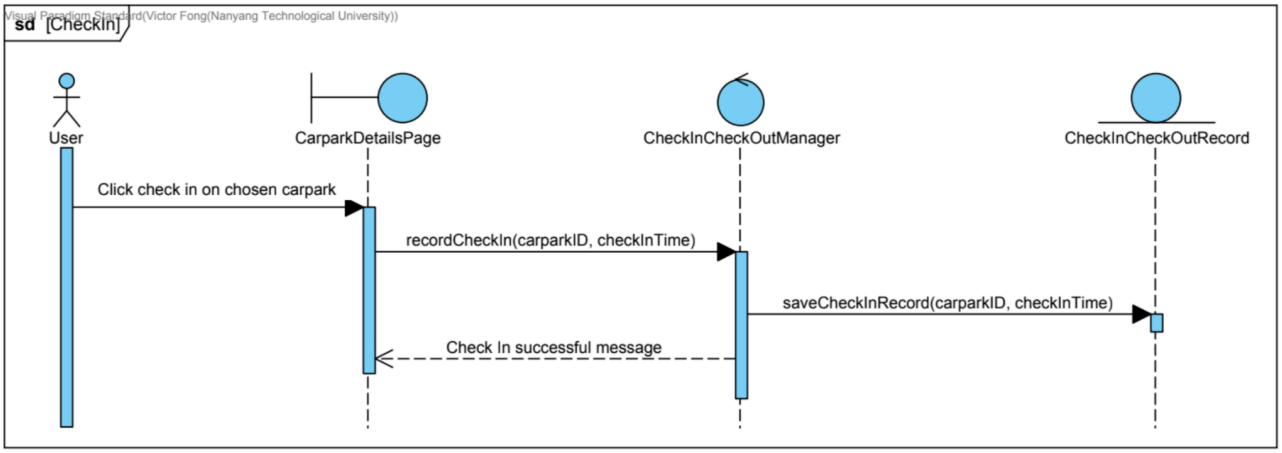
High-definition image can be accessed via
https://drive.google.com/file/d/1ZFI65R-CqkPGJbHmhXTCzDUQx_z2MdSF/view?usp=drive_link

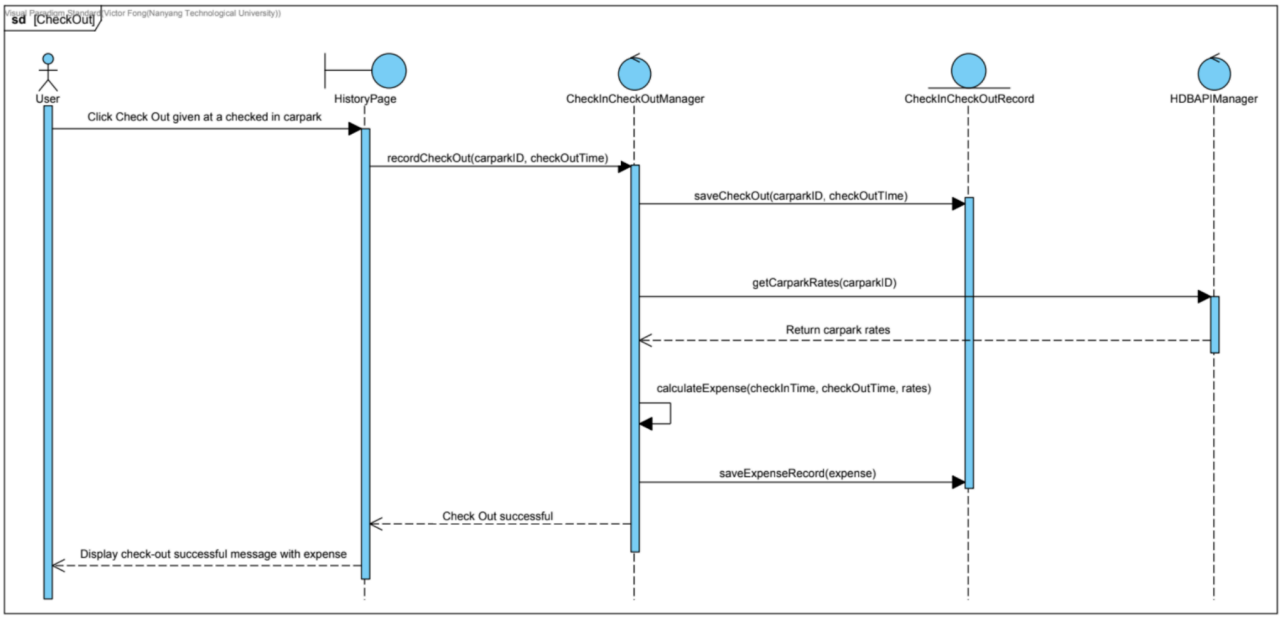


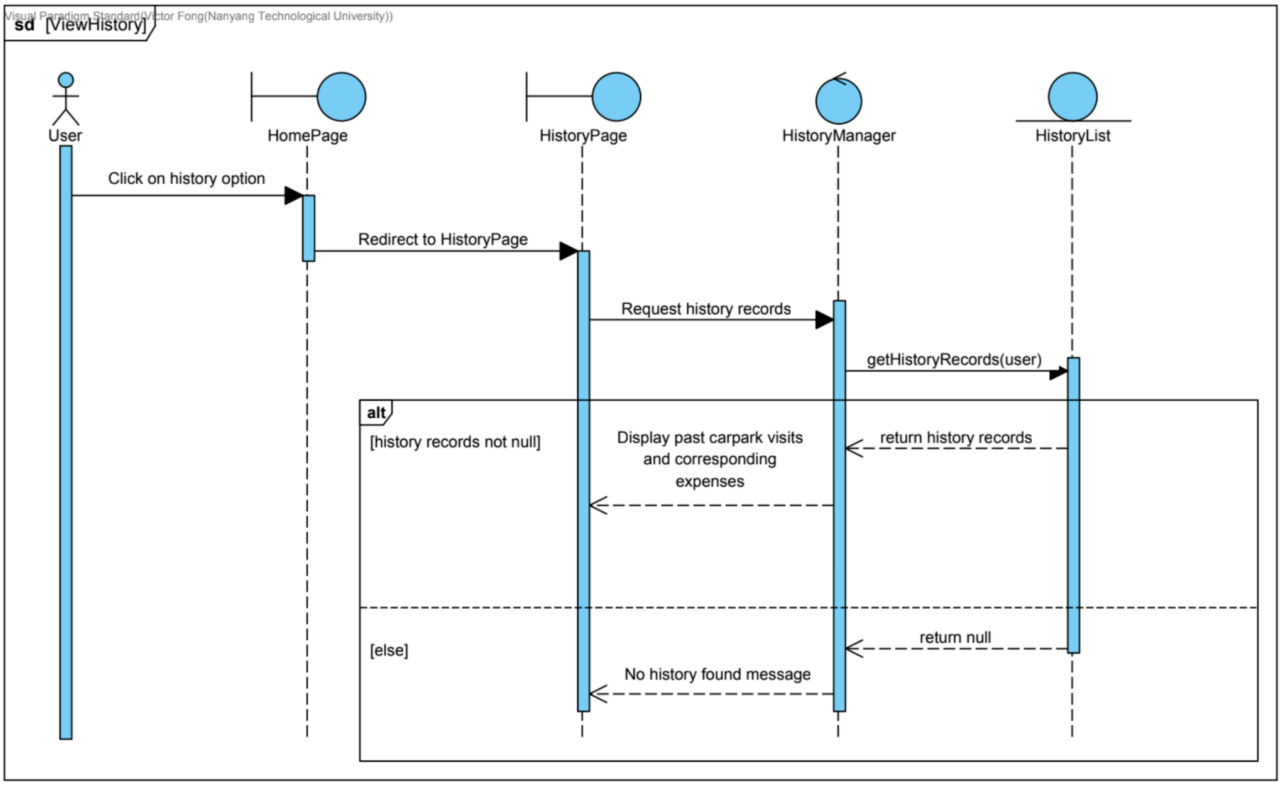

Visual Paradigm Standard(Victor Fong(Nanyang Technological University))
sd [Login]
User
LoginPage
AuthenticateUser
UserList
HomePage
Enter Email and password
login()
Submit credentials
find(user)
return user credentials
alt
[valid user]
Authentication success
Display login success message
Redirect to home page
[else]
Authentication Failure
Display login failed message

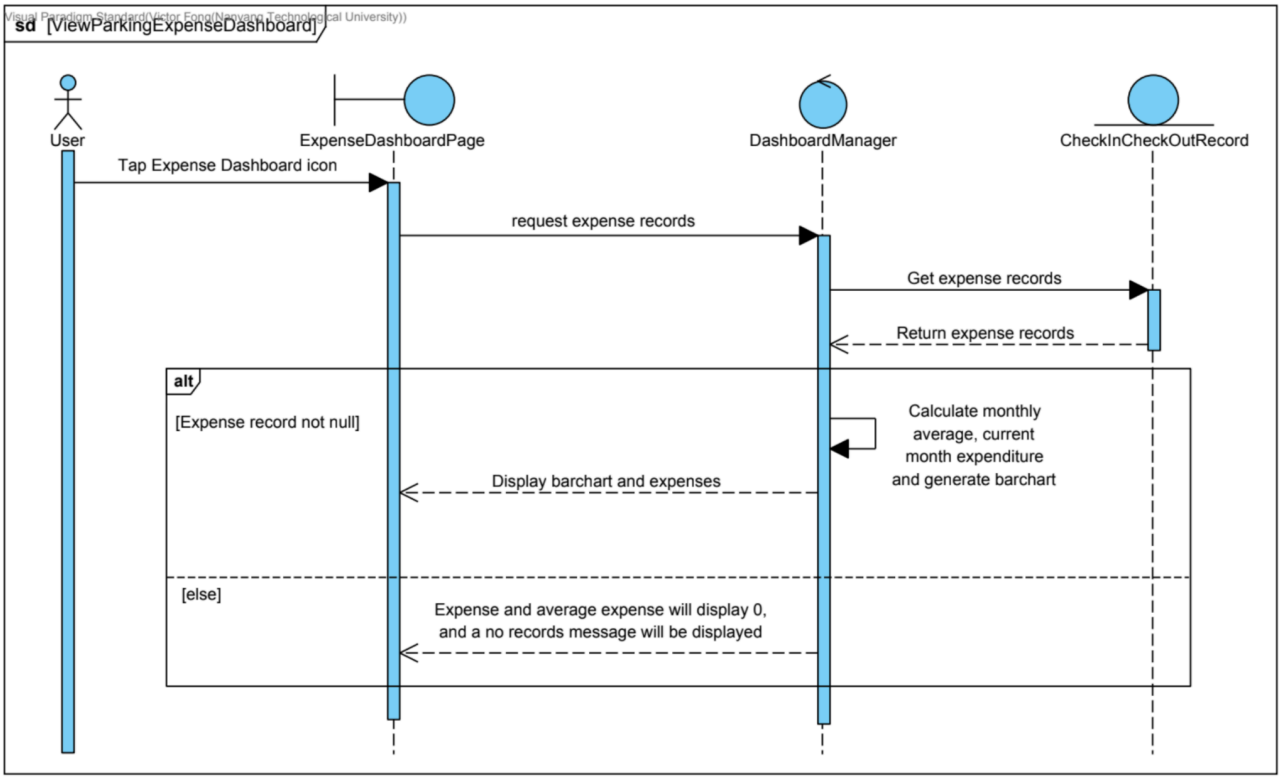

sd [SearchCarpark]
User
HomePage
FavouritesManager
FavouriteList
CarparkManager
HDBAPIManager
Input carpark address in search bar
select filter
search()
submitSearch(address, filter)
getCarpark(address)
alt
[valid address entered]
return carpark list within 1km
filterCarpark(carparks, filter)
display filtered carparks
opt
[user clicks on favourite icon]
user click on favourites icon
addToFavourites(carparkID)
saveFavourites(carparkID)
Heart icon will be coloured
[invalid address or no address entered]
return null
No carpark found message

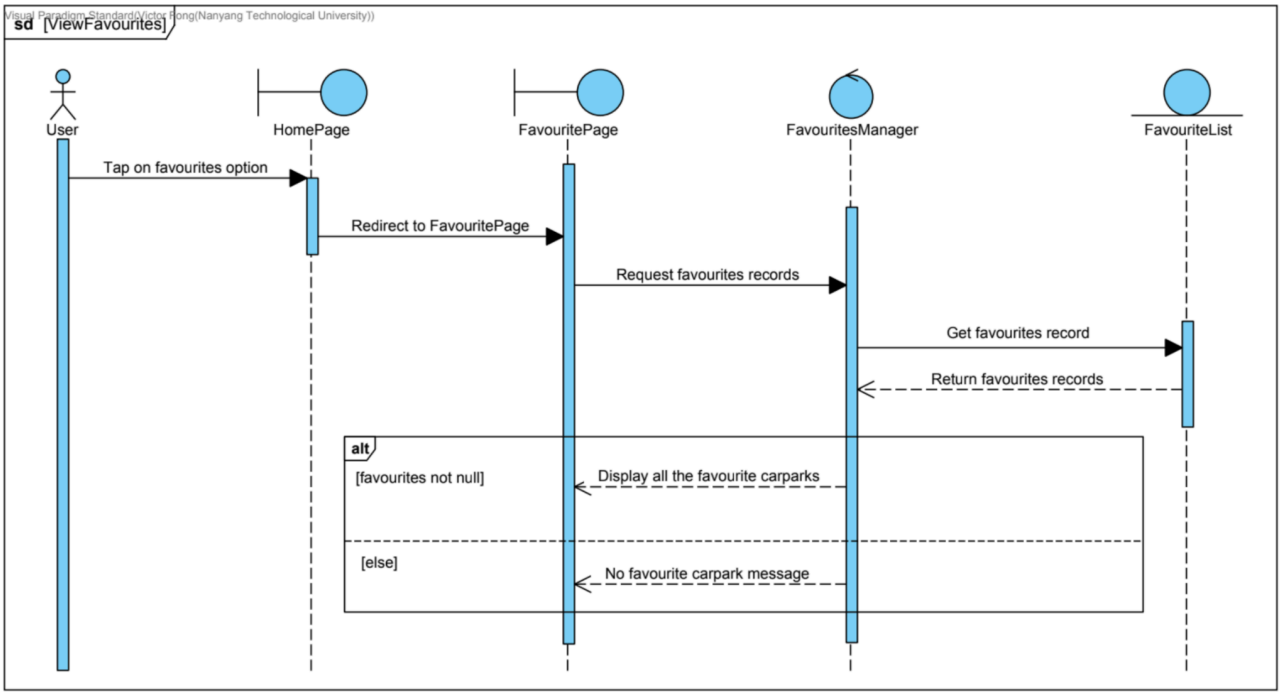

sd [CarparkDetails]
Actor
HomePage
CarparkDetailsPage
GoogleMaps
CarparkManager
HDBAPIManager
select carpark after search
redirect to carpark details page
viewCarparkDetails(carparkID)
getCarparkDetails(carparkID)
return carpark details
display carpark details
opt
get directions to selected carpark
redirect to google maps
Display route to chosen carpark

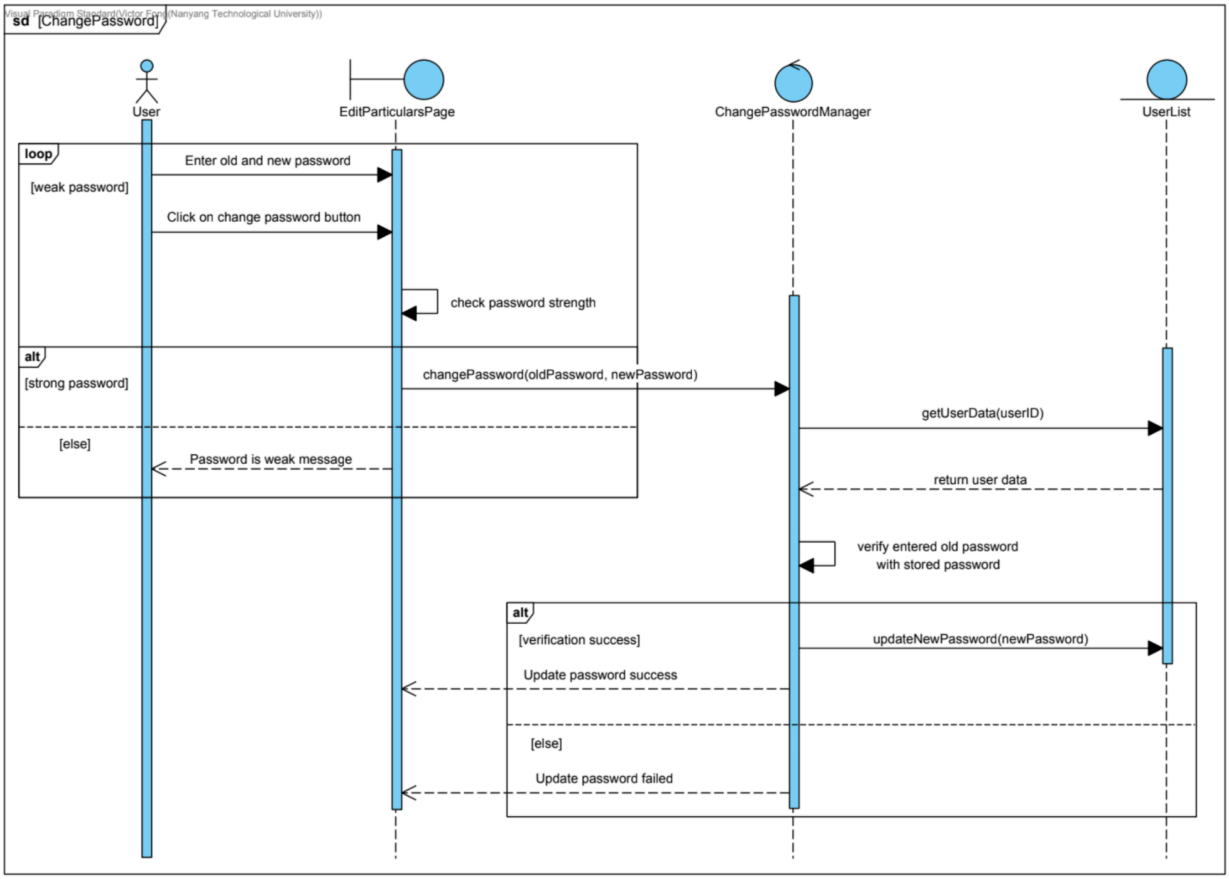

sd [CheckIn]
User
CarparkDetailsPage
CheckInCheckOutManager
CheckInCheckOutRecord
Click check in on chosen carpark
recordCheckIn(carparkID, checkInTime)
saveCheckInRecord(carparkID, checkInTime)
Check In successful message

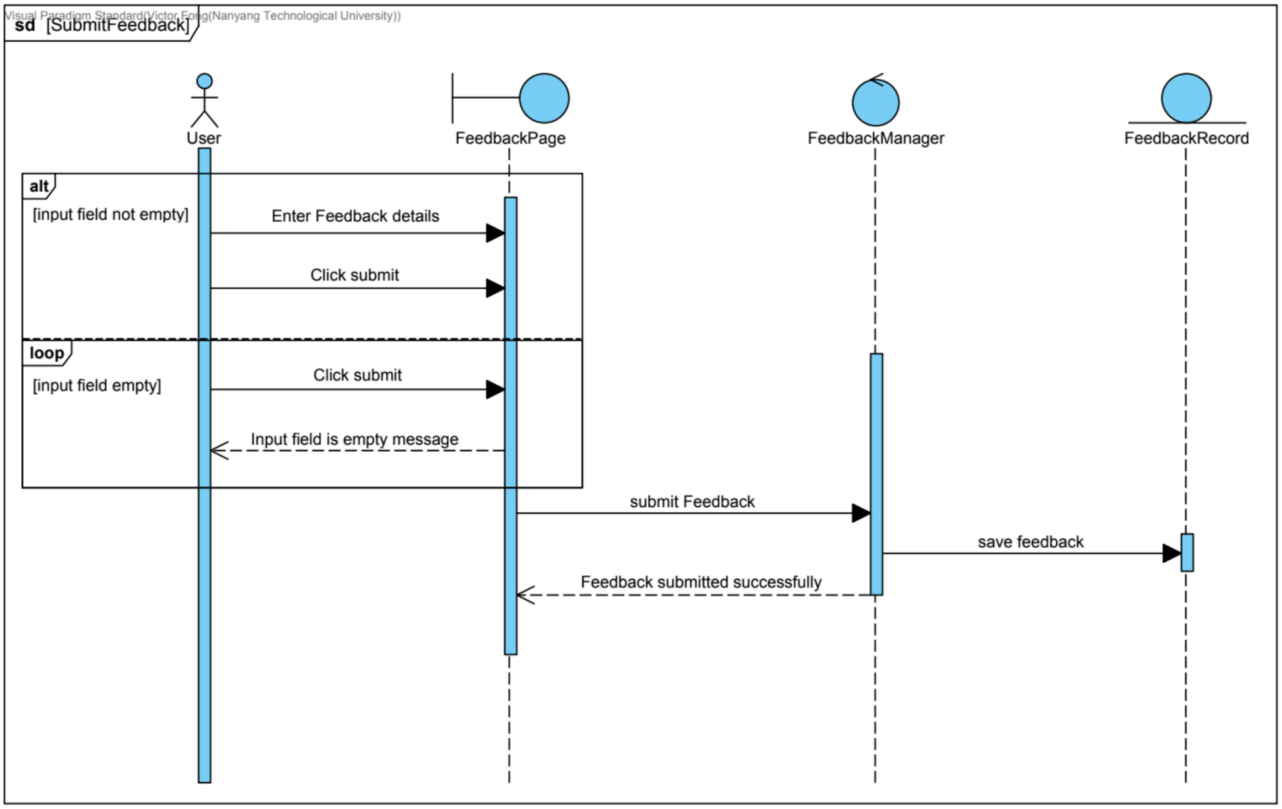

sd [CheckOut]
User
HistoryPage
CheckInCheckOutManager
CheckInCheckOutRecord
HDBAPIManager
Click Check Out given at a checked in carpark
recordCheckOut(carparkID, checkOutTime)
saveCheckOut(carparkID, checkOutTime)
getCarparkRates(carparkID)
Return carpark rates
calculateExpense(checkInTime, checkOutTime, rates)
saveExpenseRecord(expense)
Check Out successful
Display check-out successful message with expense


sd [ViewHistory]
User
HomePage
HistoryPage
HistoryManager
HistoryList
Click on history option
Redirect to HistoryPage
Request history records
getHistoryRecords(user)
alt
[history records not null]
Display past carpark visits and corresponding expenses
return history records
[else]
return null
No history found message


Visual Paradigm Standard(Victor Fong(Nanyang Technological University))
sd [ViewParkingExpenseDashboard]
User
ExpenseDashboardPage
DashboardManager
CheckInCheckOutRecord
Tap Expense Dashboard icon
request expense records
Get expense records
Return expense records
alt
[Expense record not null]
Calculate monthly average, current month expenditure and generate barchart
Display barchart and expenses
[else]
Expense and average expense will display 0, and a no records message will be displayed


Visual Paradigm Standard(Victor Fong(Nanyang Technological University))
sd [ViewFavourites]
User
HomePage
FavouritePage
FavouritesManager
FavouriteList
Tap on favourites option
Redirect to FavouritePage
Request favourites records
Get favourites record
Return favourites records
alt
[favourites not null]
Display all the favourite carparks
[else]
No favourite carpark message


sd [ChangePassword]
User
EditParticularsPage
ChangePasswordManager
UserList
loop
[weak password]
Enter old and new password
Click on change password button
check password strength
alt
[strong password]
changePassword(oldPassword, newPassword)
getUserData(userID)
[else]
Password is weak message
return user data
verify entered old password
with stored password
alt
[verification success]
updateNewPassword(newPassword)
Update password success
[else]
Update password failed



10. Appendix C: To Be Determined List

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