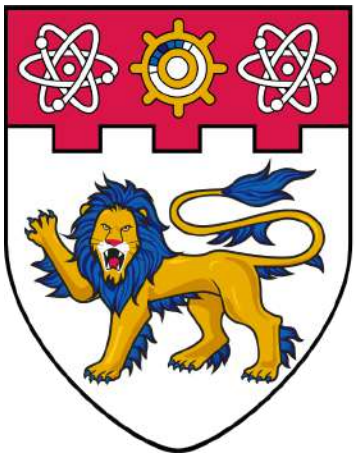


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

School of Computer Science and Engineering

SC2006 Software Engineering

Project Title: Wanderful

Wanderful

Tutorial Group: A42 - Group 5

Team Name: One Direction

Prepared by:

Jarel Tan Zhi Wen (U2121582H)

Jonathan Jiang WenXuan (U2121533J)

Tan Wei Yuan (U2122007H)

Sethi Aryan (U2123583E)

Dann, Wee Zi Jun (U2122790K)

Revision History

Name	Date	Reason For Changes	Version
Dann	9-04-2023	First Draft	1.0
Aryan, Wei Yuan, Dann, Jarel, Jonathan	16-04-2023	Final Draft	2.0

1. Requirement

1.1 Introduction

1.1.1 Purpose

The main purpose of the Software Requirements and Specifications (SRS) document is to offer a comprehensive and detailed account of Wanderful, an all-in-one web-based platform that enables drivers and passengers to arrange shared rides in Singapore.

The SRS describes the web application's goals and functionalities, its implementation details, its interfaces, and the limitations and conditions it must adhere to. Moreover, the document outlines the specifications for the project's design, analysis, and requirements.

1.1.2 Document Conventions

Title - font: Montserrat, size: 18, style: bold
Subtitle - font: Montserrat, size: 14, style: bold
Subheader - font: Montserrat, size 12, style: bold
Text - font: Montserrat, size: 10, style: normal
Line Height - 1.5

1.1.3 Intended Audience and Reading Suggestions

The document is intended for developers, project managers, investors, testers, and general users.

Developers and project managers are recommended to read the entirety of this document. Investors, testers, and general users are recommended to read the overall description and system features pertaining to the application in order to understand the features of Wanderful.

The preferred sequence of going through this SRS document would be to follow the sequence of the content page to develop a better flow of understanding.

1.1.4 Product Scope

Wanderful is a web-based platform that enables users to carpool with others to travel around Singapore easily. The application allows drivers to create carpool rides that passengers can then book, while passengers can search for available rides and find suitable options to carpool with. Furthermore, Wanderful provides information on nearby car parks, available parking lots, and nearby taxi services to assist drivers in finding convenient parking options and help passengers easily commute around Singapore.

With Wanderful, users can quickly and conveniently find carpool rides to get around Singapore and carry out their daily activities.

1.2 Overall Description

1.2.1 Product Perspective

With the Wanderful web application, drivers can set up carpooling trips that passengers can book, while passengers can search for carpooling opportunities. By checking out the rides available from drivers in their chosen starting area, users can travel around Singapore more efficiently.

The primary objective is to provide a one-stop platform that enables all commuters in Singapore to conveniently book carpools and taxi rides. By accessing the website, users can view available carpool rides and easily book their preferred options. To facilitate this process, the website integrates with the Taxi API and Car Park API, enabling users to quickly locate taxi alternatives and drivers to identify nearby parking facilities.

1.2.2 Product Functions

Major Functions:

- User Signup
- User Login
- Create Rides
    - Create Personal Car Rides
    - Create Taxi Rides
- Search Rides
- My Rides
    - My Rides
        - Ride Details
        - Carpool Group Chat
    - Ride Requests
    - FAQ
- Car Parks Availability
- Taxi Availability

Minor Functions:

- User Profile

1.2.3 User Classes and Characteristics

General users - Users of Wanderful are predominantly young and middle-aged adults ranging from 18 to 45 years old, with a balanced gender distribution and diverse income background.

In general, the demographic for our application should be users who reside in urbanised Singapore, who are minimally tech-savvy and familiar with using the internet.

1.2.4 Operating Environment

Wanderful can be operated on any modern web browser including desktop browsers and mobile browsers.

For development purposes, the application is developed on:

- React 18.2.0
- Vite 4.1.2
- Django 4.1.7

The frontend framework uses React.js. React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. React provides several features making it a popular choice for frontend development, including a Virtual DOM that makes it more efficient than traditional DOM. it also follows a component-based architecture that allows developers to create reusable and modular code.

Vite is a build tool and development server that is used to build modern web applications. It supports hot module replacement (HMR), which means that changes to the code can be reflected in the browser immediately without the need for a full page refresh.

The backend framework uses Django. Django is a high-level, robust and scalable open-source web framework that is written in Python. Django provides a lot of built-in features making it a favourable choice for backend development, including an Object-Relational Mapping system. Django also includes several built-in security features that protect against attacks like cross-site scripting (XSS) and cross-site request forgery (CSRF).

For testing the application locally on localhost, you need to make sure you have the relevant Django modules,, and then run the command ‘python manage.py runserver’. Also, you need to make sure you have NPM & Node.JS installed on your computer and then run the command “npm install” on the relevant directory to automatically install all the packages required. “npm run dev” will run a server on localhost, which you would be able to access on your browser.

1.2.5 Design and Implementation Constraints

Pick-up Locations and Destinations are postal code & address specific, hence spellings of the locations and postal codes must be correct in order for the input to be validated and accepted.

As ride details are stored using Django, it requires a relatively good internet connection to be able to see the ride details load.

1.2.6 User Documentation

A README file, as well as the code base, can be found on the official GitHub directory of the application: https://github.com/weiyuan12/Software-Eng . For any other queries, the team will provide the client with further help in terms of troubleshooting future errors.

1.2.7 Assumptions and Dependencies

Assumptions:
Wanderful requires a good, stable internet connection and an internet browser to work. Hence, the system assumes that all the requirements mentioned are met. If the user is not connected to the internet or does not have an internet browser, the web application would not work.

Dependencies:
Wanderful relies on the Google Maps API to embed a map that accurately reflects the user's pickup location and destination. In the absence of this API, the web application will not be able to display this information on the map, preventing users from visualising their pickup location and destination.

Similarly, the accuracy of Car Park availabilities on Wanderful is dependent on the API provided by the Urban Redevelopment Authority (URA). If the URA does not update the car park lot availability, we will not be able to retrieve the latest information.

Additionally, Wanderful's Taxi Availabilities feature depends on the API provided by data.gov.sg. If this API does not provide the latest information on taxis, we will not be able to retrieve it.

Our web application may face deployment and programming errors due to framework and library updates. To comply with the industry standards, we chose React as our primary framework, and JavaScript as our primary frontend language. For backend development, we chose Django with Python as the primary language.

1.3 External User Interface Requirements

1.3.1 User Interfaces

The textboxes on the web application must turn green to indicate that the user's input is valid. Upon signing in, the web application must display the navigation bar that contains the features of 'Create Rides', 'Search Rides', 'My Rides', 'Carparks', 'Taxis' and 'User Profile'.

1.3.2 Hardware Interfaces

The application is expected to run on any modern hardware devices such as smartphones, laptops and desktops that support a web browser with an internet connection.

1.3.3 Software Interfaces

- ReactJS v18.2.0 Javascript Framework is used to handle the overall navigation, responsiveness and reusable UI components.
- The system must have web browsers installed to support the application.
- Users must have NPM 9.5.0 installed to manage dependencies for Vite packages.
- URA car parks dataset - To retrieve Car Parks Dataset.
- data.gov.sg taxis dataset - To retrieve Taxis Dataset.
- Software requires Google Maps JavaScript API to be able to depict the pickup and destination locations of rides.
- Software requires Google Geocoder API to convert location names to longitude and latitude value to be used in Google Maps JavaScript API.

1.3.4 Communication Interfaces

In order to interface and load the website, Wanderful relies on web protocols like HTTP that are sent from the client/users. Additionally, to display the map and inputted locations, the web application requires an internet connection to load Google Maps JavaScript API and Geocoder API. Without an internet connection or access to these APIs, Wanderful will not be able to display the map or input locations.

1.4 Functional Requirements

1.4.1 User Profile and Authentication

User Registration

- Description and Priority:
  - The web application must allow the current user to sign up for an account if there is not one associated with their username.
  - Priority: High
- Stimulus/Response Sequences:
  - SEQ-1: User clicks on the 'Signup' navigation button and enters valid input.
  - SEQ-2: User clicks on the 'Signup' button to sign up for an account.
- Functional Requirements:
  - REQ-1: User must be able to enter the required information in the fields provided:
    - Required information includes:
      - First Name
      - Last Name
      - Email
      - Username
      - Password
      - Re-enter Password
  - REQ-2.1: If valid information is keyed in, the system will save the details in the database and redirect the user to the Main Menu.
  - REQ-2.2: If invalid information is keyed in, the system will notify the user of the invalid details that have been entered.

User Login

- Description and Priority:
    - The web application must allow the current user to log into the account that they created.
    - Priority: High
- Stimulus/Response Sequences:
    - SEQ-1: User clicks on the 'Login' navigation button and enters valid input.
    - SEQ-2: User clicks on the 'Login' button to log into the account.
- Functional Requirements:
    - REQ-1: User must be able to enter the required information in the fields provided:
        - Required information includes:
            - Username
            - Password
    - REQ-2.1: If valid information is keyed in, the system will log the user in and redirect the user to the Main Menu.
    - REQ-2.2: If invalid information is keyed in, the system will notify the user of the invalid details that have been entered and prompt the user to try again.

User Profile

- Description and Priority:
    - The web application must allow the current user to view their own user profile. User can edit their profile information or log out of their account.
    - Priority: Medium
- Stimulus/Response Sequences:
    - SEQ-1: User clicks on the 'Welcome *Username'* navigation button to view their own user profile.
    - SEQ-2.1: If user chooses to edit their profile information, user clicks on the 'Edit Profile' button and enter valid input.
    - SEQ-2.2: If user chooses to log out, user clicks on the 'Logout' button.
- Functional Requirements:
    - REQ-1.0: System must be able to navigate the user inputs to their profile page and show all the user profile information.
    - REQ-2.1: User must be able to edit the information in the field that they choose:
        - Fields that the user can choose to edit:
            - Date of Birth
            - Home Address
            - Biography
    - REQ-2.2: System must be able to log the user out and redirect the user to the Main Menu.

1.4.2 Main Menu

- Description and Priority:
    - Within the Main Menu, users can navigate to Create Ride page, Search Ride page, My Rides page, Car Parks page, Taxi page and User Profile page.
    - Priority: High
- Stimulus/Response Sequences:
    - SEQ-1.1: User clicks on the 'Create Rides' navigation button to create a drive or taxi ride.
    - SEQ-1.2: User clicks on the 'Search Rides' navigation button to search for rides.
    - SEQ-1.3: User clicks on 'My Rides' navigation button to view ride history or ride details or frequently asked questions (FAQ).
    - SEQ-1.4: User clicks on 'Carparks' navigation button to search for nearby car parks and car park details.
    - SEQ-1.5: User clicks on 'Taxis' navigation button to search for nearby taxis to carpool in.
    - SEQ-1.6: User clicks on 'Welcome *Username*' navigation button to edit the profile or log out of the account.
- Functional Requirements:
    - REQ-1: System must be able to allow users to navigate to the function that they require without any hiccups or friction.

1.4.3 Create Rides

- Description and Priority:
  - Allow users to create drives as a Driver or create taxi rides as a Rider.
  - Priority: High
- Stimulus/Response Sequences:
  - SEQ-1.0: User clicks on the 'Create Rides' navigation button to view Create Rides Page.
  - SEQ-2.1: If user clicks on the 'Personal Car' button, user must enter valid input.
  - SEQ-2.2: If user clicks on the 'Taxi' button, user must enter valid input.
  - SEQ-3.0: User clicks on the 'Submit' button to finalise the ride creation details.
- Functional Requirements:
  - REQ-1: System must be able to navigate the user to the Create Rides page with no hiccups or friction.
  - REQ-2.1: User must be able to enter the required information in the fields provided:
    - Required information includes:
      - Pick-up Location
      - Destination
      - Date and Time
      - Choose 'Recurring' or 'One-Time'
      - Number of Seats
  - REQ-2.2: User must be able to enter the required information in the fields provided:
    - Required information includes:
      - Pick-up Location
      - Destination
      - Date and Time
      - Number of Seats
  - REQ-3: System must be able to process the user input, create ride and add it into the database.

1.4.4 Search Rides

- Description and Priority:
  - Allow users to search and book rides.
  - Priority: High
- Stimulus/Response Sequences:
  - SEQ-1: User clicks on the 'Search Ride' navigation button to view the list of rides available
  - SEQ-2: User enters pickup location input to search for an available ride in that location.
  - SEQ-3: User chooses a ride that they would like to book and proceeds to click on the 'Request Ride' button which will redirect them to the Ride Details page.
  - SEQ-4.1: If user is certain that they would want to request the ride, the user clicks on the 'Confirm Request' button to send a request to the Driver who created the ride
  - SEQ-4.2: If user wants to continue browsing the list of available rides, the user clicks on the 'Go Back' button to return to the list of rides available.
- Functional Requirements:
  - REQ-1: System must be able to navigate the user to the Search Rides page with no hiccups or friction.
  - REQ-2: System must be able to process the user input and display the list of available rides that match the input.
  - REQ-3: System must be able to navigate the user to the corresponding Ride Details page with no hiccups or friction.
  - REQ-4.1: System must be able to process the user request and send the Ride Request to the corresponding driver to accept.
  - REQ-4.2: System must be able to navigate the user back to the Search Rides page with no hiccups or friction.

1.4.5 My Rides - MyRides

- Description and Priority:
    - Allow users to view their trip details and past trip details. Users can choose to chat with the driver and other passengers in the carpool ride.
    - Priority: High
- Stimulus/Response Sequences:
    - SEQ-1: User clicks on the 'My Rides' navigation button to view their rides, ride requests and FAQ.
    - SEQ-2.1: User clicks on the 'MyRides' category button to view their past trips or upcoming trips.
    - SEQ-2.2: User clicks on the 'View Details' button to view the ride details and the names of the driver and other passengers involved in the carpool ride.
    - SEQ-2.3: User clicks on the 'Chat' button to chat with the driver and other passengers involved in the carpool ride.
    - SEQ-2.4: User is able to view the chat history and send messages in the group chat.
- Functional Requirements:
    - REQ-1: System must be able to navigate the user to the My Rides page with no hiccups or friction.
    - REQ-2.1: System must be able to navigate the user to the MyRides category page with no hiccups or friction.
    - REQ-2.2: System must be able to navigate the user to the Ride Details page containing the full ride details and participating members with no hiccups or friction.
    - REQ-2.3: System must be able to navigate the user to the Chat page that shows the chat history of all carpool group members with no hiccups or friction.
    - REQ-2.4: User must be able to send messages into the chat. System must be able to record all messages sent and depict all messages to all carpool group members.

1.4.6 My Rides - Ride Requests

- Description and Priority:
    - Allow users to view their pending ride requests and accepted ride requests.
    - Priority: High
- Stimulus/Response Sequences:
    - SEQ-1: User clicks on the ‘My Rides’ navigation button to view their rides, ride requests and FAQ.
    - SEQ-2.1: User clicks on the ‘Ride Requests’ category button to view their pending ride requests list and accepted ride requests list.
    - SEQ-2.2.1: If user clicks on the ‘Accept’ button on a pending request, the ride request will be reflected in the accepted ride request list.
    - SEQ-2.2.2: If user clicks on the ‘Reject’ button on a pending request, the pending ride request will be deleted.
- Functional Requirements:
    - REQ-1: System must be able to navigate the user to the Ride Requests page with no hiccups or friction.
    - REQ-2.1: System must be able to navigate the user to the Ride Requests category page with no hiccups or friction.
    - REQ-2.2.1: Upon the user clicking on the ‘Accept’ button, system must be able to move the pending ride request to the accepted ride request.
    - REQ-2.2.2: Upon the user clicking on the ‘Reject’ button, system must be able to delete the pending ride request.

1.4.7 My Rides - FAQ

- Description and Priority:
  - Allow users to view the Frequently Asked Questions (FAQ).
  - Priority: Low
- Stimulus/Response Sequences:
  - SEQ-1: User clicks on the 'FAQ' category button to view the list of Frequently Asked Questions.
- Functional Requirements:
  - REQ-1: System must be able to navigate the user to the FAQ page with no hiccups or friction.

1.4.8 Car Park Availability - Filter by lots available

- Description and Priority:
  - Allow users to search for car parks and filter by lots available..
  - Priority: Medium
- Stimulus/Response Sequences:
  - SEQ-1: User clicks on the 'Carparks' navigation button and enters a valid input.
  - SEQ-2: User clicks on the search icon button to search for a car park near the input location.
  - SEQ-3: User clicks on the 'Filter by lots available' button to filter the car park lots available in descending order.
  - SEQ-4: User then clicks on the 'Show all on map' button to show all the car parks that are on the list.
- Functional Requirements:
  - REQ-1: System must be able to navigate the user to the Carparks page with no hiccups or friction and provide validation when user input is invalid.
  - REQ-2: System must be able to search for nearby car parks and list them.
  - REQ-3: System must be able to filter car parks by lots available in descending order.
  - REQ-4: System must be able to show car parks listed on the map.

1.4.9 Car Park Availability - Filter by distance

- Description and Priority:
    - Allow users to search for car parks and filter by distance.
    - Priority: Medium
- Stimulus/Response Sequences:
    - SEQ-1: User clicks on the 'Carparks' navigation button and enters a valid input.
    - SEQ-2: User clicks on the search icon button to search for a car park near the input location.
    - SEQ-3: User clicks on the 'Filter by distance' button to filter the car park lots available in descending order.
    - SEQ-4: User then clicks on the 'Show all on map' button to show all the car parks that are on the list.
- Functional Requirements:
    - REQ-1: System must be able to navigate the user to the Carparks page with no hiccups or friction and provide validation when user input is invalid.
    - REQ-2: System must be able to search for nearby car parks and list them.
    - REQ-3: System must be able to filter car parks by distance.
    - REQ-4: System must be able to show car parks listed on the map.

1.4.10 Taxi Availability

- Description and Priority:
  - Allow users to search for nearby taxis
  - Priority: Medium
- Stimulus/Response Sequences:
  - SEQ-1: User clicks on the 'Taxis' navigation button and enters a valid input.
  - SEQ-2: User clicks on the search icon button to search for taxis near the input location.
  - SEQ-3: User clicks on 'Show Taxis Nearby' button to show all taxis nearby on the map.
- Functional Requirements:
  - REQ-1: System must be able to navigate the user to the Taxis page with no hiccups or friction and provide validation when user input is invalid.
  - REQ-2: System must be able to search for nearby taxis.
  - REQ-3: System must be able to show all nearby taxis on the map.

1.5 Non-Functional Requirements

1.5.1 Performance Requirements

- The system must be able to return Django query results to the user within 3-8 seconds.
- The system must be able to support 10,000 concurrent user visits at the same time while maintaining optimal performance.
- The Google map should be able to render all locations within Singapore.
- During the initial loading of the webpage, the system shall load the page in less than 3 seconds over a 56 Kbps modem to ensure a fast page load and a good SEO score.
- The application will run on all web browsers.

1.5.2 Safety Requirements

- The system must be able to ensure data integrity within the system. This can be achieved by implementing measures such as error handling data validation, and correction.

1.5.3 Security Requirements

- Users are required to log in with their username and password.
- Users are to keep their password safe and not share it with any other people, applications, or websites under any circumstances.
- Users are reminded not to leak their personal information such as addresses over the web application.
- The system must ensure that sensitive or personal data can only be accessed by authorised users.
- The system must ensure that data is protected in accordance with PDPA.

1.5.4 Reliability Requirements

- Within 3 minutes of a system reboot, the entire system must be restored to its full functionality.
- The system should not experience critical failure under normal usage in 95% of use cases.
- The system uptime must be 99% or higher.
- The system must provide up-to-date information on the car park availability and taxi availability.

1.5.5 Usability Requirements

- The user must be able to navigate smoothly between pages

1.5.6 Localisation Requirements

- The system must be able to display date format in DD/MM/YYYY HH:MM format
- The system must be able to read addresses and postal codes in Singapore.

1.5.7 Software Quality Attributes

Security

Whenever a new user is created, a token is generated as part of the account creation process. This token is essentially a piece of information that proves the user's identity and authorisation to access certain parts of the web application. This token is utilised for every REST API on our backend to ensure that only authorised users are able to access it.

Encapsulation, Abstraction

The Entity Boundary Class structure is followed throughout our source doe by creating a class that acts as an interface between the software system's internal entities and the external world. This structure promotes clear separation and exposes a simpler and more intuitive interface.

Low Coupling

The design of the components, pages, and functions ensures that they are not heavily interconnected, which significantly reduces the dependencies between them. As a result, if a single file were to be modified or a new use case was to be introduced, the impact on the overall system would be minimised.

Maintainability and Readability

Our system are organised such that our project has a clear and consistent directory structure. Appropriate Javascript filenames were used for each web page and proper JSdoc documentation and comments were made on the source code.

Resource Optimisation

Our system uses many ways to optimise resources to save and improve latency. We use UseEffect hooks to make function calls only when a certain state has been changed. This cuts downs on the total number of API calls that are being made because the API calls do not run when the page rerenders, hence saving resources.

1.5.8 Benefits of our Technology Stack

Our technology stack comprises of popular web development frameworks commonly used by web developers in the field, which makes it easy for clients to perform future maintenance and upgrades on the web application without requiring specialised developers. Additionally, we use React.js, which allows us to create completely modular and reusable UI components, such as a Navigation bar that can be used on every webpage without needing to copy and paste the same code.

Furthermore, our Django backend follows the “don’t repeat yourself” (DRY) principle, enabling developers to write code more efficiently and reduce the amount of time and effort needed to build complex applications. Additionally, we use Vite, a build tool for JavaScript applications that offers fast and efficient bundling of code using modern ES modules. Vite also provides hot module replacement, allowing developers to see real-time changes without reloading the entire application, and a flexible plugin system that can be used to extend its functionality and customise build processes, and make compiling faster.

1.5.9 Business Rules

a. Developer
    i. Developers can remove users that do not abide by the web application guidelines
    ii. Developers can remove rides that breaches the web application guidelines
b. User
    i. Initiators can make adjustments to the profile details, such as Date of Birth, Home Address and Biography
    ii. Initiators can manage the ride requests
        1. Accept or reject the ride request

1.6 Use Case Diagram

1.7 Use Case Descriptions

Use Case ID:	**1.0**		
Use Case Name:	Sign-Up		
Created By:	Tan Wei Yuan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by ***User***
Description:	*User* creates an account in our system database. To create an account, users must provide a unique username and valid password.
Preconditions:	1. *User* has not created an account
Postconditions:	1. New *User* account is created 2. *User* account is added to database
Priority:	High
Frequency of Use:	Once per lifetime
Flow of Events:	1. *User* Clicks on the Signup button 2. *User* is prompted to input first name, last name, email, username, and password and re-enter the password. 3. System checks if an account already exists with the username mentioned 4. System verifies the availability of username in database 5. System ensures password meets the requirements: a. 1 Uppercase alphabet

	b. 1 Lowercase alphabet c. Between 8 - 24 characters 6. System creates a *User* account and adds information about the created *User* account into the database 7. User is moved to the login screen
Alternative Flows:	AF-S1: Entered Username is already in the database and is not unique 1. The UI displays the message “Username is already taken. Please enter another username.” 2. UI returns to Step 1 AF-S1: Entered Password does not meet any of the 3 requirements 1. The UI displays the message “Password does not meet requirements. Please Enter a password with 1 Uppercase, 1 Lowercase and a length of 8 - 24 Characters.” 2. UI returns to Step 1 AF-S1: Entered Re-enter Password does not match Password 1. The UI displays the message “Password does not match. Please re-enter password.” 2. UI returns to Step 1 AF-S1: Entered Email is invalid 1. The UI displays the message “Email is invalid. Please enter a valid Email Address” 2. UI returns to Step 1
Exceptions:	-
Includes:	-
Notes and Issues:	*User* creates an account in our system database. To create an account, users must proved a unique username and valid password.

Use Case ID:	2.0		
Use Case Name:	Login		
Created By:	Tan Wei Yuan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by ***User***
Description:	*User* logs in to his account. For the *User* to log in to an account, the username and password must match the database records.
Preconditions:	1. *User* has already created an account 2. Username and Password are entered correctly
Postconditions:	*User* is logged in to the account
Priority:	High
Frequency of Use:	Low
Flow of Events:	1. *User* clicks on the login button 2. *User* is prompted to input a username and password 3. System verifies the account details with the database *4.* *User* is logged into the account if the username and passwords match
Alternative Flows:	AF-S1: Username is not present in the database (Either *User* has entered the wrong username or the account does not exist) 1. UI displays the message “Account does not exist. Please check login details or Sign up with a new account.”

	2. UI returns to Step 1 AF-S1: Username is present in the database but the password does not match 1. UI displays the message “Password is invalid. Please re-enter the password” 2. UI returns to Step 1
Exceptions:	EX1: *User* enters an invalid password three times 1. UI displays the message “Too many attempts. Please try again later.” 2. UI returns to Step 1
Includes:	-
Notes and Issues:	-

Use Case ID:	3.0		
Use Case Name:	Create Ride		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by ***User (Driver)***
Description:	*Driver* creates a ride as a Personal Car driver or Taxi driver. To do so, *Driver* has to fill up the required details of the ride such as pickup location, end location, date & time of ride, recurring or one-time ride, and the number of available seats on the ride.
Preconditions:	1. *Driver* has an account 2. *Driver* has successfully Logged into the account
Postconditions:	The database containing all the rides has been updated with a new record
Priority:	High
Frequency of Use:	High
Flow of Events:	1. *User* clicks on the “Create Ride” Button in the navigation bar 2. The “Create Ride UI” is overlaid onto the Google map behind *3.* *Driver* is prompted to enter ride details: a. Starting Point: (Pickup location) b. Ending Point: (Destination) c. Date & Time: (DD/MM/YYYY HH:MM) d. Available Seats: (Number of Seats)

	e. Recurring ride: [Recurring / One-Time] f. Type of Ride: (Personal Car / Taxi) 4. The user must validate the start and end location by clicking on the “Verify” button next to their start and end input 5. If validation is successful, the “Verify” button will become a green tick 6. *Driver* clicks on the Create Ride Button and the System will check: a. Both start and end locations have a green tick indicating they are verified b. Seats are more than 0 7. The system will display the start and end locations as markers on the map along with the best path calculated. 8. The System will add the details of the ride into the database 9. UI displays the message “Ride Created Successfully”
Alternative Flows:	AF-S5: Verification not successful - An alert will display prompting the user to re-enter a valid location AF-S5: User changes the location name after a successful verification - The green tick will change back to the “Verify” button which would require the user to re-verify the new location AF-S6: Start and/or End location are not verified - The system returns an alert requiring both locations to be verified AF-S6: Available Seats entered is 0 - The system returns an alert requiring the Available Seats entered to be more than 0
Exceptions:	-
Includes:	Use Case 8.0: validateLocation(location)
Notes and Issues:	Slight latency of around 200ms due to API calls to validate location and generate a path

Use Case ID:	4.0		
Use Case Name:	Search & Request Ride		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by ***User*** *(**Passenger**)*
Description:	*User* searches for available rides as a *Passenger* When on the page to select a ride, the system will display all available rides in the system *Passenger* is then able to select the ride to view the details of the ride and are able to request for a ride with that *user*.
Preconditions:	1. *Passenger* has an account 2. *Passenger* has successfully logged into the account
Postconditions:	A request is sent to the *Driver* who made the ride chosen by the *Passenger*.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. *User* clicks on the “Search Ride” Button in the navigation bar 2. The “Search Ride UI” is overlaid onto the google map behind 3. The user can enter a location into the search bar 4. System displays brief header information of all created rides that have a similar starting location 5. *Passenger* selects a ride using the “Request Ride” button

	6. System displays the full information of the selected ride including: a. Start Location b. End Location c. Date & Time d. Number of Seats Available e. Recurring ride / One Time ride f. Type of ride (Personal Car / Taxi) 7. Markers are placed on the map corresponding to the start and end location along with the path between them for a more visual representation 8. *Passenger* may confirm the request or go back by clicking on the "Confirm request" or "Go back" buttons 9. If confirmed, *System* Records the Details in the database and sends a ride request to the *Driver*
Alternative Flows:	AF-S1: No Existing Rides 1. UI displays message "No available rides" 2. Use Case does not continue to step 3 AF-S8: User clicks on the "Go back" button 1. Use Case goes back to step 4 2. No update to the database
Exceptions:	-
Includes:	-
Notes and Issues:	-

Use Case ID:	5.0		
Use Case Name:	Receive Passenger Request		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Handled by ***User*** (***Driver)***
Description:	The *Driver* receives a request made by the *Passenger* to be a part of the ride scheduled, and inspects it
Preconditions:	1. The *Passenger* has scheduled a ride 2. A *Passenger* has made a request to be a part of the same ride
Postconditions:	The *Driver* is given the decision of either accepting or rejecting the request
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The *Driver* receives a request from the *passenger* 2. The *Driver* goes through the passenger's details and chooses to either accepts or rejects the request
Alternative Flows:	-
Exceptions:	-
Includes:	-

Notes and Issues:	-

Use Case ID:	6.0		
Use Case Name:	Handle Passenger Request		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	Handled by ***User (Driver)***
Description:	The *Driver* accepts or rejects a request made by the *Passenger* to be a part of the ride scheduled
Preconditions:	The *Driver* received a request made by a *Passenger* to be a part of the ride scheduled by the *Driver*
Postconditions:	The *Passenger* is able to see if their request is accepted, and then a communication link is established between the *passenger* and the *driver*
Priority:	High
Frequency of Use:	High
Flow of Events:	1. After inspecting the request, the *Driver* either accepts or rejects the request 2. If the request is accepted, a communication link is established between the *driver* and the *Passenger* 3. If the request is rejected, the ride request will be deleted
Alternative Flows:	-

Exceptions:	If the *Driver* fails to respond to the request of the *Passenger*, the request automatically gets declined
Includes:	-
Notes and Issues:	-

Use Case ID:	7.0		
Use Case Name:	Chat		
Created By:	Tan Wei Yuan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	***Driver*** and ***Passenger(s)***
Description:	To allow the *driver* to communicate with the passenger. This will help facilitate carpooling, allowing all passengers to communicate before the ride
Preconditions:	1. *Driver* must have accepted the ride request 2. Ride confirmed between *driver* and *passenger* 3. Chat is accessed from the MyRides page
Postconditions:	1. All messages sent is immediately stored in the database and displayed to the other *users* in the chat
Priority:	Medium
Frequency of Use:	High
Flow of Events:	1. A ride is confirmed between *driver* and *passenger* 2. *User* clicks on “view details” button of the confirmed ride, which displays the usernames of all other *Users* in the chat 3. *User* clicks on the chat button which loads all prior chatting history *4.* *User* inputs text and clicks on the send icon to send the message 5. Message is saved in the database and updated to all other *Users* in the chat

Alternative Flows:	AF - S1: First passenger confirmed to the ride 1. Database initialises a new conversation AF-S1: Conversation is already created, and another passenger is accepted 1. Passenger is added into the conversation, and is displayed all previous communication history
Exceptions:	-
Includes:	-
Notes and Issues:	-

Use Case ID:	8.0		
Use Case Name:	Location Validation		
Created By:	Tan Wei Yuan	Last Updated By:	Dann Wee
Date Created:	07/02/2023	Date Last Updated:	14/04/2023

Actor:	***System***
Description:	validateLocation takes in a string input containing a location name. It calls the google maps validate API to validate the input.
Preconditions:	1. Called from: a. Use Case 3.0: Create Ride b. Use Case 9.0: Taxi Availability c. Use Case 10.0: Carpark Availability
Postconditions:	1. GeoCode containing the coordinates in the form {lat, lng} is returned to the calling function if the validation is successful 2. An alert is displayed on the screen and nothing is returned to the calling function if the validation is unsuccessful
Priority:	High
Frequency of Use:	High
Flow of Events:	1. validateLocation(location) use case is called from another use case 2. It makes a request to the Google Maps Validate API, passing in the location

	3. If the input is a valid location, it returns a GeoCode containing the coordinates for the calling function to process. 4. If the input is an invalid location, it displays an alert asking the user to reenter a valid location
Alternative Flows:	-
Exceptions:	-
Includes:	-
Notes and Issues:	Requires around 100ms for the entire validation call

Use Case ID:	9.0		
Use Case Name:	Taxi Availability		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	21/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by *User*
Description:	*User* enters a location into the search bar, and the system will display all nearby taxis on the map
Preconditions:	1. *User* has an account 2. *User* has successfully logged into account 3. *User* enters a valid search location
Postconditions:	1. Location Markers are placed on the map indicating the locations of the taxis 2. The markers will remain on the map until a. User searches for a different location b. User clicks away from the taxis availability tab
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. *User* clicks on the “Taxi” Button in the navigation bar 2. The “Taxi UI” is overlaid onto the Google map behind 3. The user enters a location into the search bar 4. The system returns the real-time location of all taxis within 1km

	5. Once returned, the user can click on the “Show taxis nearby” button which drops markers on each taxi’s current location on the map.
Alternative Flows:	AF-S3: User enters an invalid location - validateLocation( ) returns an alert “Please enter a valid location” - No taxis are returned AF-S3: No available taxis nearby - No taxis are displayed to the user
Exceptions:	-
Includes:	Use Case 8.0: validateLocation(location) where location is a string containing the location entered by the user
Notes and Issues:	Slight latency of around 200ms due to API calls to validate location and get all taxis

Use Case ID:	10.0		
Use Case Name:	Carpark Availability		
Created By:	Sethi Aryan	Last Updated By:	Dann Wee
Date Created:	21/02/2023	Date Last Updated:	14/04/2023

Actor:	Initiated by *User*
Description:	*User* enters a location into the search bar, and the system will display all car parks on the map. The results can be filtered by distance or carpark lots available
Preconditions:	1. *User* has an account 2. *User* has successfully logged into account 3. *User* enters a valid search location
Postconditions:	1. Markers are placed on the map indicating the locations of the car parks 2. The markers will remain on the map until a. User searches for a different location b. The car park filter changes c. User clicks away from the car park availability tab
Priority:	Medium
Frequency of Use:	Medium
Flow of Events:	1. *User* clicks on the “Carpark Availability” Button in the navigation bar 2. The “Carpark Availability UI” is overlaid onto the Google map behind 3. The user enters a location into the search bar

	4. The system returns all car parks within 1km and automatically filter by car park lots available 5. The user can toggle between filter by distance and filter by lots available by clicking on the appropriate filter buttons 6. Once a filter is selected, the user can click the “Show all on map” button which drops markers on the car park’s location on the map. 7. Clicking each marker will display the car park number and the lots available
Alternative Flows:	AF-S3: User enters an invalid location - validateLocation( ) returns an alert “Please enter a valid location” - No car parks are returned AF-S3: No available car parks nearby - No car parks are displayed to the user
Exceptions:	-
Includes:	Use Case 8.0: validateLocation(location) where location is a string containing the location entered by the user
Notes and Issues:	Slight latency of around 200ms due to API calls to validate location and get all carparks

1.8 Data Dictionary

Term	**Definition**
User	The person using our application
Passenger	A User booking a Ride
Driver	A User providing a Ride
Username	A string of characters to uniquely identify a user
Password	A string of 8 to 24 characters containing at least 1 uppercase and 1 lowercase character
Date-Time	Exact Date and Time of a ride in format (DD/MM/YYYY HH:MM)
Ride	Journey from pickup location to destination
Recurring Ride	After the ride is completed, a new ride is created with the same details for the next week for user convenience
One-Time Ride	The ride created is just for the one instance

1.9 UI Mockups

The complete initial UI Mockups can be found at the following Figma link: https://www.figma.com/file/rkINFbuGfItQhaXMO12MeE/UI-Mockups?node-id=0%3A1&t=P4WCyiqf3waxo18Y-1

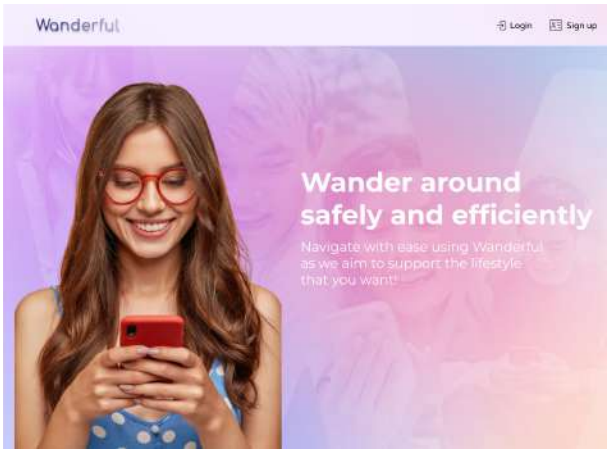


Fig 1.1: *Main Menu*

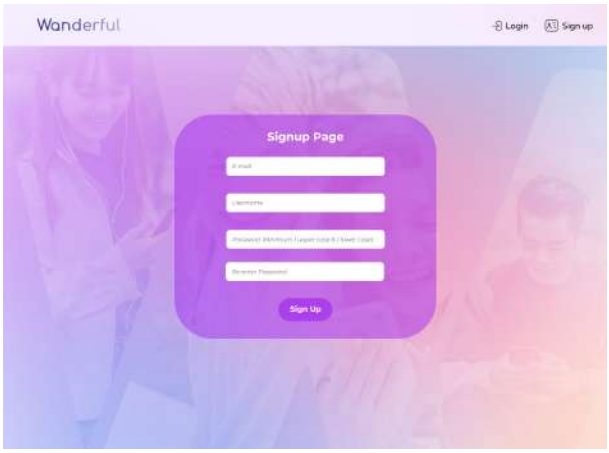


Fig 1.2: *Signup Page*

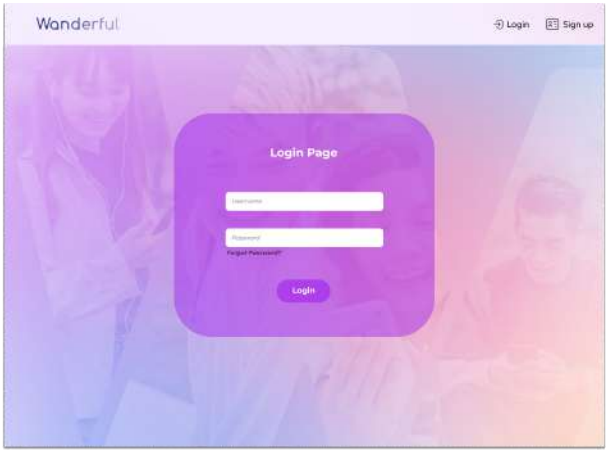


Fig 1.3: *Login Page*

The figure above shows the user interface for MainMenu, Log in and Registration. The design of the screen is kept simple and only essential information is displayed on the screen. This will enable the user to have a seamless onboarding experience with Wanderful. The use of a smiling human model using her phone is to show how easy and convenient Wanderful is.

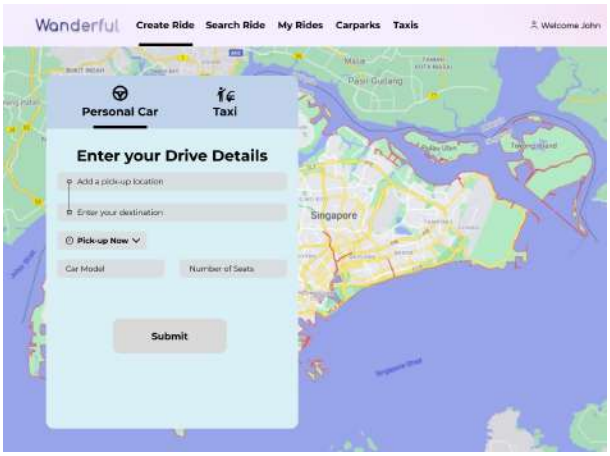


Fig 1.4: Create *Personal Car Ride Page*

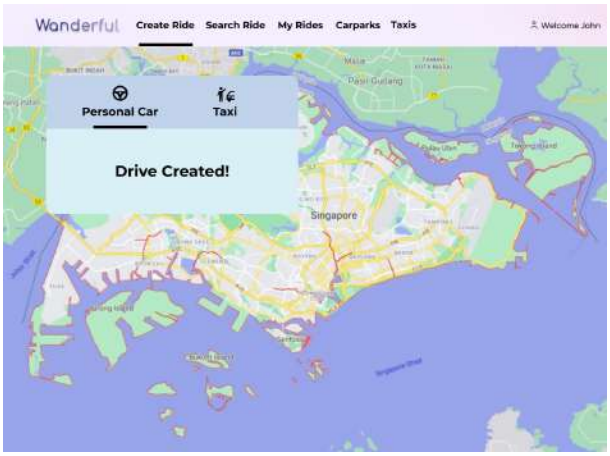


Fig 1.5: *Personal Car Ride Confirm Page*

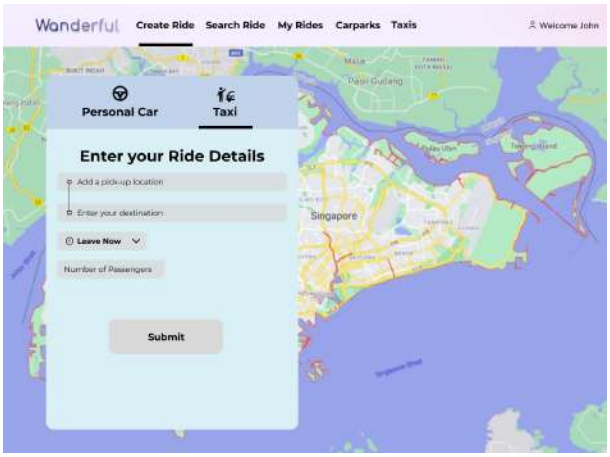


Fig 1.6: Create *Taxi Ride Page*

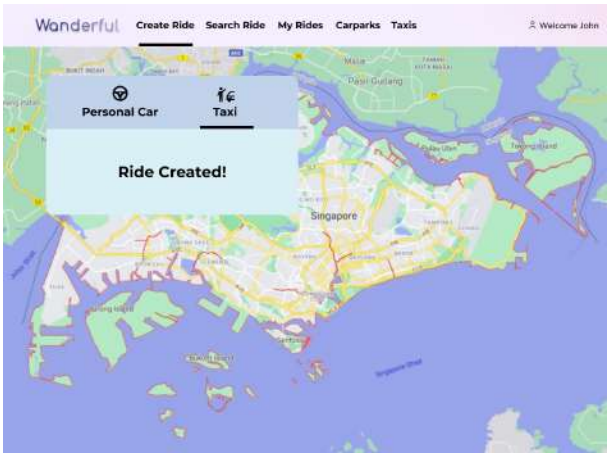


Fig 1.7: *Taxi Ride Confirm Page*

The Create Rides Page UI contains 2 different modes. Personal Car mode (for private hire drivers) and Taxi (for taxi drivers). The user interface on these screens has the Google Maps API as the background of the web pages to function as a visual aid for users to see when the navigation markers land on the locations that they have entered.

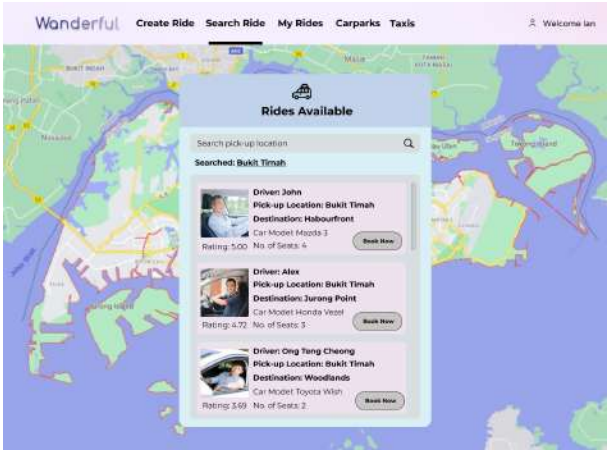


Fig 1.8: Search *Ride Page*

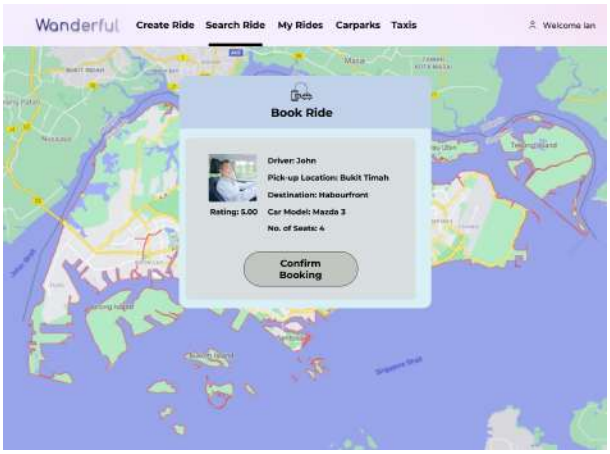


Fig 1.9: *Book Ride Page*

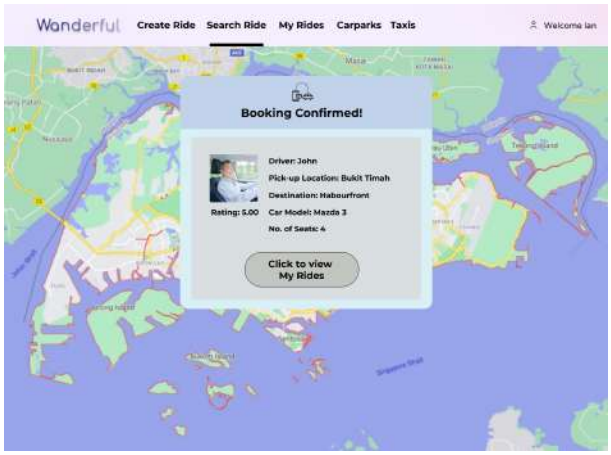


Fig 1.10: *Booking Confirmed Page*

In the Search Ride Page UI, users can search for rides available in Wanderful by entering the pick-up location that they would like to be picked up at. Users can view the details of the carpool rides available and request to book the rides. These ride requests will be sent to the driver who created the ride.

Fig 1.11: My Rides *Page*

Fig 1.12: *Ride Details Page*

Fig 1.13: Carpool *Group Chat Page*

In My Rides UI, users can view their upcoming trips and past trips that they have booked or created before. They can see the trip details and click on the 'View Details' button to view the names of the other carpool passengers. Users can chat in the group chat created by this ride entity where the driver and passengers involved can message each other.

Fig 1.14: *Ride Requests Page*

The Ride Requests UI is where users can view their Accepted Trips (Past Trips) and Pending Trips (Upcoming Trips). The user can choose to accept a ride request or reject a ride request.

Fig 1.15: *FAQ Page*

The Frequently Asked Questions (FAQ) UI shows a list of frequently asked questions that have been cumulated from existing Wanderful users.

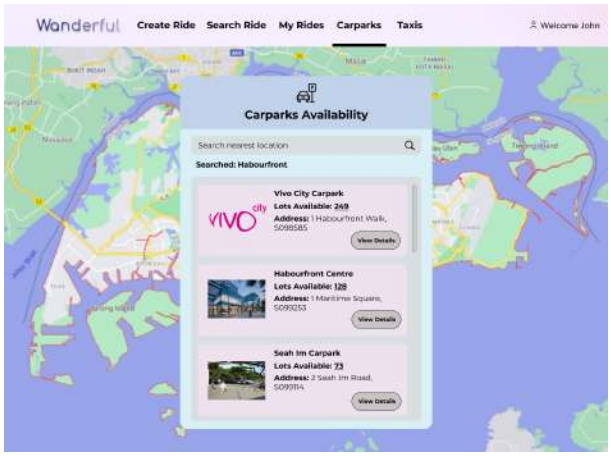


Fig 1.16: *Carpark Availability Page*

The Carparks Availability Page shows the list of car parks that are based on the input location. User will enter the location that they are looking for parking at and click on the search icon. The list will show the relevant car parks near the location indicated and the details of the parking lot such as Name of the Parking Lot, Lots Available at the Parking Lot, Address of the Parking Lot.

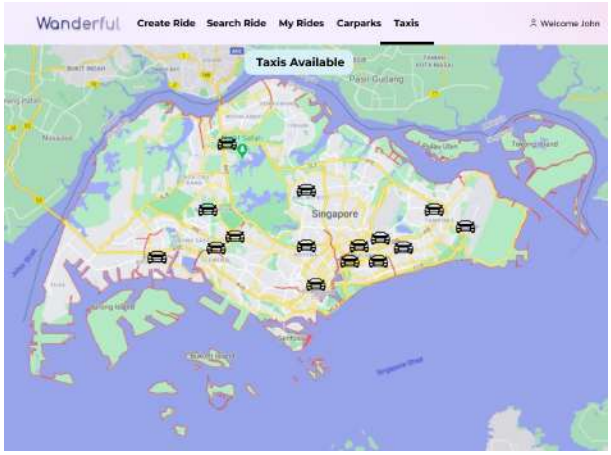


Fig 1.17: *Taxi Availbility Page*

In the Taxi Availability UI, users can view the taxis that are nearby the location that they have searched for. The user will input the pick-up location that they want to be picked up at, and the UI will depict the taxis that are near the user's input location.

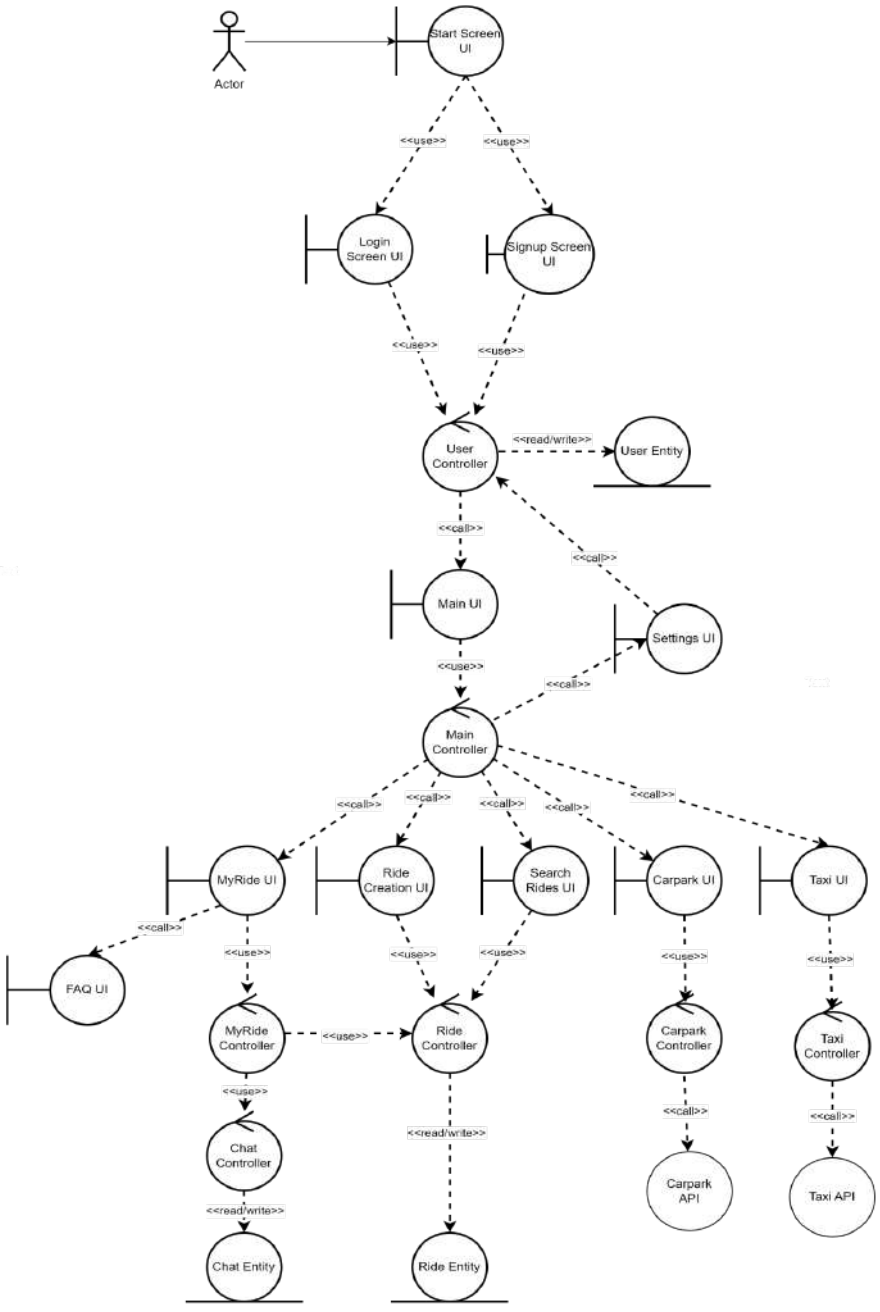
Fig 1.18: *User Profile Settings Page*

In the User Profile Settings UI, user is able to view and edit their profile details. User can click on the ‘Edit Profile’ button which allows the user to edit fields such as ‘Date of Birth’, ‘Home Address’ and ‘Biography’. The logout feature can also be found in this UI.

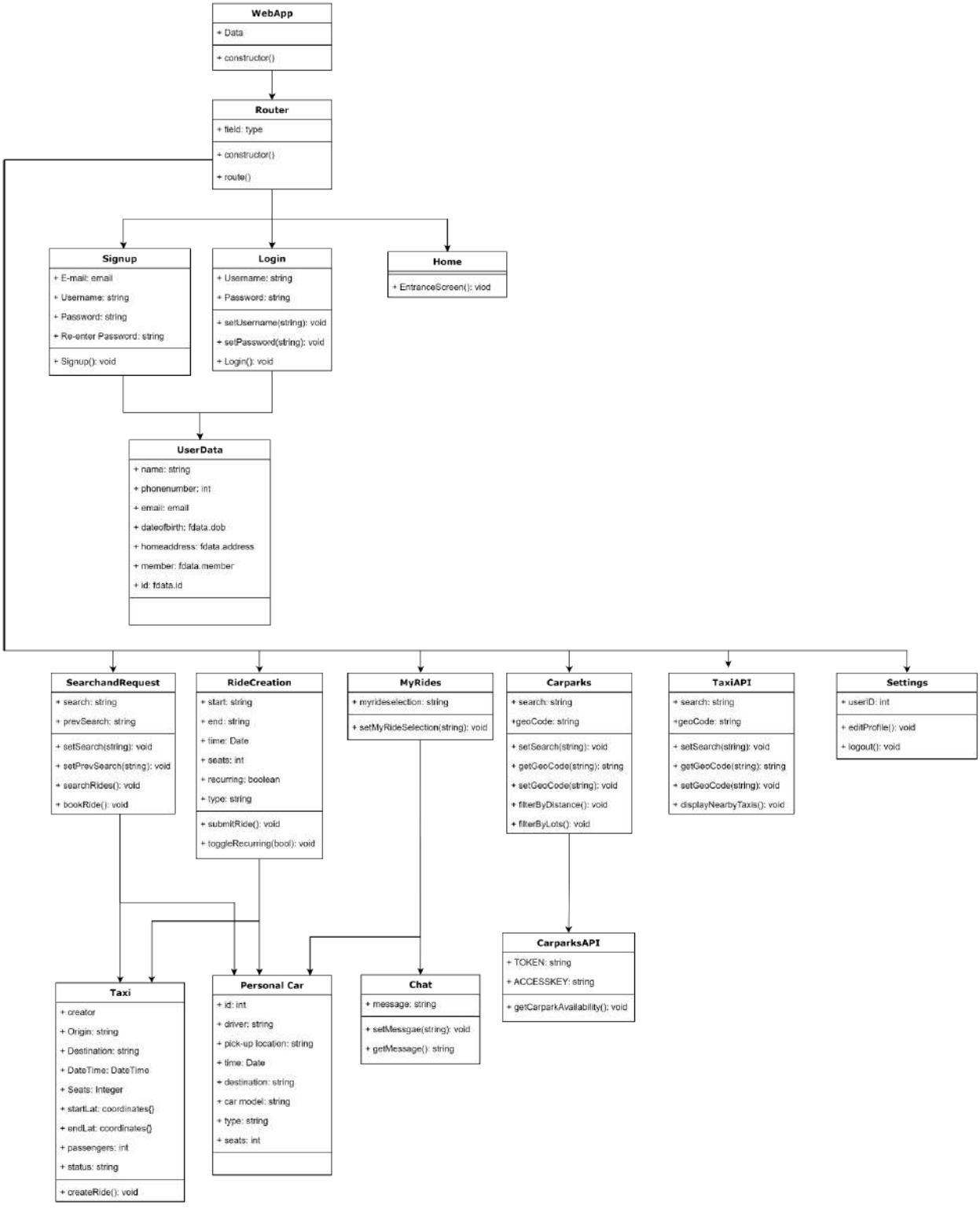
2. Design

2.1 Class Diagrams

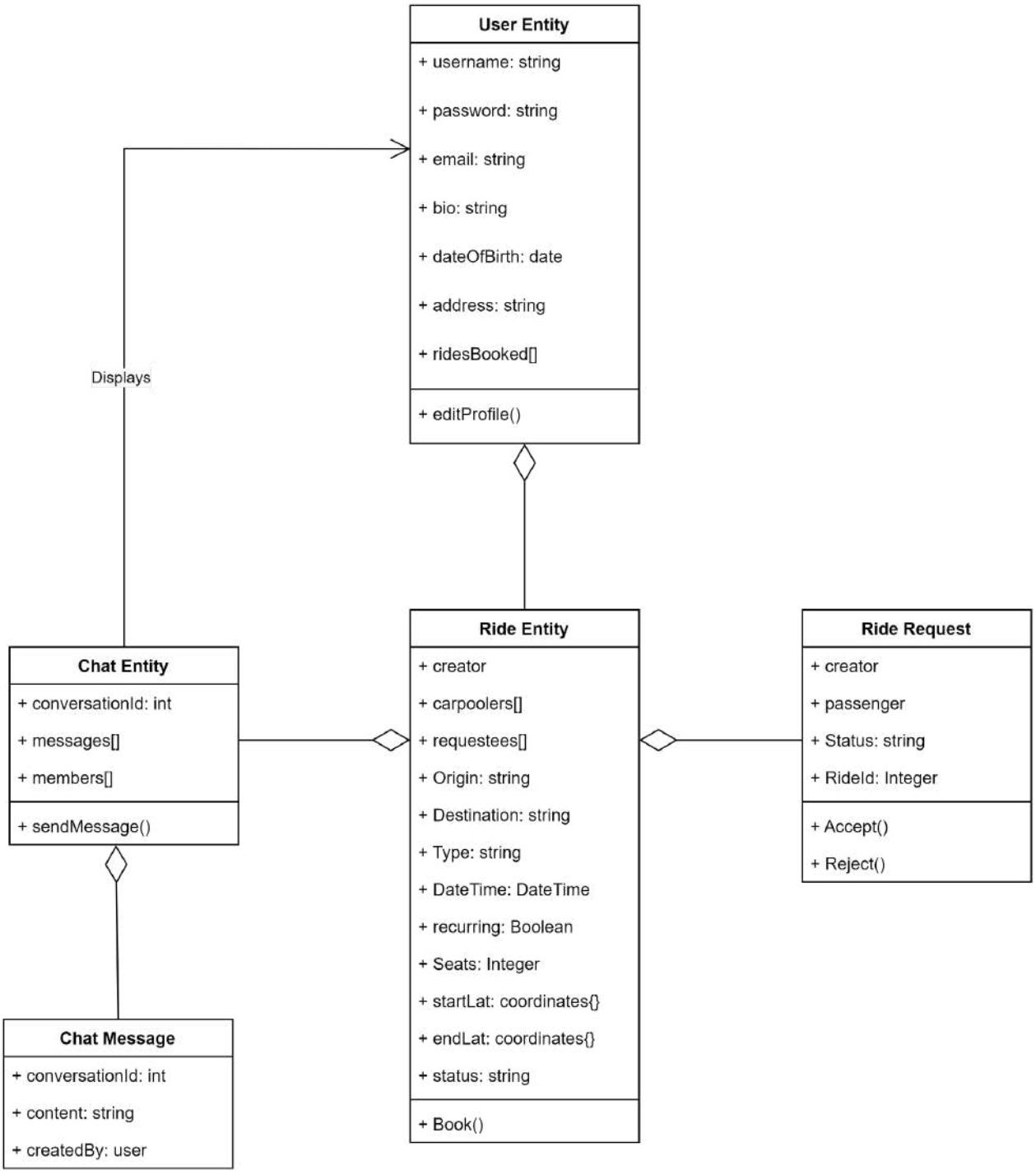
2.1.1 Class Diagram



2.1.2 Design Class Diagram

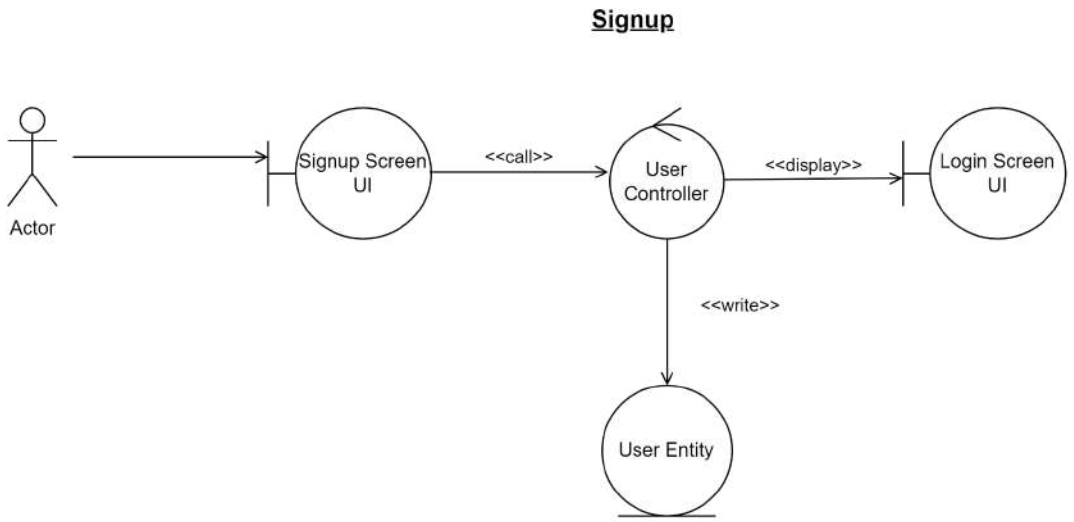


2.1.3 Entity Class Diagram

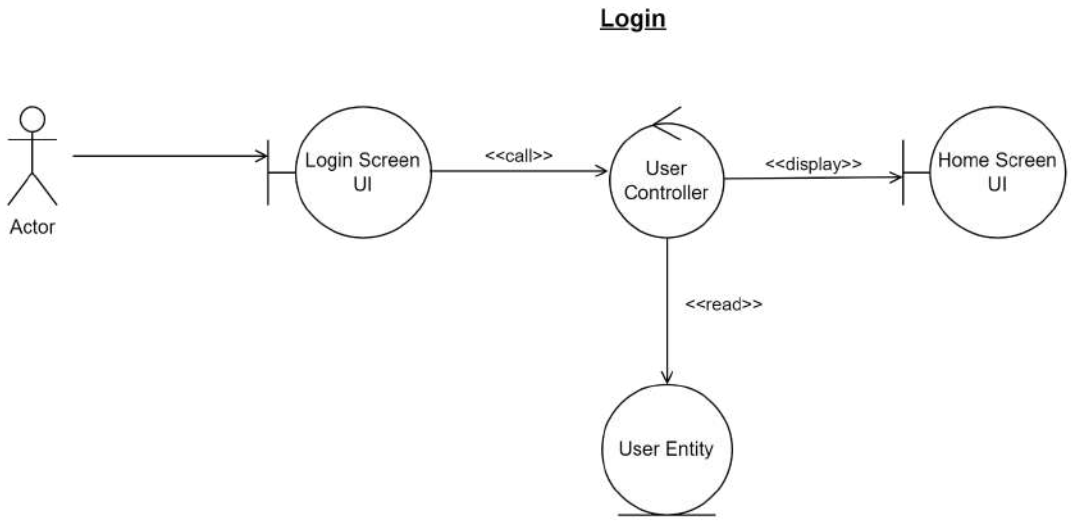


2.2 Conceptual Models

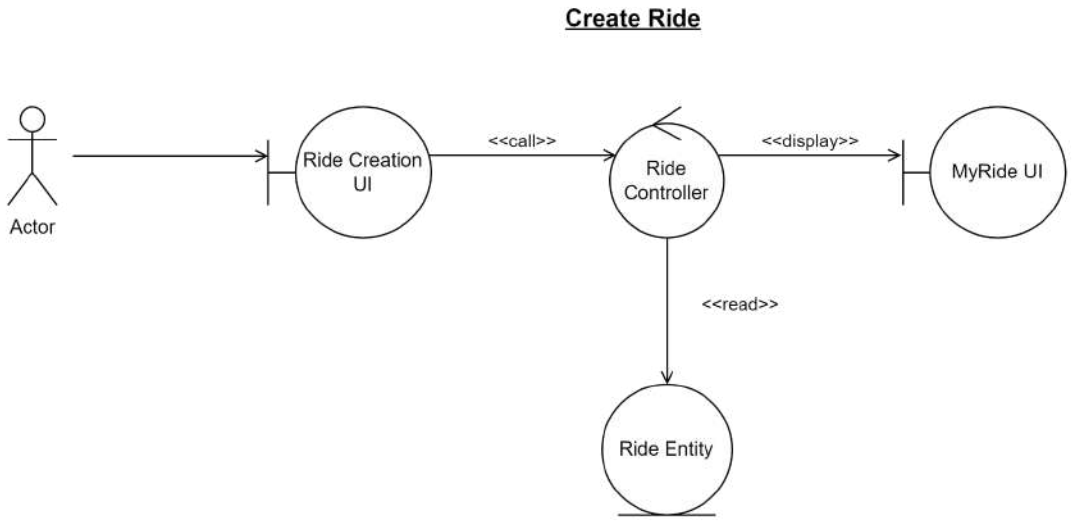
2.2.1 Signup



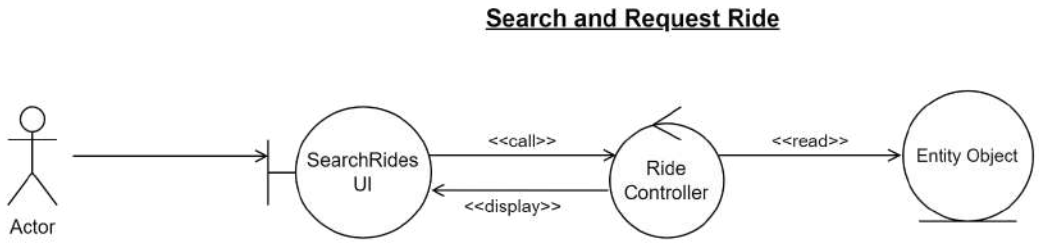
2.2.2 Login


Login
Actor
Login Screen UI
<<call>>
User Controller
<<display>>
Home Screen UI
<<read>>
User Entity

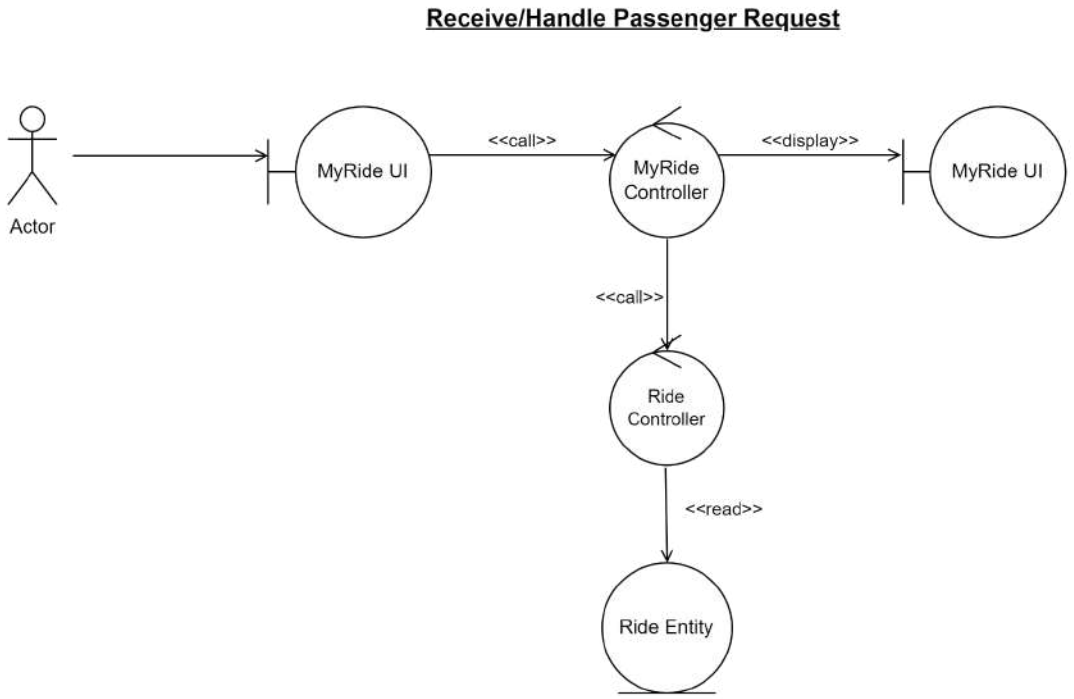
2.2.3 Create Ride



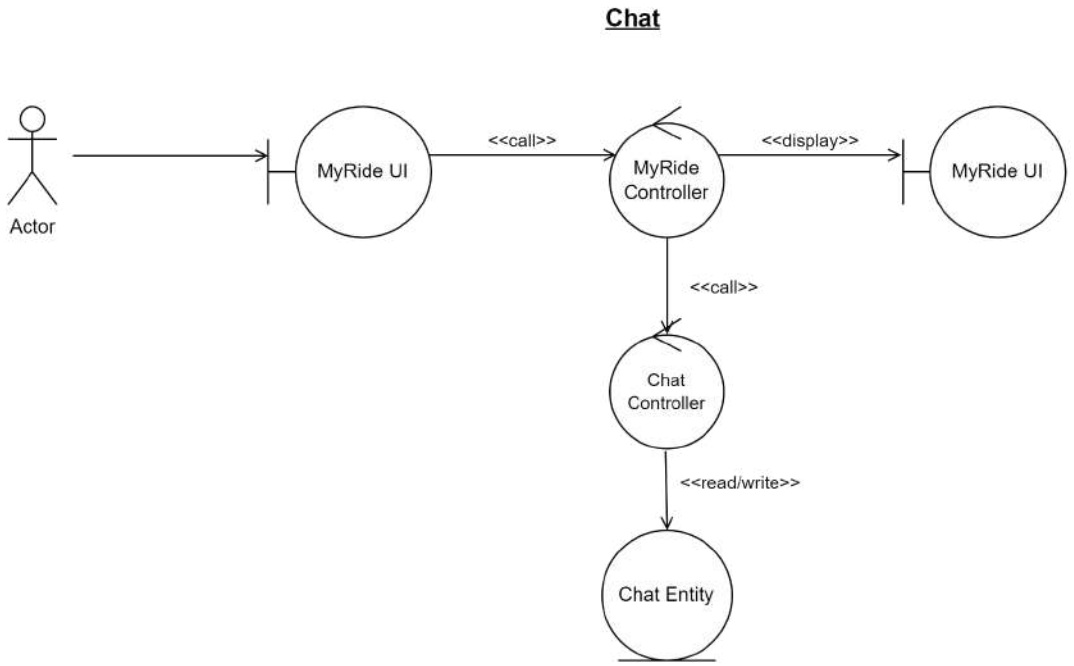
2.2.4 Search and Request Ride



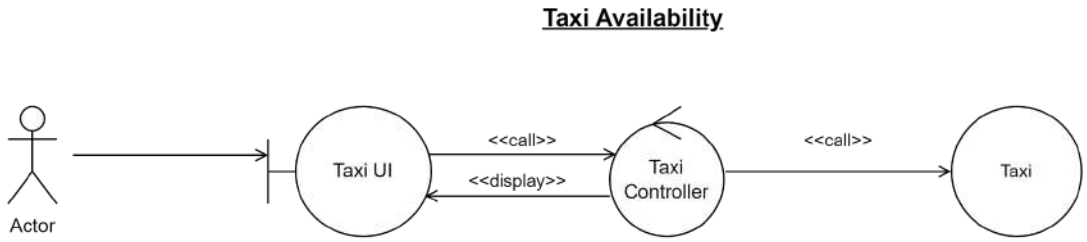
2.2.5 Recieve/Handle Ride



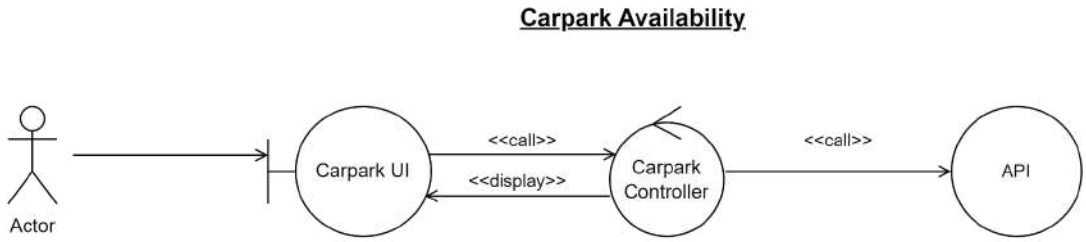
2.2.6 Chat


Chat
Actor
MyRide UI
<<call>>
MyRide Controller
<<display>>
MyRide UI
<<call>>
Chat Controller
<<read/write>>
Chat Entity

2.2.7 Taxi Availability

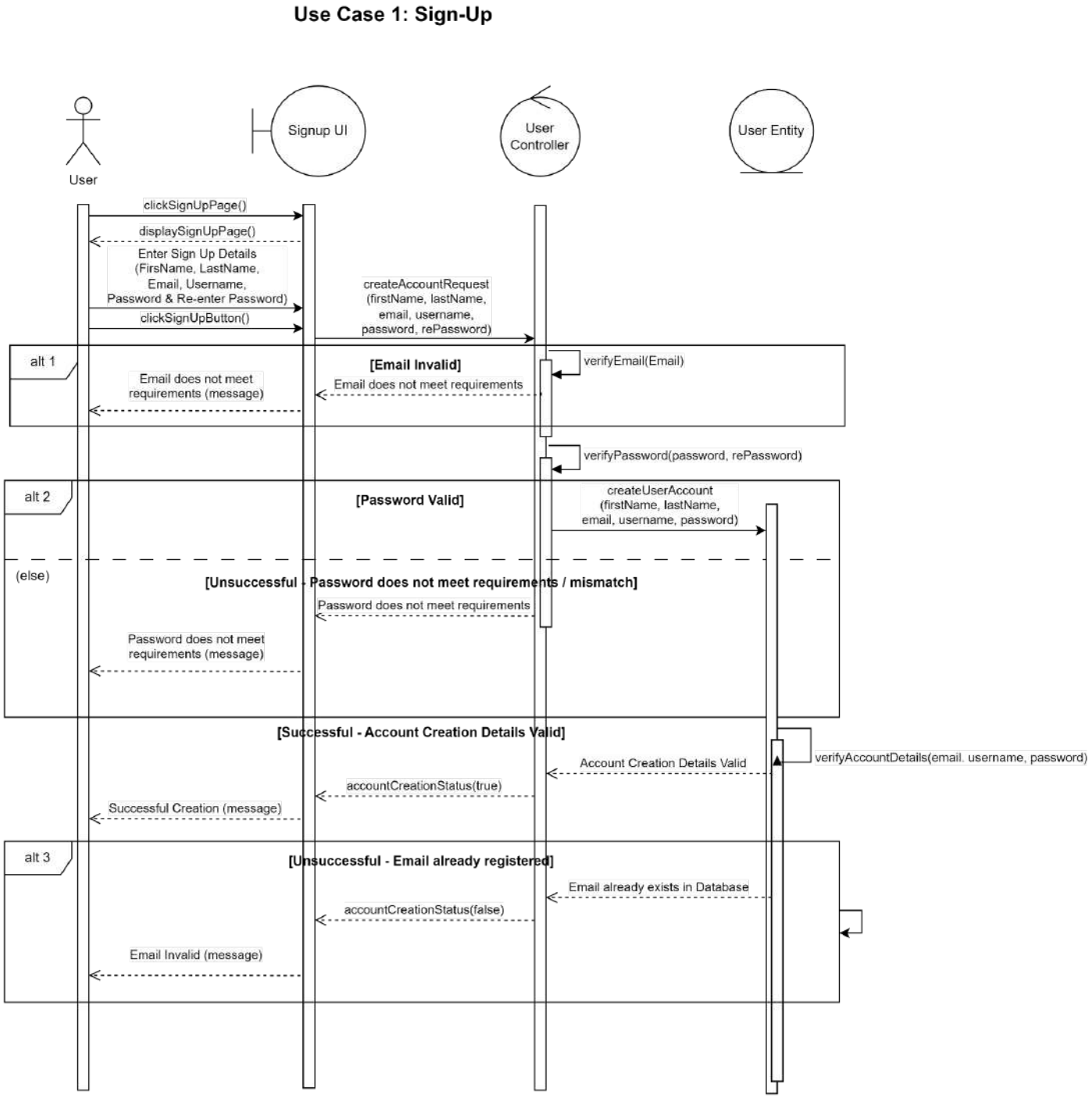


2.2.8 Carpark Availablity

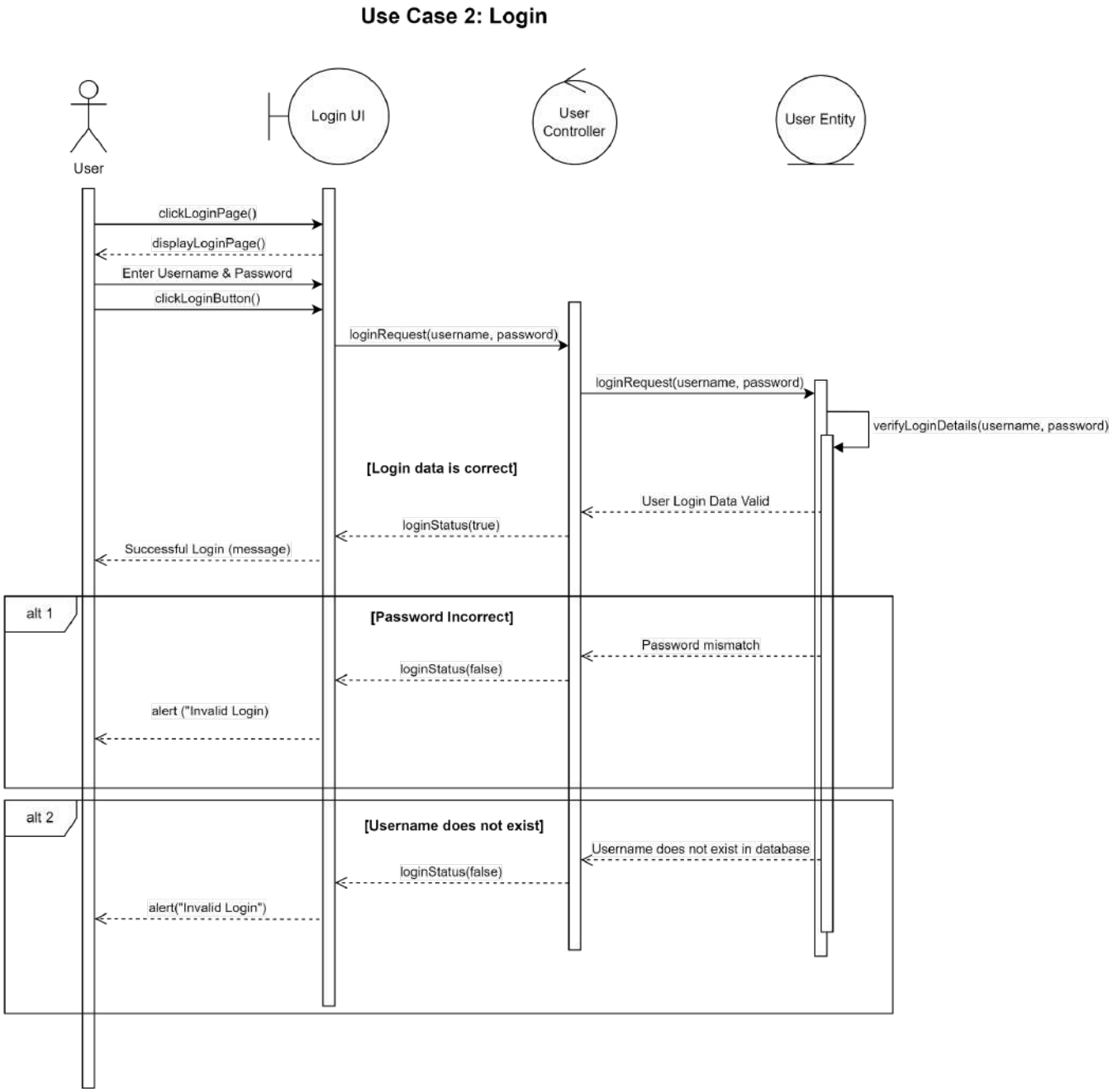


2.3 Sequence Diagrams

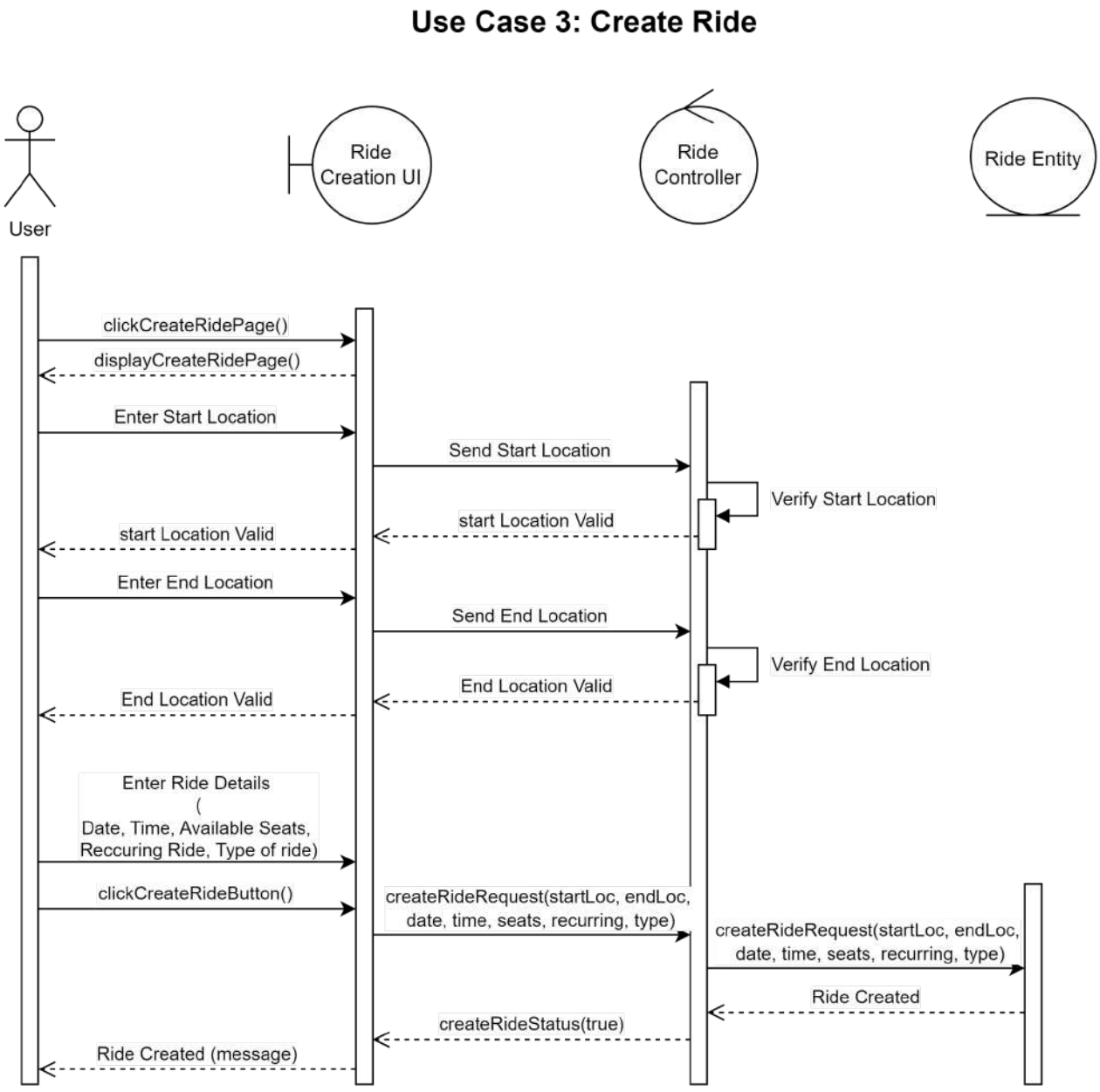
2.3.1 Signup



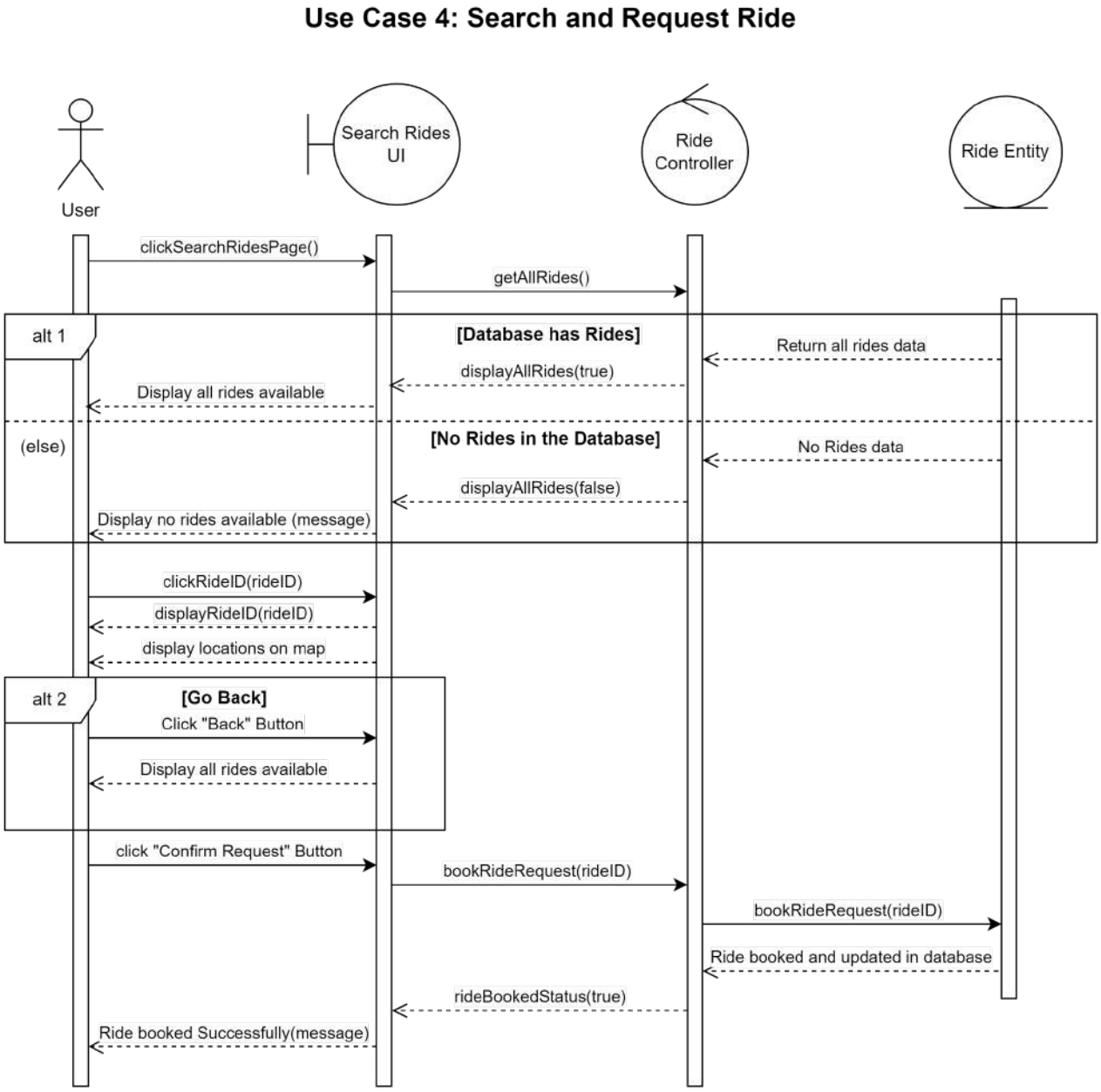
2.3.2 Login


Use Case 2: Login
User
Login UI
User Controller
User Entity
clickLoginPage()
displayLoginPage()
Enter Username & Password
clickLoginButton()
loginRequest(username, password)
loginRequest(username, password)
verifyLoginDetails(username, password)
[Login data is correct]
User Login Data Valid
loginStatus(true)
Successful Login (message)
alt 1
[Password Incorrect]
Password mismatch
loginStatus(false)
alert ("Invalid Login)
alt 2
[Username does not exist]
Username does not exist in database
loginStatus(false)
alert("Invalid Login")

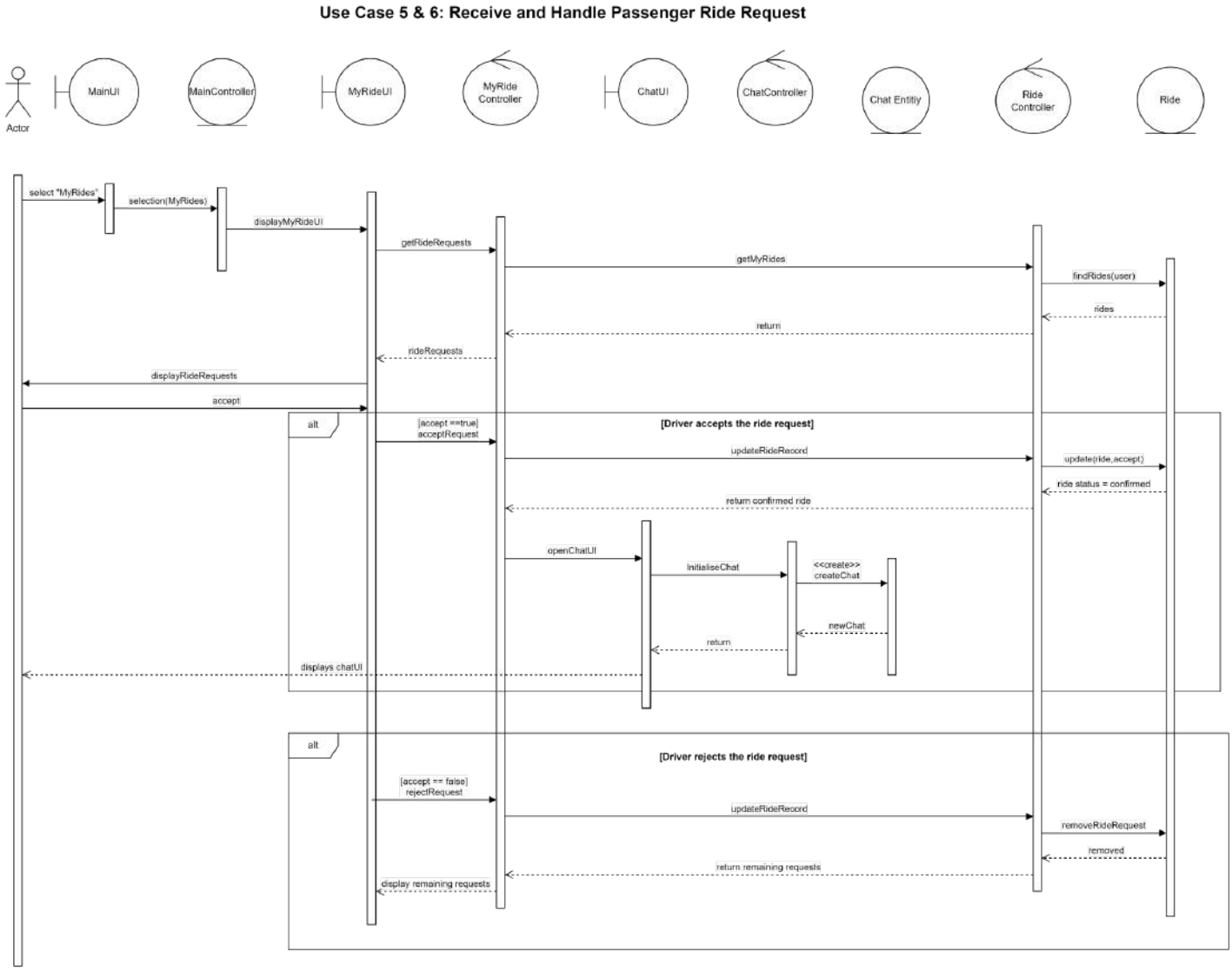
2.3.3 Create Ride



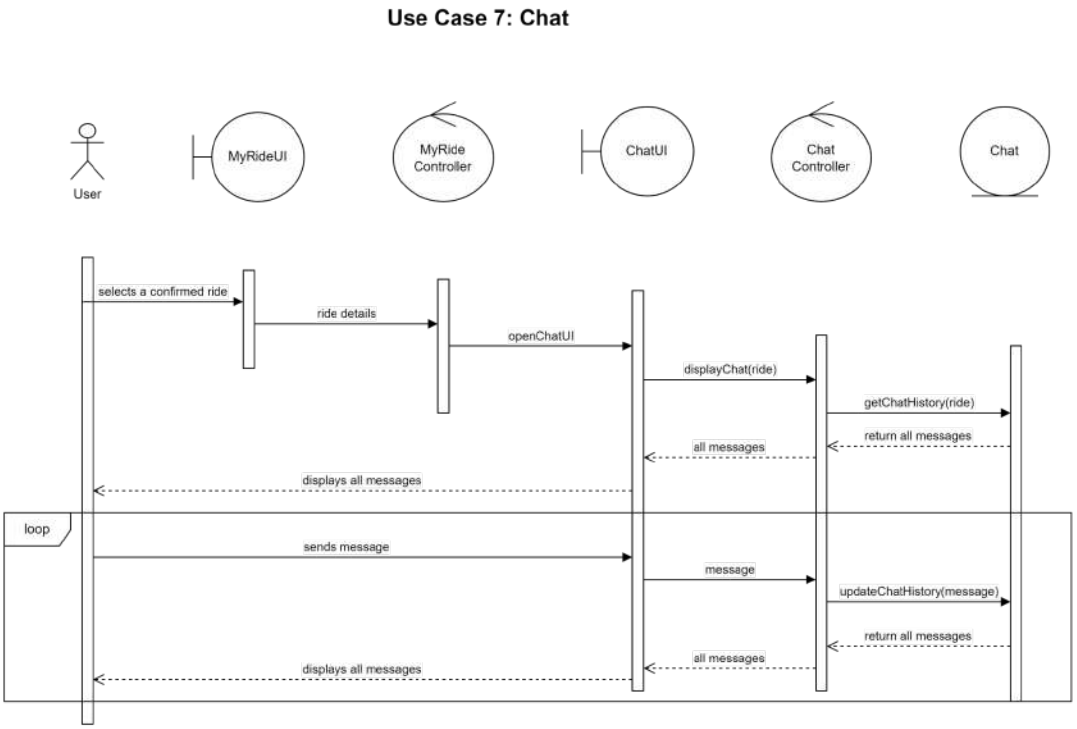
2.3.4 Search And Request Ride



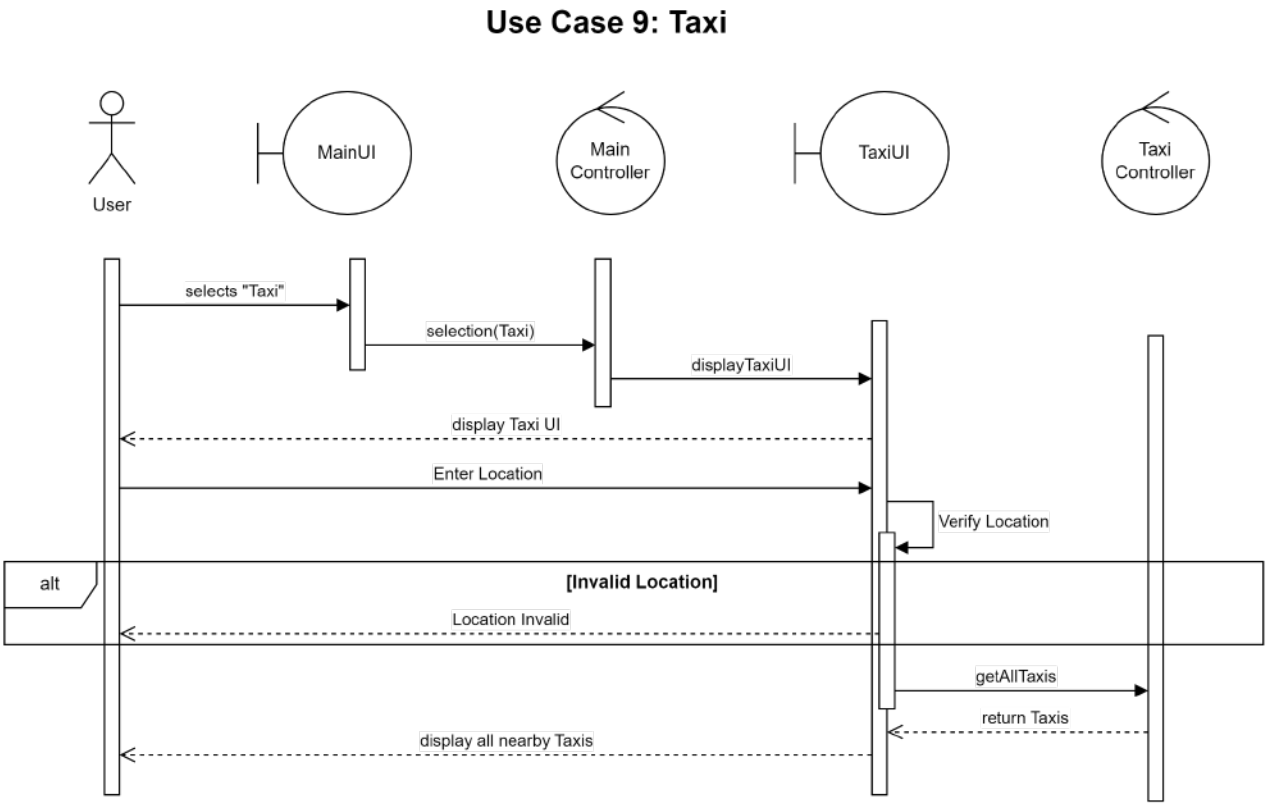
2.3.5 Receive and Handle Passenger Ride Request



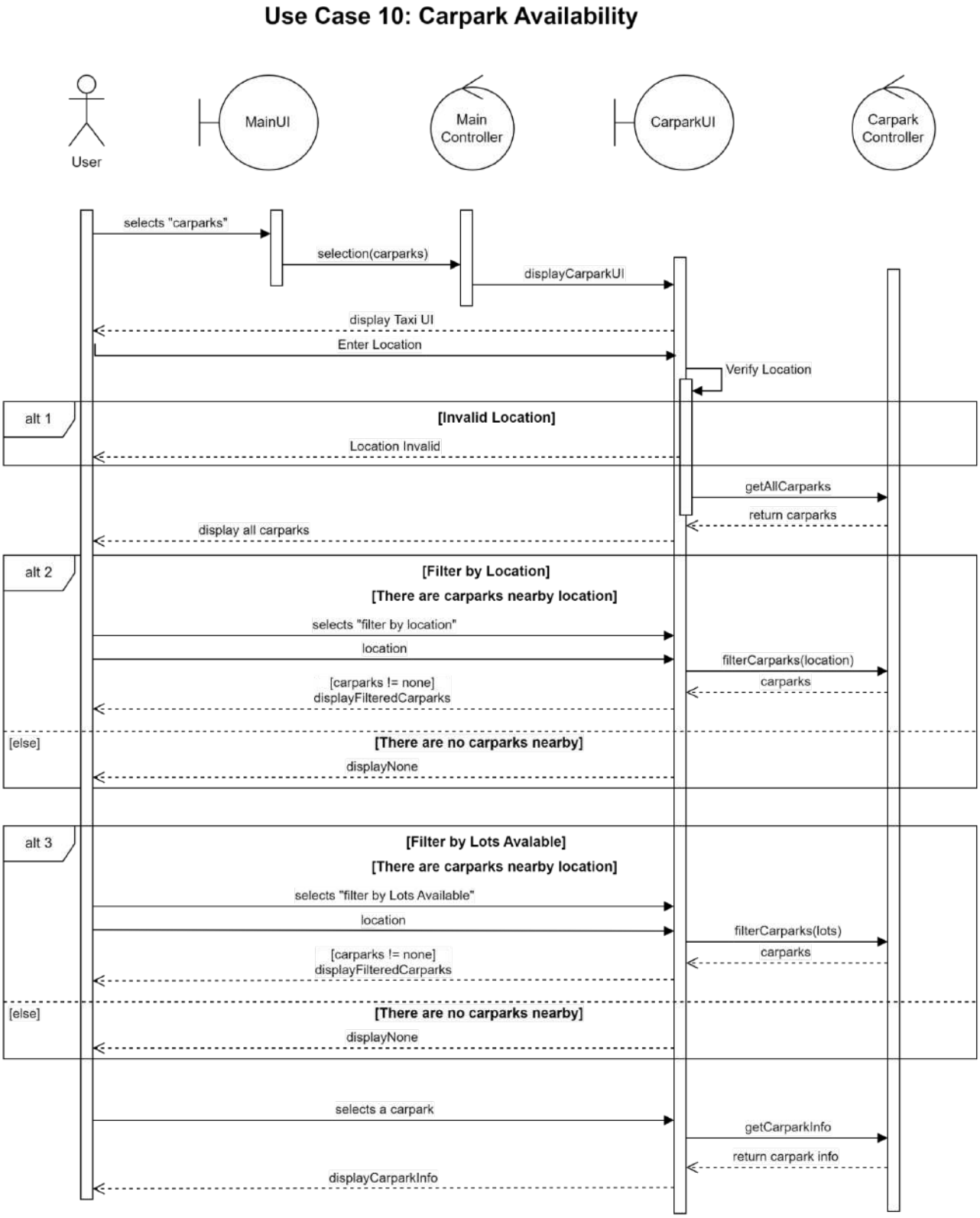
2.3.6 Chat


Use Case 7: Chat
User
MyRideUI
MyRide Controller
ChatUI
Chat Controller
Chat
selects a confirmed ride
ride details
openChatUI
displayChat(ride)
getChatHistory(ride)
return all messages
all messages
displays all messages
loop
sends message
message
updateChatHistory(message)
return all messages
all messages
displays all messages

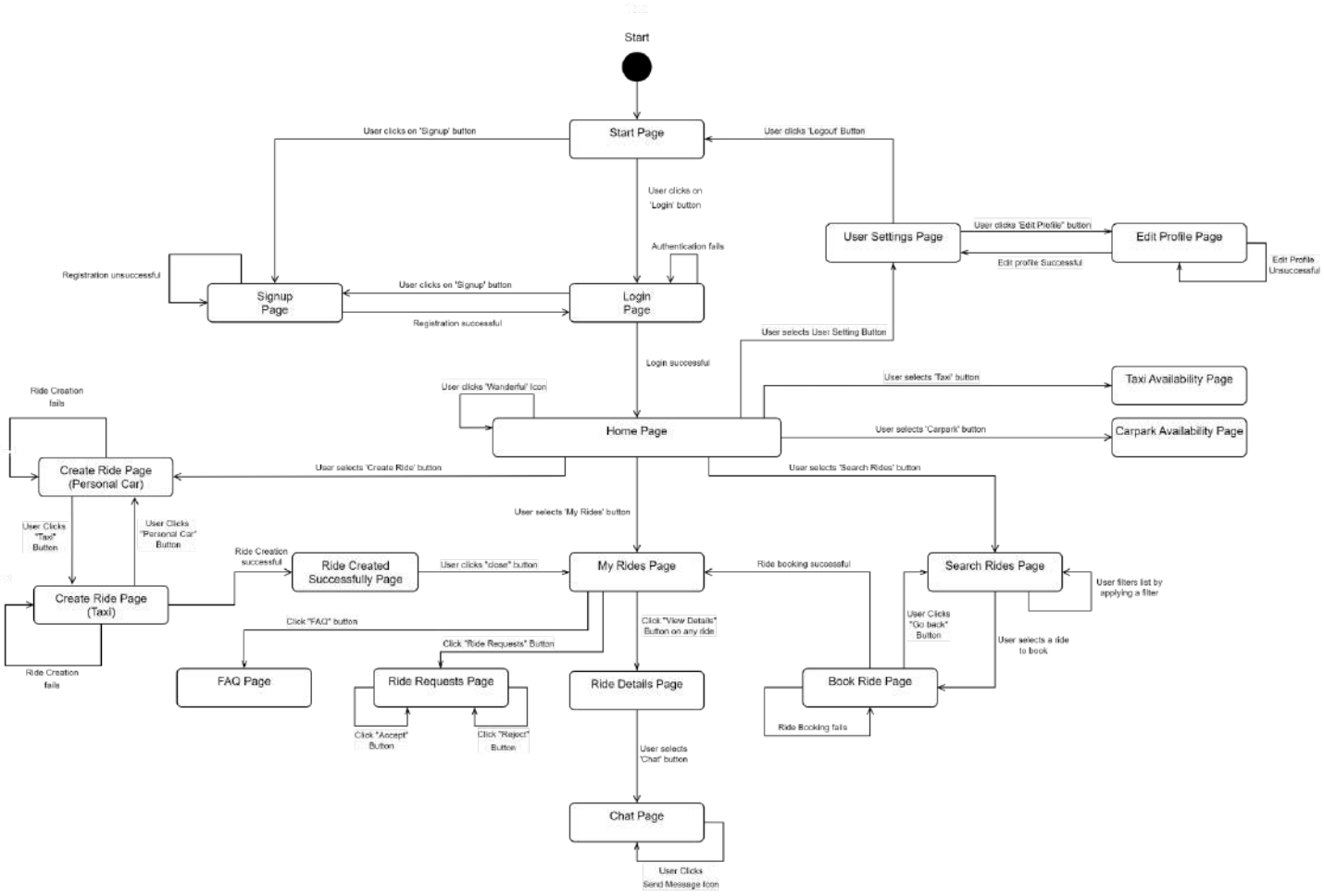
2.3.7 Taxi


Use Case 9: Taxi
User
MainUI
Main Controller
TaxiUI
Taxi Controller
selects "Taxi"
selection(Taxi)
displayTaxiUI
display Taxi UI
Enter Location
Verify Location
alt
[Invalid Location]
Location Invalid
getAllTaxis
return Taxis
display all nearby Taxis

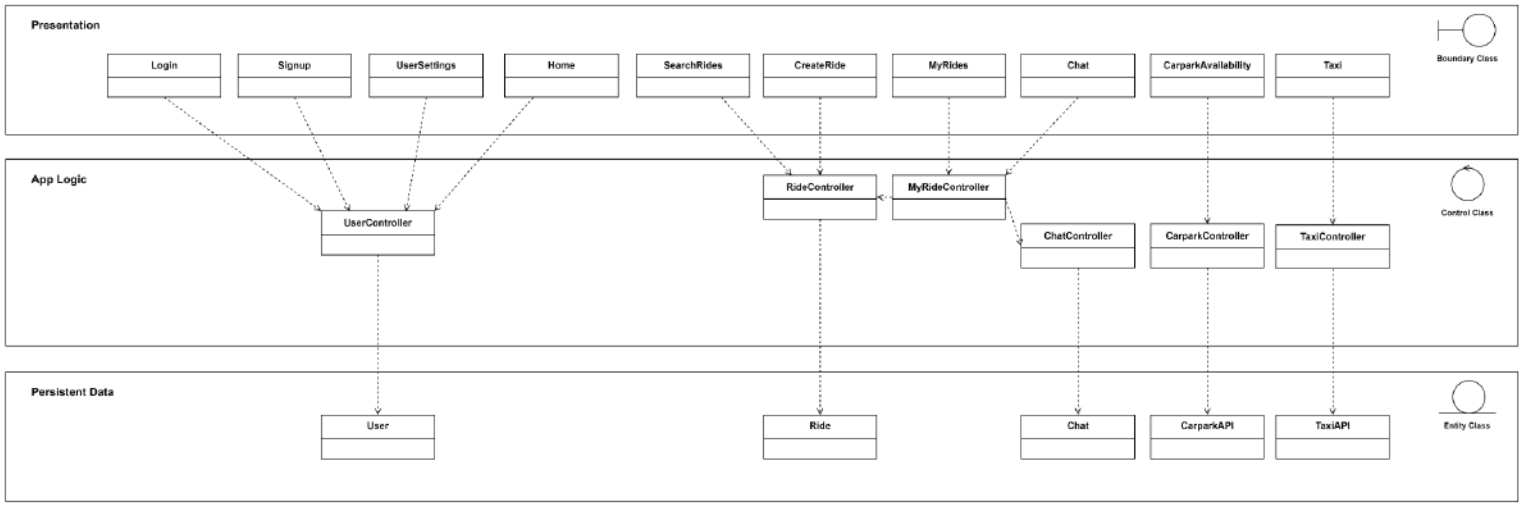
2.3.8 Carpark Availablity



2.4 Dialog Map



2.5 System Architecture Diagram



3. Implementation

3.1 Application Skeleton

3.1.1 Boundary Classes

3.1.1.1 Login

```
return (
  <div>
      {user.token && <Navigate to="/"/>}
      <div className='bodymain'>
        <div className='interface_login'>
          <form className="login__form" onSubmit={handleSubmit}>
            <div style={{height:'100%',width:'100%',display:'flex',justifyContent:'center',alignItems:'center',flexDirection:'column'}}>
              <h2 style={{color:"white", display:"flex",justifyContent:"center"}}>Login Page</h2>
              <input type="text" placeholder='Username' className="login-input" onChange={e => setUsername(e.target.value)} />
              <input type="password" placeholder='Password' className="login-input" onChange={e => setPassword(e.target.value)} />
              <input type="submit" className='login-button' value="Login"/>
            </div>
          </form>


        </div>
      </div>
    </div>
  </div>
)
```

This function displays a loginUI. The login screen accepts a username and password which will be passed to (XXX) to authenticate. Once authenticated, the function would load in the user's settings from the database and grant the user access to the application.

3.1.1.2 Signup

This function displays a signupUI. The signup screen requires the user to input a username, password and email, creates a new entry in the database.

3.1.1.3 Main Menu

```
return(
    <div style={{backgroundColor:"green"}}>
    <div className="page">
        <div className="form">
            {selection==="Create Ride" && <RideCreation parentCallBack = {handleSelection}/>}
            {selection==="Search Ride" && <SearchRide parentCallBack = {handleSelection}/>}
            {selection==="Carparks" && <Carpark/>}
            {selection==="Taxi" && <Taxi/>}
        </div>
        {!user && <Navigate to="/"/>}
    </div>
        {selection==="My Rides" && <MyRides/>}
        {selection!=='My Rides' &&
    <div className="map">
        <DynamicMap  className="map"/>
     </div>
        }

    </div>

)
```

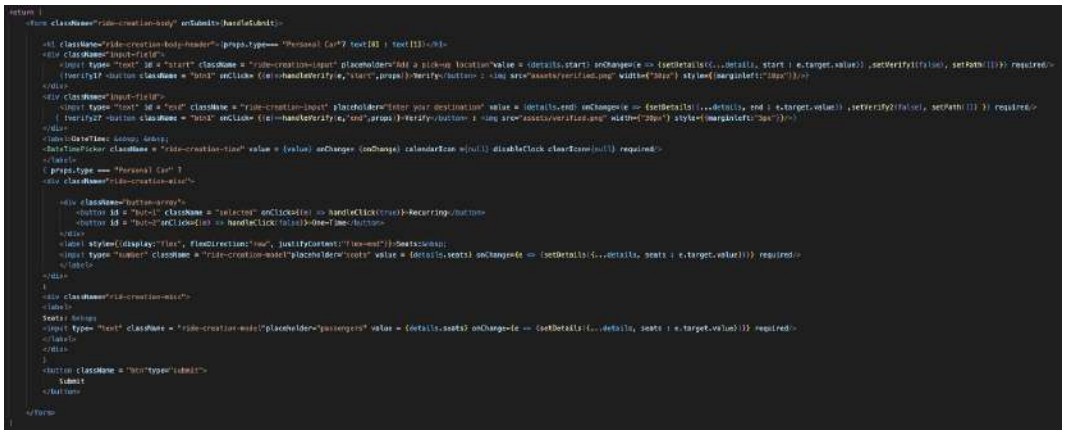
This function displays the main UI screen. The main UI screen has 4 navigation to “Create Ride” “Search Ride” “My Rides” “Carpark” along with a dynamic google map centered on Singapore. Selecting any would navigate to the respective page.

3.1.1.4 CreateRide - Main

```
return (

    <div className="form-group" >

        <div className = "ride-creation-header">
            <button type ="button"className={type === "Personal Car" ? "ride-header-button-selected" :"ride-header-button"  }onClick = {() => setType("Personal Car")} >
                <img className = "img" src='assets/steeringWheel.png'/>
                <h1 className="ride-header-button-text">Personal Car</h1>
            </button>
            <button type ="button" className={type === "Taxi" ? "ride-header-button-selected" :"ride-header-button"  }onClick = {() => setType("Taxi")}>
                <img className = "img" src='assets/rideHailing.png'/>
                <h1 className="ride-header-button-text">Taxi</h1>
            </button>
        </div>
        {step === 0 ? <CreateRide type = {type} parentCallBack = {handleCallback}/>   : <DisplayComplete parentCallBack = {handleComplete}/> }
    </div>
);
```

This function displays the RideCreationUI, which allows the user to toggle between selecting to create a personal car ride or a taxi ride, by calling **(3.1.1.5) CreateRide - Personal Car or Taxi** with the type “Personal Car” or “Taxi” as parameters. It also accepts a callback function from **(3.1.1.5) CreateRide - Personal Car or Taxi** that passes back the input details of the ride. Once **(3.1.1.5) CreateRide - Personal Car or Taxi** is complete it will call **(3.1.1.6) CreateRide - DisplayComplete.**

3.1.1.5 CreateRide - Personal Car or Taxi



This function displays the RideCreationUI, depending on whether “Personal Car” or “Taxi” has been selected. This function allows the user to fill in the relevant ride information depending on whether “Personal Car” or “Taxi” has been selected. The input information will be handled by the RideController.

3.1.1.6 CreateRide - DisplayComplete

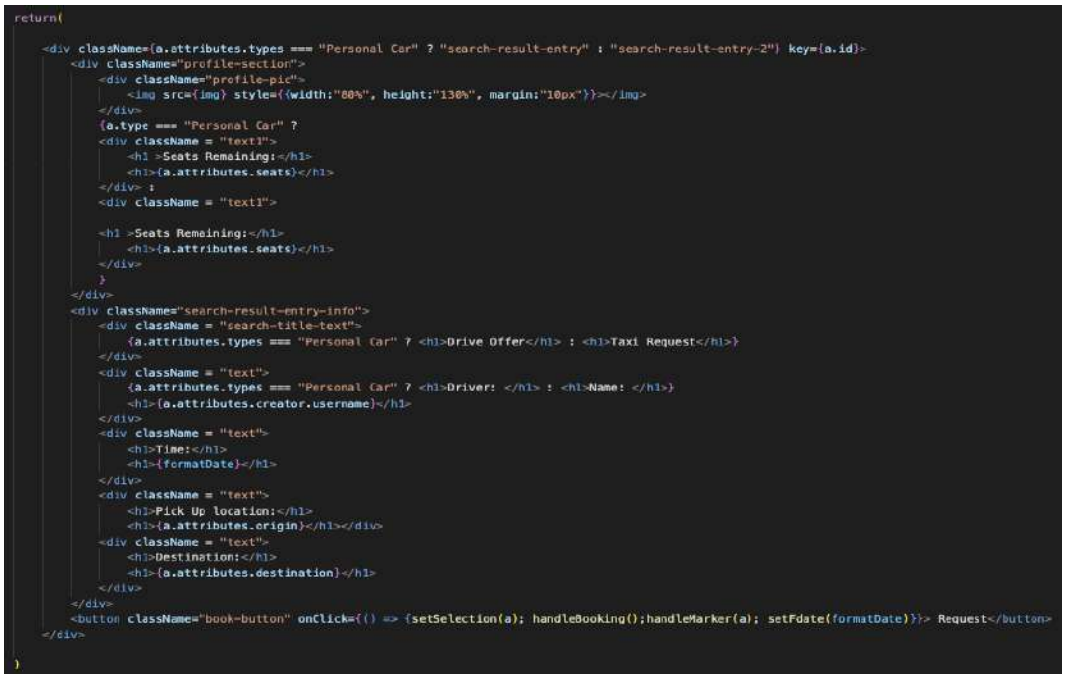
This function displays the RideCreationUI, to show that Ride has been successfully created. This function will show when the *user* has filled in all relevant ride creation information and validated properly with the RideController.

3.1.1.7 Search Ride - Main

```
return(
<div className="search-ride">
    <div className="search-ride-header">

        <img src ="assets/carRideIcon.png" width="30px" style={{marginTop:"10px"}}></img>
        <h1 className="search-ride-headerText">Rides Available</h1>
    </div>
    <div className = "search-ride-upper">
        <div className="search-ride-searchbar">
            <textarea rows={1}placeholder ="Search pick-up location" value ={search} onChange={e =>setSearch(e.target.value)} >

            </textarea>
            <div className="search-ride-searchIcon">
                <img src = "assets/searchIcon.png" width="15px" style={{marginTop:"40%"}} onClick ={handleSearch}></img>

            </div>
        </div>
        <div className="search-ride-result">
            <h4 >Searched: </h4><h4 style= {{borderBottom : "solid"}}>{prevSearch}</h4>
        </div>
    </div>
    <div className="search-ride-body">
       {displayResults}
    </div>

</div>
```

This function displays the SearchRideUI, which allows users to input a destination location. It calls the RidesController to return all rides that have similar destinations to the input from the database.

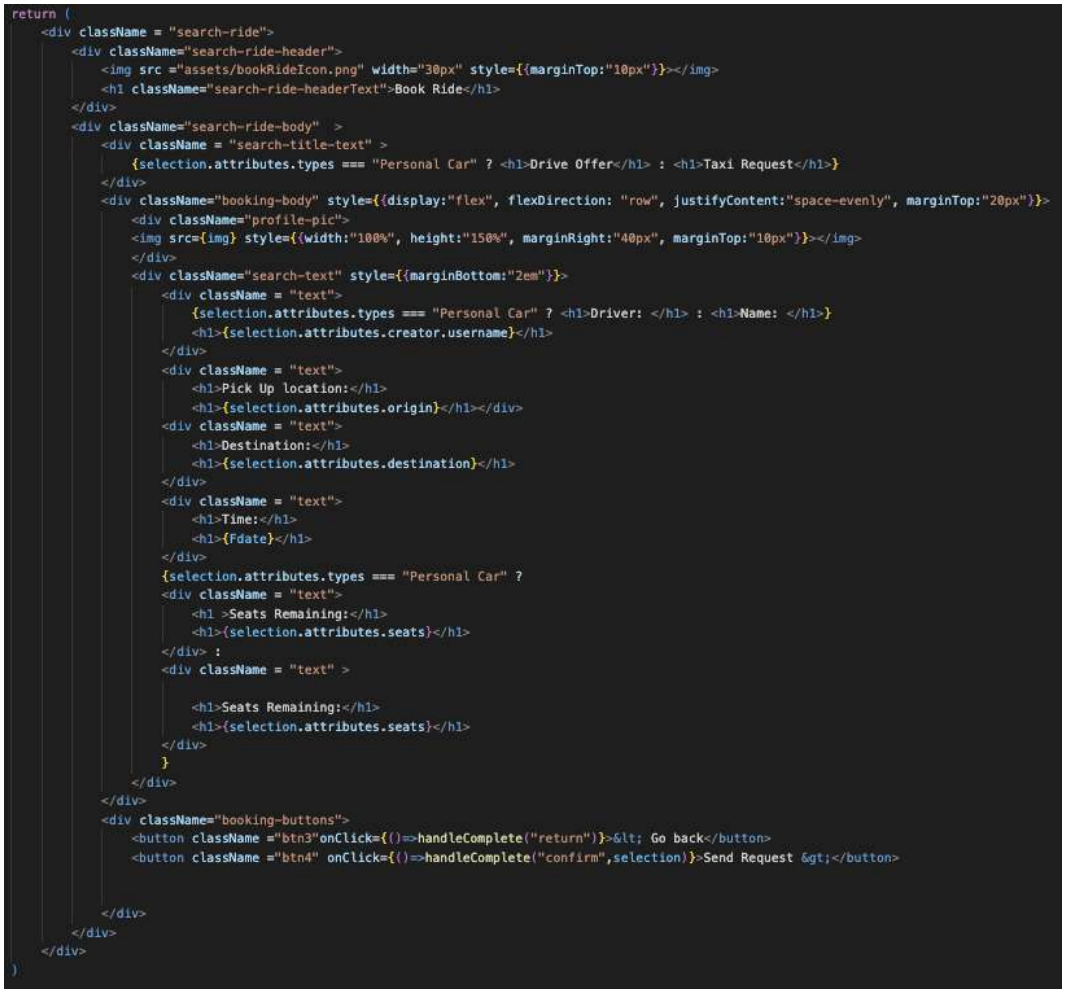
3.1.1.8 Search Rides - Display Rides



```
return(

    <div className={a.attributes.types === "Personal Car" ? "search-result-entry" : "search-result-entry-2"} key={a.id}>
        <div className="profile-section">
            <div className="profile-pic">
                <img src={img} style={{width:"80%", height:"130%", margin:"10px"}}></img>
            </div>
            {a.type === "Personal Car" ?
            <div className = "text1">
                <h1 >Seats Remaining:</h1>
                <h1>{a.attributes.seats}</h1>
            </div> :
            <div className = "text1">

            <h1 >Seats Remaining:</h1>
                <h1>{a.attributes.seats}</h1>
            </div>
            }
        </div>
        <div className="search-result-entry-info">
            <div className = "search-title-text">
                {a.attributes.types === "Personal Car" ? <h1>Drive Offer</h1> : <h1>Taxi Request</h1>}
            </div>
            <div className = "text">
                {a.attributes.types === "Personal Car" ? <h1>Driver: </h1> : <h1>Name: </h1>}
                <h1>{a.attributes.creator.username}</h1>
            </div>
            <div className = "text">
                <h1>Time:</h1>
                <h1>{formatDate}</h1>
            </div>
            <div className = "text">
                <h1>Pick Up location:</h1>
                <h1>{a.attributes.origin}</h1></div>
            <div className = "text">
                <h1>Destination:</h1>
                <h1>{a.attributes.destination}</h1>
            </div>
        </div>
        <button className="book-button" onClick={() => {setSelection(a); handleBooking();handleMarker(a); setFdate(formatDate)}}> Request</button>
    </div>

)
```

This function displays the SearchRideUI, after the *user* has input a destination location. It formats the data called by the RideController which fetches the rides from the database and display the return results to the *user*.

3.1.1.9 Search Rides - Request Ride



```
return (
    <div className = "search-ride">
        <div className="search-ride-header">
            <img src ="assets/bookRideIcon.png" width="30px" style={{marginTop:"10px"}}></img>
            <h1 className="search-ride-headerText">Book Ride</h1>
        </div>
        <div className="search-ride-body"  >
            <div className = "search-title-text" >
                {selection.attributes.types === "Personal Car" ? <h1>Drive Offer</h1> : <h1>Taxi Request</h1>}
            </div>
            <div className="booking-body" style={{display:"flex", flexDirection: "row", justifyContent:"space-evenly", marginTop:"20px"}}>
                <div className="profile-pic">
                <img src={img} style={{width:"100%", height:"150%", marginRight:"40px", marginTop:"10px"}}></img>
                </div>
                <div className="search-text" style={{marginBottom:"2em"}}>
                    <div className = "text">
                        {selection.attributes.types === "Personal Car" ? <h1>Driver: </h1> : <h1>Name: </h1>}
                        <h1>{selection.attributes.creator.username}</h1>
                    </div>
                    <div className = "text">
                        <h1>Pick Up location:</h1>
                        <h1>{selection.attributes.origin}</h1></div>
                    <div className = "text">
                        <h1>Destination:</h1>
                        <h1>{selection.attributes.destination}</h1>
                    </div>
                    <div className = "text">
                        <h1>Time:</h1>
                        <h1>{Fdate}</h1>
                    </div>
                    {selection.attributes.types === "Personal Car" ?
                    <div className = "text">
                        <h1 >Seats Remaining:</h1>
                        <h1>{selection.attributes.seats}</h1>
                    </div> :
                    <div className = "text" >

                        <h1>Seats Remaining:</h1>
                        <h1>{selection.attributes.seats}</h1>
                    </div>
                    }
                </div>
            </div>
            <div className="booking-buttons">
                <button className ="btn3"onClick={()=>handleComplete("return")}>&lt; Go back</button>
                <button className ="btn4" onClick={()=>handleComplete("confirm",selection)}>Send Request &gt;</button>


            </div>
        </div>
    </div>
)
```

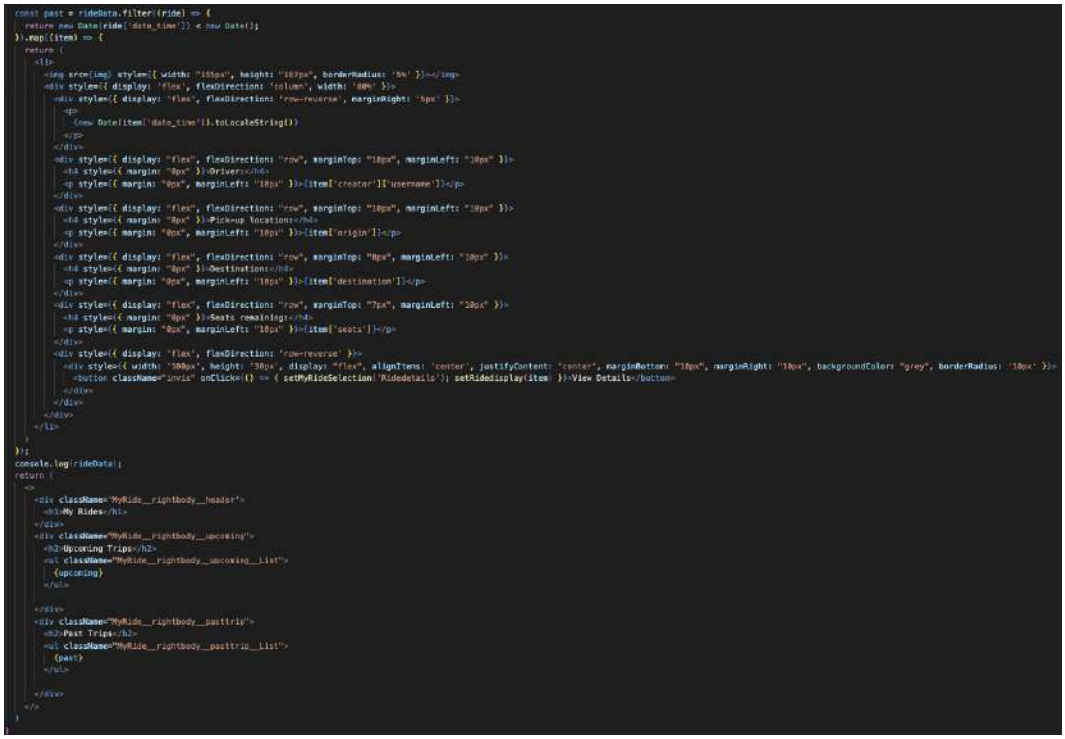
This function displays the SearchRideUI, after the user has selected a ride from the display in **3.1.1.8 Search Rides - Display Rides.** It displays the full ride information along with a go back and send request button. “Go back” returns the user to the main display page **3.1.1.8 Search Rides - Display Rides.** “Send request” brings the user to **3.1.1.10 Search Rides - Display Ride Request Sent**

3.1.1.10 Search Rides - Display Ride Request Sent

```
return(
    <div className = "search-ride" style={{height:"300px"}}>
        <div className="search-ride-header">
            <img src ="assets/bookRideIcon.png" width="30px" style={{marginTop:"10px"}}></img>
            <h1 className="search-ride-headerText">Request Confirmed</h1>
        </div>
        <div className="search-ride-body" style={{display:"flex", flexDirection:"column", alignItems:"center"}}>
            <div style={{margin:"20px"}}>
                <h1 style={{fontSize:"2em"}}>
                    Ride Request sent successfully
                </h1>
            </div>
            <button className="btn2" onClick = {closeWindow}>Go to MyRides</button>
        </div>
    </div>

)
```

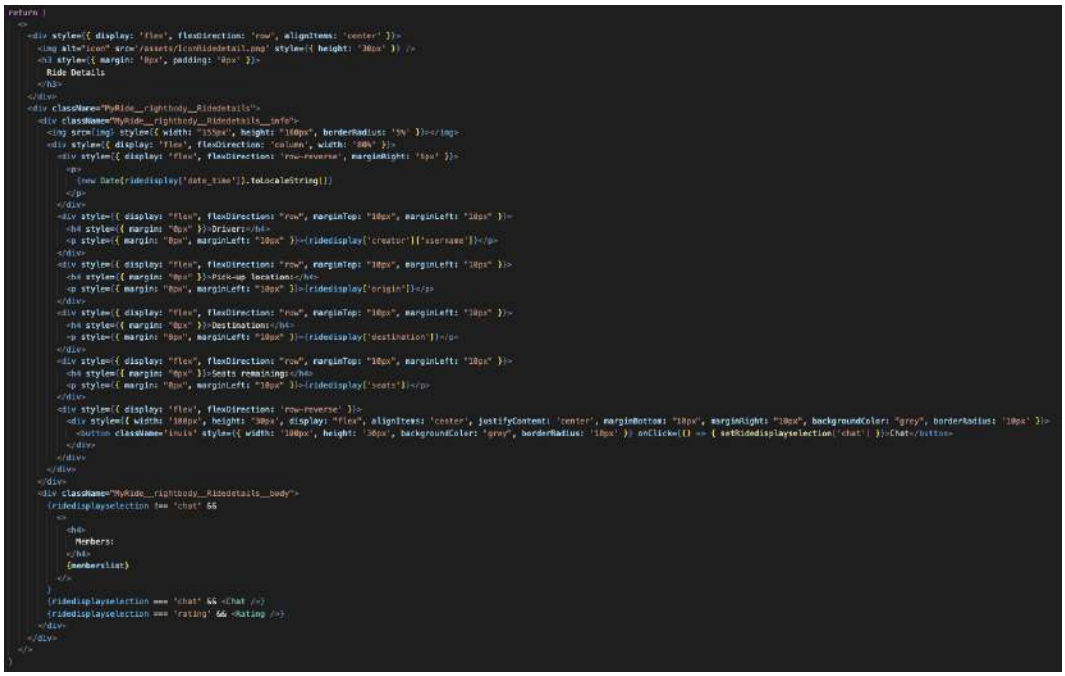
This function displays the SearchRideUI, after the *user* has clicked on ‘Confirm Request’ button to proceed to request for the ride. It displays the request successfully sent page with a button that brings the user to the **3.1.1.11 MyRides - Main**

3.1.1.11 MyRides - Main



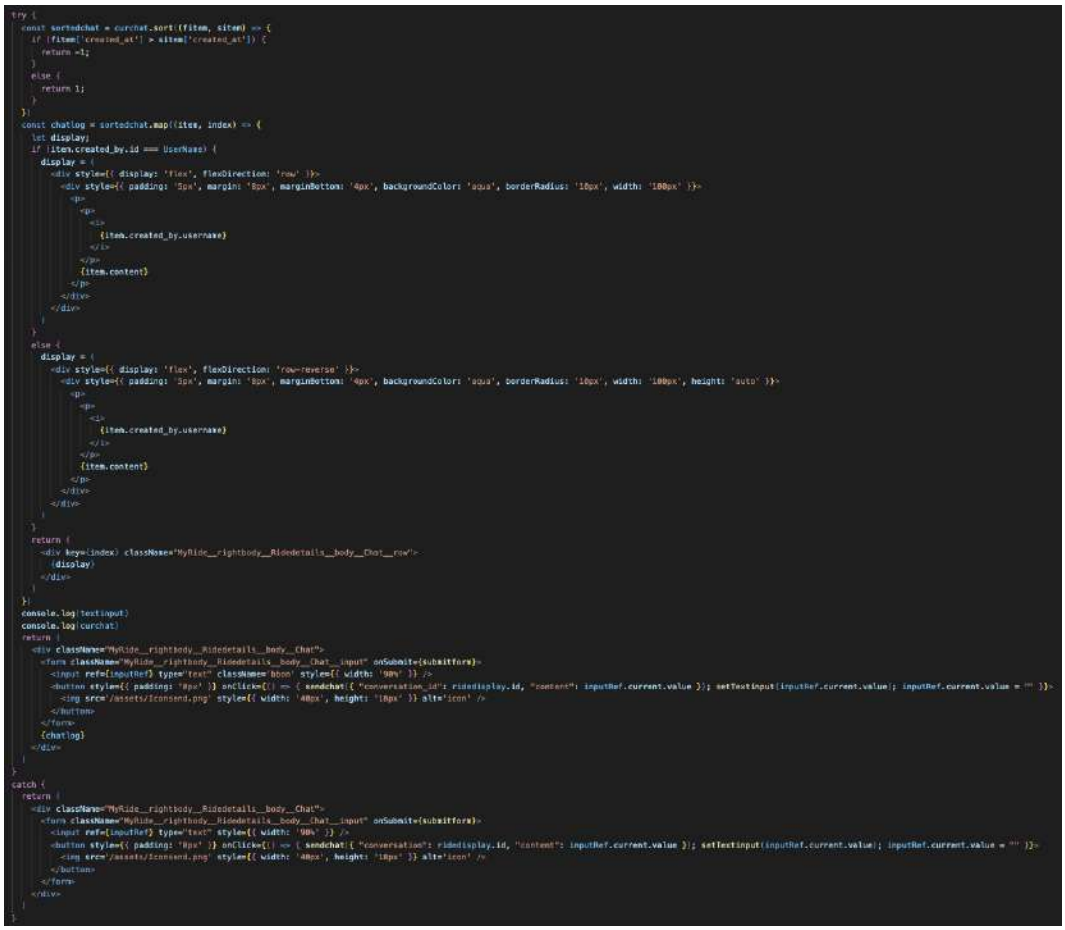
This function displays the MyRidesUI main page, which shows all the rides related to the *user*. It calls the MyRidesController to return all rides that the user has “Upcoming Trips” and “Past Trips” in the database. It will then format the data and display it to the user. MyRidesUI also contains category tabs for **3.1.1.14 MyRides - Ride Requests** and **3.1.1.15 MyRides - FAQ.**

3.1.1.12 MyRides - Ride Details



This function displays the MyRidesUI Ride Details, which shows the ride details. It calls the MyRidesController to return all ride details and the usernames of the other carpool passengers and driver. It will then format the data and display it to the user. MyRidesUI also links to the **3.1.1.13 MyRides - Chat.**

3.1.1.13 MyRides - Chat



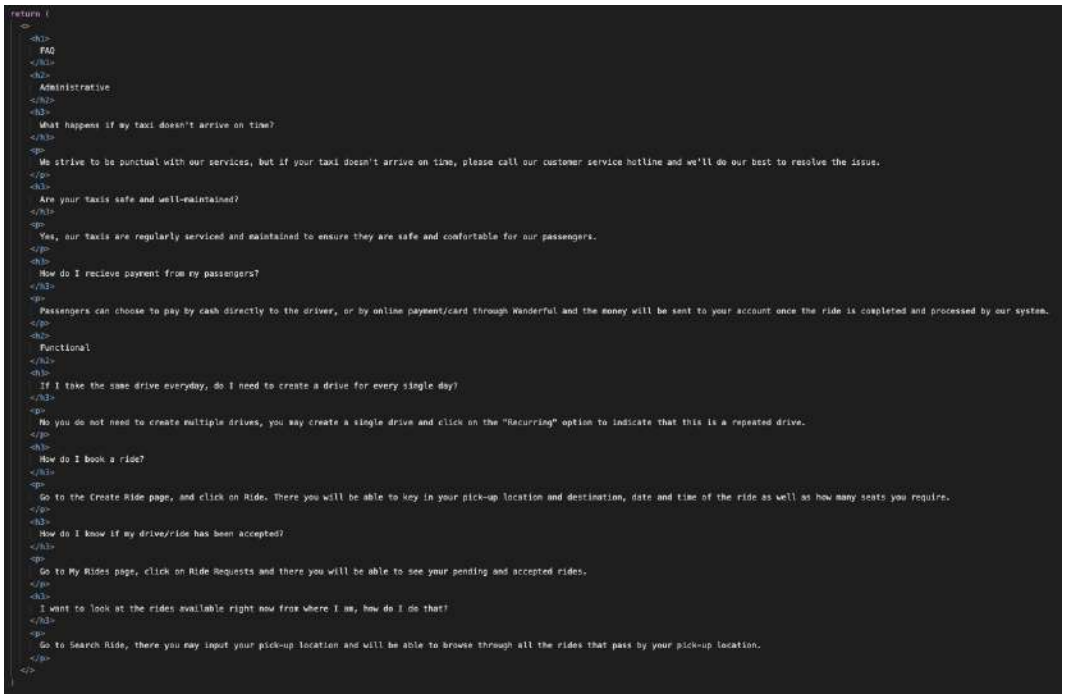
This function displays the MyRidesUI Chat page, where user can communicate and send text messages to other carpool passengers and driver. It calls the MyRidesController and the ChatController for message history and also to read and write new messages that has been sent. It will then format the data and display it to the user.

3.1.1.14 My Rides - Ride Requests

```
const accepted = riderequest.filter((ride) => {
  return ride['attributes']['status'] === 'Accepted';
}).map((item) => {
  return (
    <li>
      <img src={img} style={{ width: "155px", height: "160px", borderRadius: '5%' }}></img>
      <div style={{ display: 'flex', flexDirection: 'column', width: '80%' }}>
        <div style={{ display: 'flex', flexDirection: 'row-reverse', marginRight: '5px' }}>
          <p>
            {new Date(item['attributes']['ride']['date_time']).toLocaleString()}
          </p>
        </div>
        <div style={{ display: "flex", flexDirection: "row", marginTop: "10px", marginLeft: "10px" }}>
          <h4 style={{ margin: "0px" }}>Passenger:</h4>
          <p style={{ margin: "0px", marginLeft: "10px" }}>{item['attributes']['passenger']['username']}</p>
        </div>
        <div style={{ display: "flex", flexDirection: "row", marginTop: "20px", marginLeft: "10px" }}>
          <h4 style={{ margin: "0px" }}>Pick-up location:</h4>
          <p style={{ margin: "0px", marginLeft: "10px" }}>{item['attributes']['ride']['origin']}</p>
        </div>
        <div style={{ display: "flex", flexDirection: "row", marginTop: "20px", marginLeft: "10px" }}>
          <h4 style={{ margin: "0px" }}>Destination:</h4>
          <p style={{ margin: "0px", marginLeft: "10px" }}>{item['attributes']['ride']['destination']}</p>
        </div>
        <div style={{ height: '30px' }}>
        </div>
      </div>
    </li>
  )
});
return (
  <>
    <div className='MyRide__rightbody__header'>
      <h1>Ride Requests</h1>
    </div>
    <div className="MyRide__rightbody__upcoming">
      <h2>Pending</h2>
      <ul className="MyRide__rightbody__upcoming__List">
        {pending}
      </ul>

    </div>
    <div className="MyRide__rightbody__pasttrip">
      <h2>Accepted</h2>
      <ul className="MyRide__rightbody__pasttrip__List">
        {accepted}
      </ul>

    </div>
  </>
)
```

This function displays the MyRidesUI Ride Requests, which shows all the ride requests related to the *user*. It calls the MyRidesController to return all ride requests that the user has “Pending” and “Accepted” in the database. It will then format the data and display it to the user.

3.1.1.15 MyRides - FAQ



```
return (
  <>
    <h1>
      FAQ
    </h1>
    <h2>
      Administrative
    </h2>
    <h3>
      What happens if my taxi doesn't arrive on time?
    </h3>
    <p>
      We strive to be punctual with our services, but if your taxi doesn't arrive on time, please call our customer service hotline and we'll do our best to resolve the issue.
    </p>
    <h3>
      Are your taxis safe and well-maintained?
    </h3>
    <p>
      Yes, our taxis are regularly serviced and maintained to ensure they are safe and comfortable for our passengers.
    </p>
    <h3>
      How do I recieve payment from my passengers?
    </h3>
    <p>
      Passengers can choose to pay by cash directly to the driver, or by online payment/card through Wanderful and the money will be sent to your account once the ride is completed and processed by our system.
    </p>
    <h2>
      Functional
    </h2>
    <h3>
      If I take the same drive everyday, do I need to create a drive for every single day?
    </h3>
    <p>
      No you do not need to create multiple drives, you may create a single drive and click on the "Recurring" option to indicate that this is a repeated drive.
    </p>
    <h3>
      How do I book a ride?
    </h3>
    <p>
      Go to the Create Ride page, and click on Ride. There you will be able to key in your pick-up location and destination, date and time of the ride as well as how many seats you require.
    </p>
    <h3>
      How do I know if my drive/ride has been accepted?
    </h3>
    <p>
      Go to My Rides page, click on Ride Requests and there you will be able to see your pending and accepted rides.
    </p>
    <h3>
      I want to look at the rides available right now from where I am, how do I do that?
    </h3>
    <p>
      Go to Search Ride, there you may input your pick-up location and will be able to browse through all the rides that pass by your pick-up location.
    </p>
  </>
)
```

This function displays the MyRides FAQ page which allows the users to view frequently asked questions whenever in doubt of what Wanderful does and how Wanderful works.

3.1.1.16 Carparks

```
return (
    <div className="search-ride">

        <div className="search-ride-header">

            <img src ="assets/carparkIcon.png" width="30px" style={{marginTop:"10px"}}></img>
            <h1 className="search-ride-headerText">Carparks availability</h1>

        </div>
        <div className = "search-ride-upper">
            <div className="search-ride-searchbar">
                <textarea rows={1}placeholder ="Search nearest location" value ={search} onChange={e =>setSearch(e.target.value)} >
                </textarea>
                <div className="search-ride-searchIcon">
                    <img src = "assets/searchIcon.png" width="15px" style={{marginTop:"40%"}} onClick ={handleSearch}></img>
                </div>
            </div>
            <div className="search-ride-result">
                <h4 >Searched: </h4><h4 style= {{borderBottom : "solid"}}>{prevSearch}</h4>
            </div>

        </div>
        <div className="CarparkButtons">
            <button id="show"className="btn2"onClick={() =>{
                setCarparkMarker(filteredCarparks)
                show.style.backgroundColor =  "rgb(94, 184, 94)"
            }}>
                Show all on map
            </button>
            <button id="lots"className="btn2"onClick={()=>{
                setFilter("Lots")
                show.style.backgroundColor = "#A8B5E0"
                lots.style.backgroundColor =  "rgb(94, 184, 94)"
                dist.style.backgroundColor =  "#A8B5E0"
                }}>
                Filter by lots available
            </button>
            <button id="dist"className="btn2"onClick={()=>{
                setFilter("Distance")
                dist.style.backgroundColor =  "rgb(94, 184, 94)"
                lots.style.backgroundColor =  "#A8B5E0"
                show.style.backgroundColor = "#A8B5E0"
                }}>
                Filter by distance
            </button>
        </div>
        <div className="search-ride-body">
            {displayCarparks}
        </div>
    </div>
)
```

This function displays the CarparkAvailabilityUI which allows the users to input any location. It calls the CarparkController which returns a list of all taxis that are in a 0.5km radius from the input location. It will then format the data and display it to the user.

3.1.1.17 Taxis

This function displays the TaxiAvailabilityUI which allows the users to input any location. It calls the TaxiController which returns a list of all taxis that are in a 1km radius from the input location. It will then format the data and display it to the user.

3.1.2 Controller Classes

3.1.2.1 User Controller

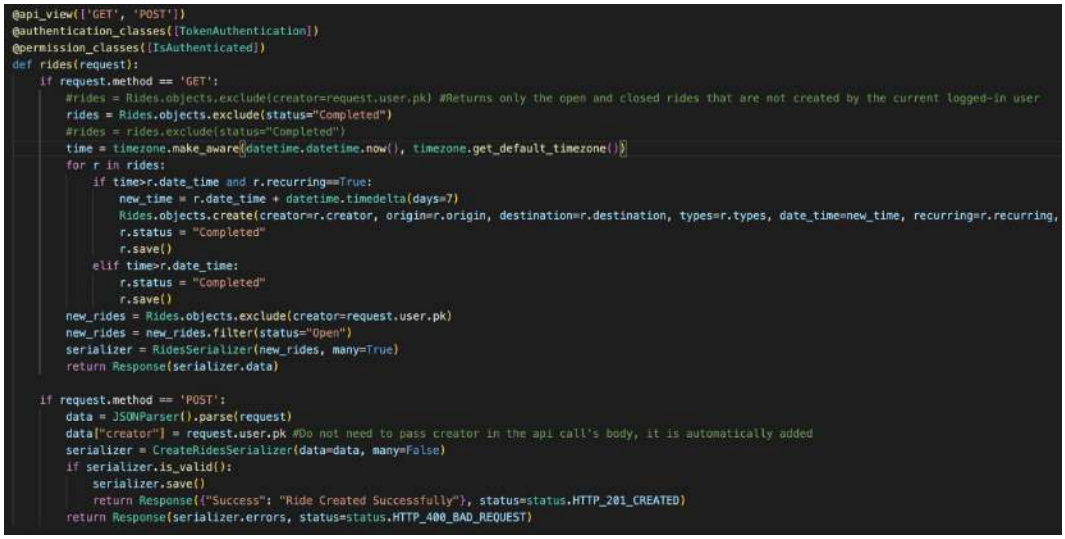
```
@csrf_exempt
def login(request):
    if request.method =='POST':
        data = JSONParser().parse(request)
        print(data)
        serializer = UserProfilesSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return JsonResponse(serializer.data)
        return JsonResponse(serializer.errors)
    else:
        return HttpResponse("failed")

@api_view(['POST'])
@permission_classes([AllowAny])
def signup(request):
        if request.method == 'POST':
            data=JSONParser().parse(request)
            print(data)
            serializer = UserSerializer(data=data)

            if serializer.is_valid():
                serializer.save()

                return Response({'Success': 'User created successfully'}, status=status.HTTP_201_CREATED)

            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

This function is used to help users create an account and login

```
@api_view(['GET', 'POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def myUserProfile(request):
    if request.method == 'GET':
        try:
            user = request.user
            data = UserProfiles.objects.get(user = user)
            serializer = UserProfilesSerializer(data, many=False)
            return JsonResponse(serializer.data)
        except:
            return JsonResponse({'failed':"failed"},status=status.HTTP_400_BAD_REQUEST)

    if request.method == 'POST':
        data = JSONParser().parse(request)
        data["user"] = request.user
        userprofile = UserProfiles.objects.get(user = request.user)
        serializer = UpdateUserProfilesSerializer(userprofile, data=data, many=False)
        if serializer.is_valid():
            serializer.save()
            profile_serialize = UserProfilesSerializer(userprofile, many=False)
            return JsonResponse(profile_serialize.data, status=status.HTTP_201_CREATED)
        return JsonResponse(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

This function is used to access the user data of the current user to get information to be returned to the UI.

3.1.2.2 Rides controller



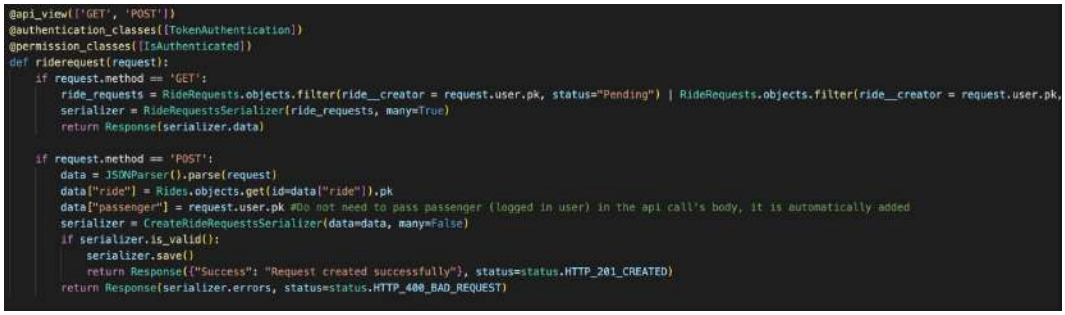
```
@api_view(['GET', 'POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def rides(request):
    if request.method == 'GET':
        #rides = Rides.objects.exclude(creator=request.user.pk) #Returns only the open and closed rides that are not created by the current logged-in user
        rides = Rides.objects.exclude(status="Completed")
        #rides = rides.exclude(status="Completed")
        time = timezone.make_aware(datetime.datetime.now(), timezone.get_default_timezone())
        for r in rides:
            if time>r.date_time and r.recurring==True:
                new_time = r.date_time + datetime.timedelta(days=7)
                Rides.objects.create(creator=r.creator, origin=r.origin, destination=r.destination, types=r.types, date_time=new_time, recurring=r.recurring,
                r.status = "Completed"
                r.save()
            elif time>r.date_time:
                r.status = "Completed"
                r.save()
        new_rides = Rides.objects.exclude(creator=request.user.pk)
        new_rides = new_rides.filter(status="Open")
        serializer = RidesSerializer(new_rides, many=True)
        return Response(serializer.data)

    if request.method == 'POST':
        data = JSONParser().parse(request)
        data["creator"] = request.user.pk #Do not need to pass creator in the api call's body, it is automatically added
        serializer = CreateRidesSerializer(data=data, many=False)
        if serializer.is_valid():
            serializer.save()
            return Response({"Success": "Ride Created Successfully"}, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

This function is used to create a ride and return the rides every user has created

```
@api_view(['GET'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def myrides(request):
    user = request.user
    data = RideRequests.objects.filter((Q(passenger=user) | Q(ride__creator=user)) & Q(status='Accepted')).values('ride__id').distinct()
    newdata = Rides.objects.filter(Q(id__in=data) | Q(creator=user))
    serializer = RidesSerializer(newdata, many=True)
    return JsonResponse(serializer.data, safe= False, status=status.HTTP_200_OK)
```

This function return all the rides that the logged in user is a part of for the *My Rides* page

3.1.2.3 Ride Requests Controller



```
@api_view(['GET', 'POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def riderequest(request):
    if request.method == 'GET':
        ride_requests = RideRequests.objects.filter(ride__creator = request.user.pk, status="Pending") | RideRequests.objects.filter(ride__creator = request.user.pk,
        serializer = RideRequestsSerializer(ride_requests, many=True)
        return Response(serializer.data)

    if request.method == 'POST':
        data = JSONParser().parse(request)
        data["ride"] = Rides.objects.get(id=data["ride"]).pk
        data["passenger"] = request.user.pk #Do not need to pass passenger (logged in user) in the api call's body, it is automatically added
        serializer = CreateRideRequestsSerializer(data=data, many=False)
        if serializer.is_valid():
            serializer.save()
            return Response({"Success": "Request created successfully"}, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

This function is used to create a ride request and let the creators of the ride view the ride requests made

```
@api_view(['POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def handle_request(request):
    if request.method == 'POST':
        data = JSONParser().parse(request)
        ride = Rides.objects.get(id=data["ride_id"])
        riderequest = RideRequests.objects.get(id=data["request_id"])
        riderequest.status = data["status"]
        riderequest.save()
        if data["status"] == "Accepted":
            ride.seats -= 1
            ride.save()
            conversations = Conversation.objects.get(rides=ride)
            passenger = riderequest.passenger
            conversations.members.add(passenger)
            conversations.save()
        if ride.seats < 0:
            ride.seats = 0
            ride.save()
        if ride.seats == 0:
            ride.status = "Closed"
            ride.save()
        return redirect("/core/riderequest/")
```

This function is used to handle a ride request and includes the logic of what happens if a ride request is accepted/rejected

3.1.2.4 Chat Controller

```
@api_view(['GET'])
@authentication_classes([TokenAuthentication,])
@permission_classes([IsAuthenticated,])
def chat(request,index):
    user=request.user
    try:
        conversationdata = Conversation.objects.get(rides=index ,  members=user)
        messagedata = ConversationMessage.objects.filter(conversation = conversationdata)
        print(messagedata)
        serializer = ConversationMessageSerializer(messagedata, many=True)
        return JsonResponse(serializer.data, safe=False)
    except ObjectDoesNotExist:
        return HttpResponse("ObjectDoesNotExist")
```

This function returns all the chats of which the logged in user is a part

```
@api_view(['POST'])
@authentication_classes([TokenAuthentication])
@permission_classes([IsAuthenticated])
def sendchat(request):
    try:
        data = JSONParser().parse(request)
        data["created_by"] = request.user.pk
        data["conversation"] = Conversation.objects.get(rides__id=data["conversation_id"]).pk
        serializer = CreateConversationMessageSerializer(data=data, many=False)
        if not serializer.is_valid():
            print(serializer.errors)
            return Response(serializer.errors)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response({'Success': 'Message sent successfully'}, status=status.HTTP_201_CREATED)
    except ValidationError as e:
        return Response({'Failed': str(e)}, status=status.HTTP_400_BAD_REQUEST)
    except Exception as e:
        return Response({'Failed': 'Message failed to be sent'}, status=status.HTTP_400_BAD_REQUEST)
```

This function is used to create a message in a particular conversation and send it in the chat

3.1.2.5 Carpark Availability API call

```
def get_token():
    url = 'https://www.ura.gov.sg/uraDataService/insertNewToken.action'
    headers = CaseInsensitiveDict()
    headers= {
        "User-Agent": "Myhost",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8",
        "Accept-Encoding": "gzip, deflate",
        "Connection": "keep-alive",
        "Pragma": "no-cache",
        "Cache-Control": "no-cache",
        "Accept-Language": "en-US,en;q=0.5",
        "AccessKey": "70c5cc64-7065-40e8-9921-3b068b8f237e"
    }
    response = requests.get(url, headers=headers)
    response_data = response.text
    return response_data


#/api/carpark/
@api_view(['GET'])
@authentication_classes([TokenAuthentication])
@permission_classes([AllowAny])
def carpark(request):
    tokenResponse = get_token()
    tokenResponseJson = json.loads(tokenResponse)
    token = tokenResponseJson["Result"]
    url = "https://www.ura.gov.sg/uraDataService/invokeUraDS?service=Car_Park_Availability"
    headers = CaseInsensitiveDict()
    headers= {
        "User-Agent": "Myhost",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8",
        "Accept-Encoding": "gzip, deflate",
        "Connection": "keep-alive",
        "Pragma": "no-cache",
        "Cache-Control": "no-cache",
        "Accept-Language": "en-US,en;q=0.5",
        "AccessKey": "70c5cc64-7065-40e8-9921-3b068b8f237e",
        "Token": token
    }
    response = requests.get(url, headers=headers)
    response_data = response.json()
    return Response(response_data)
```

This function is used to make an API call to get the latest carpark availability information

3.1.2.6 Taxi Availability API call

```
#/api/taxi/
@api_view(['GET'])
@authentication_classes([TokenAuthentication])
@permission_classes([AllowAny])
def taxiAvailability(request):
    url = "https://api.data.gov.sg/v1/transport/taxi-availability"
    response = requests.get(url)
    response_data = response.json()
    return Response(response_data)
```

This function is used to make an API call to get the latest taxi locations

3.1.3 Entity Classes

3.1.3.1 User Entity

```python
class AbstractUser(AbstractBaseUser, PermissionsMixin):
    """
    An abstract base class implementing a fully featured User model with
    admin-compliant permissions.

    Username and password are required. Other fields are optional.
    """

    username_validator = UnicodeUsernameValidator()

    username = models.CharField(
        _("username"),
        max_length=150,
        unique=True,
        help_text=_(
            "Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only."
        ),
        validators=[username_validator],
        error_messages={
            "unique": _("A user with that username already exists."),
        },
    )
    first_name = models.CharField(_("first name"), max_length=150, blank=True)
    last_name = models.CharField(_("last name"), max_length=150, blank=True)
    email = models.EmailField(_("email address"), blank=True, unique=True)
    is_staff = models.BooleanField(
        _("staff status"),
        default=False,
        help_text=_("Designates whether the user can log into this admin site."),
    )
    is_active = models.BooleanField(
        _("active"),
        default=True,
        help_text=_(
            "Designates whether this user should be treated as active. "
            "Unselect this instead of deleting accounts."
        ),
    )
    date_joined = models.DateTimeField(_("date joined"), default=timezone.now)

    objects = UserManager()

    EMAIL_FIELD = "email"
    USERNAME_FIELD = "username"
    REQUIRED_FIELDS = ["email"]
```

This class contains data for the users

```python
class UserProfiles(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    bio = models.TextField(blank=True, null=True)
    profile_pic = models.ImageField(upload_to='profile_pics', blank=True, null=True)
    avg_rating = models.IntegerField(default=0)

    class Meta:
        verbose_name = ("User Profile")

    def __str__(self):
        return f"{self.user}"
```

This class contains data for User Profiles

3.1.3.2 Rides Entity

```
class Rides(models.Model):
    creator = models.ForeignKey(User, related_name='creator', on_delete=models.CASCADE)
    origin = models.CharField(max_length=100)
    destination = models.CharField(max_length=100)
    CHOICES = (
        ('Taxi', 'Taxi'),
        ('Personal Car', 'Personal Car'),
    )
    types = models.CharField(max_length=15, choices=CHOICES, default='Personal Car')
    date_time = models.DateTimeField()
    recurring = models.BooleanField()
    seats = models.PositiveSmallIntegerField()
    start_lat = models.CharField(max_length=50, null=True)
    end_lat = models.CharField(max_length=50, null=True)
    CHOICES = (
        ('Open', 'Open'),
        ('Closed', 'Closed'),
        ('Completed', 'Completed')
    )
    status = models.CharField(max_length=10, choices = CHOICES, default='Open')

    class Meta:
        verbose_name = ("Ride")
        get_latest_by = ["date_time"]

    def __str__(self):
        return f"{self.origin} to {self.destination} with {self.types}"
```

```
class RideRequests(models.Model):
    ride = models.ForeignKey(Rides, on_delete=models.CASCADE)
    passenger = models.ForeignKey(User, on_delete=models.CASCADE)
    CHOICES = (
        ('Accepted', 'Accepted'),
        ('Pending', 'Pending'),
        ('Rejected', 'Rejected'),
    )
    status = models.CharField(max_length=10, choices=CHOICES, default='Pending')

    class Meta:
        verbose_name = ("Ride Request")

    def __str__(self):
        return f"{self.passenger} requests to ride with {self.ride}"
```

This class contains data for the Rides and RideRequests entity.

3.1.3.3 Chat entity

```python
class Conversation(models.Model):
    rides = models.ForeignKey(Rides, related_name='conversations', on_delete=models.CASCADE)
    members = models.ManyToManyField(User, related_name='users')
    created_at = models.DateTimeField(auto_now_add=True)
    modified_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ('-modified_at',)

class ConversationMessage(models.Model):
    conversation = models.ForeignKey(Conversation, related_name='messages', on_delete=models.CASCADE)
    content = models.CharField(max_length=1000)
    created_by = models.ForeignKey(User, related_name='created_messages', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ('created_at',)

    def clean(self):
        if self.created_by not in self.conversation.members.all():
            raise ValidationError('The creator of the message must be a member of the conversation.')
```

This class contains data for the conversations and the messages sent in a particular conversation

3.2 Source Code

Frontend - https://github.com/weiyuan12/Software-Eng
Backend - https://github.com/AryanSethi20/Backend-wanderer

3.3 Demo Script

1. Registration
    a. Navigate to sign up
    b. Insert First Name: Bryan
    c. Insert Last Name: Tan
    d. Insert username: bryantan
    e. Insert email: bryantan@gmail.com
    f. Insert password: Password#1203
    g. *redirected to login page*

2. Login
    a. Insert username: bryantan
    b. Insert password: Password#1203
    c. *redirected to home page*

3. Create Ride
    a. Click on Create Ride from NavBar
    b. Select Personal Car / Taxi
    c. Enter Starting location and validate it: NTU, Nanyang Avenue
    d. Enter Ending location and validate it: Jurong East
    e. Enter date-time: 4/20/2023 - 6:03 PM
    f. Select type of ride: Recurring
    g. Enter the available seats: 3
    h. Select Submit

4. Make Ride Request
    a. Click on Search Rides from NavBar
    b. Browse through all the rides available
    c. Select a ride which matches your requirements and click on request
    d. Go through the details of the ride again and send request
    e. *redirected to *my rides* page*

5. My Rides
    a. Click on My Rides
    b. Browse through the upcoming and past rides

6. Carpark Availability
    a. Insert area name: Jurong East

b. Browse through the carparks available according to lots available
c. Click on *Show all on map* to look at the results on the map
d. Click on *filter by distance* to look at the carparks according to their proximity
e. Click on *Show all on map* to look at the new results on the map

7. Taxi Availability
    a. Insert area name: Nanyang Avenue
    b. Browse through the taxis available near the area

8. Logout and Login using John Cena's credentials
    a. Click on username at top right and then click on logout
    b. *redirected to login page*
    c. Insert username: johncena
    d. Insert password: Password#1203
    e. *redirected to home page*

9. Handle Ride Requests
    a. Click on My Rides from NavBar
    b. Click on Ride Requests
    c. Browse through all the requests and find bryantan's request
    d. Accept the request
    e. *Bryan Tan gets added to the ride*

10. Chat
    a. Click on My Rides from NavBar
    b. Browse through upcoming rides and pick the latest upcoming ride
    c. Click on *View Details*
    d. Browse through all the users part of the ride
    e. Click on *Chat* button
    f. Communicate with the users part of the ride
    g. Enter a message: Hi guys see you on the 17th!

11. Search Rides
    a. Enter in the search box: Nanyang Avenue
    b. Rides are filtered according to the starting location specified in the search box
    c. Browse through the rides to find bryantan's ride
    d. Make a request for the ride

12. Edit Profile
    a. Click on the username at top right
    b. Click on *Edit Profile*
    c. Make changes to the profile
    d. Click *Done*

13. Logout and Login using Bryan Tan's credentials
    a. Click on username at top right and then click on logout
    b. *redirected to login page*
    c. Insert username: bryantan
    d. Insert password: Password#1203
    e. *redirected to home page*

14. Reply to the chat
    a. Click on *My Rides* from NavBar
    b. Browse through the upcoming rides and find John Cena's ride
    c. Click on *View Details*
    d. Click on *Chat*
    e. Reply to the latest message in the chat: See you!

15. Handle Ride Requests
    a. Click on *My Rides* from NavBar
    b. Click on *Ride Requests*
    c. Browse through the ride requests and find John Cena's request
    d. Handle the request: Reject

16. FAQ Page
    a. Click on *My Rides* from NavBar
    b. Click on FAQ
    c. Browse through the FAQ page

3.4 Demo Video

Wanderful - A Ride Sharing Web App - Video Link: https://youtu.be/x5ggTytjw38

4. Testing

4.1 Black-Box Testing

4.1.1 Functionality Signup

Generic Cases

TestID	Test Input	Expected Output	Actual Output
1.a.1	Username already exist.	Return error message for username already taken	Return error message for username already taken
1.a.2	Invalid Email format	Colour of input bar remains white	Colour of input bar remains white
1.a.3	Password does not meet specification Requirements: 1 Uppercase, 1 lowercase	Colour of input bar remains white	Colour of input bar remains white
1.a.4	Re-enter password does not match	Colour of input bar remains white	Colour of input bar remains white
1.a.5	Submitting with invalid fields	Return error message corresponding to 1.1, 1.2, 1.3, 1.4	Return error message corresponding to 1.1, 1.2, 1.3, 1.4
1.a.6	Submitting with all fields valid	Redirected to login page	Redirected to login page

Specific cases

TestID	Test Input	Expected Output	Actual Output
1.b.1	Input fields Username: Joncena Email:jon@gma Password: Password#1203 Re-enter Password: Password#1203 Submit	Colour of email bar remains white Error message “Enter a valid email”	Colour of email bar remains white Error message “Enter a valid email”
1.b.2	Input fields Username: Joncena Email:jon@gmail.com Password: password Re-enter password: password Submit	Colour of password bar remains white Error message “Enter a valid password”	Colour of password bar remains white Error message “Enter a valid password”
1.b.3	Input fields Username: Joncena Email:jon@gmail.com Password: Password#1203 Re-enter password: Password#1202	Colour of re-enter password bar remains white	Colour of re-enter password bar remains white

	Submit	Error message “Password must match”	Error message “Password must match”
1.b.4	Input fields Username: Joncena Email:jon@gmail.com Password: Password#1203 Re-enter password: Password#1203 Submit	Sign up Successful Redirected to login page	Sign up Successful Redirected to login page

4.1.2 Functionality Login

Functionality: Login

TestID	**Test Input**	**Expected Output**	**Actual Output**
2.1	Invalid username and password	Returns error message for invalid login credentials	Returns error message for invalid login credentials
2.2	Valid username and password	Redirects to dashboard and logs into user’s account	Redirects to dashboard and logs into user’s account

4.1.3 Functionality: Create Ride

Generic cases

TestID	Test Input	Expected Output	Actual Output
3.a.1	User tries to verify an Invalid location	Return error message to prompt entering of valid location	Return error message to prompt entering of valid location
3.a.2	User tries to verify a Valid location	Verify button turns into green tick	Verify button turns into green tick
3.a.3	Submitting with invalid inputs	User is prompted to correct the invalid input	User is prompted to correct the invalid input
3.a.4	Submitting without validating location	Return error message to verify location	Return error message to verify location
3.a.5	Submitting with valid inputs	New ride is created Display “ride creation success”	New ride is created Display “ride creation success”

Specific cases

TestID	Test Input	Expected Output	Actual Output
3.b.1	Input fields Start Location: xaxaxa Click on “verify”	Error message “Please enter a valid start location”	Error message “Please enter a valid start location”
3.b.2	Input fields Start Location: Bukit timah Click on “verify”	Verify button turns into green tick	Verify button turns into green tick

3.b.3	Input fields End Location: xaxaxa Click on “verify”	Error message “Please enter a valid end location”	Error message “Please enter a valid end location”
3.b.4	Input fields End Location: Bukit timah Click on “verify”	Verify button turns into green tick	Verify button turns into green tick
3.b.5	Input fields Start Location: Bukit timah End Location: Boon Lay Time: 14-4-2023 8:20PM Seats: 0 Click “Submit”	Error message “Please enter a valid seat number:	Error message “Please enter a valid seat number:
3.b.6	Input fields Start Location: Bukit timah End Location: Boon Lay Time: 14-4-2023 8:20PM Seats: 0 Click “Submit”	New ride is created Display “ride creation success”	New ride is created Display “ride creation success”

4.1.4 Functionality: Search Ride

TestID	Test Input	Expected Output	Actual Output
4.1	Pressing search button without input	Return a list of all rides	Return a list of all rides
4.2	Pressing the search button with a location as input	Return a list of all rides that has destination of input location	Return a list of all rides that has destination of input location
4.3	Pressing “Request” on a search result	Displays the full ride details	Displays the full ride details
4.4	Pressing “Send Request” on the ride details page	Displays “Ride Request Successful” Decreases Seats available by 1 Sends a ride request to the creator	Displays “Ride Request Successful” Decreases Seats available by 1 Sends a ride request to the creator
4.5	Pressing “back” on the ride details page	Returns back to list of all rides	Returns back to list of all rides

4.1.5 Functionality: Handle Passenger Request

TestID	Test Input	Expected Output	Actual Output
5.1	Pressing ‘Accept’ on passenger request	Passenger request accepted and moved to accepted section	Passenger request accepted and moved to accepted section
5.2	Pressing ‘Reject’ on passenger request	Passenger request is rejected and is not shown	Passenger request is rejected and is not shown

4.1.6 Functionality: Chat

TestID	Test Input	Expected Output	Actual Output
6.1	Clicking ‘chat’ button	Displays conversation history for current chat	Displays conversation history for current chat
6.2	Inputing message and pressing send	Sends message to all members of the chat Updates conversation history to show message	Sends message to all members of the chat Updates conversation history to show message

4.1.7 Functionality: Taxi Availability

TestID	Test Input	Expected Output	Actual Output
7.1	Pressing search button with invalid input	Return error message to enter a valid location	Return error message to enter a valid location
7.2	Pressing search button with valid input	“Show taxis nearby” button is displayed	“Show taxis nearby” button is displayed
7.3	Pressing the “Show taxis nearby” button	Markers are placed on the map for the input location and for the current location of each taxi	Markers are placed on the map for the input location and for the current location of each taxi
7.4	Searching another input	The previous markers will be cleared and the expected result of 11.1 or 11.2 will be displayed	The previous markers will be cleared and the expected result of 11.1 or 11.2 will be displayed

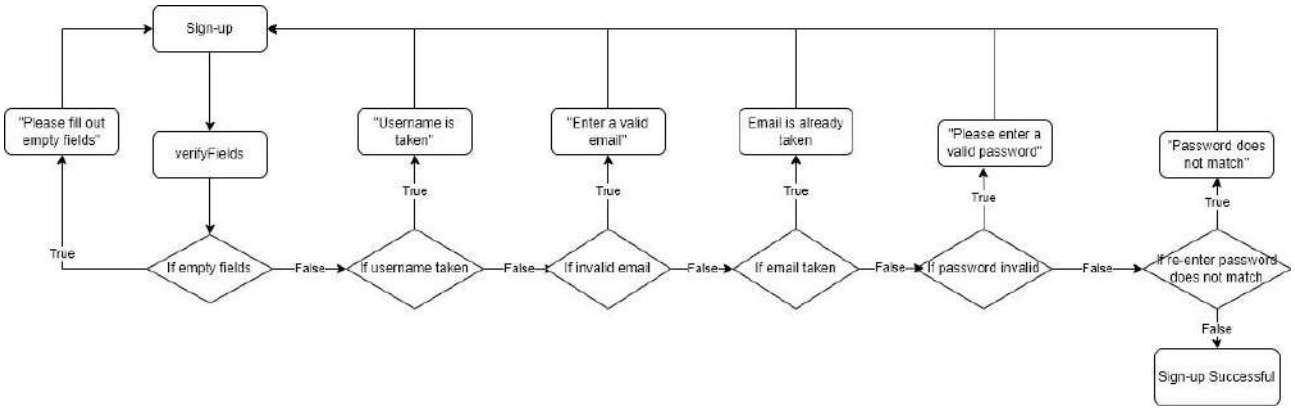
4.1.8 Functionality: Carpark Availability

TestID	Test Input	Expected Output	Actual Output
8.1	Pressing search button with invalid input	Return error message to enter a valid location	Return error message to enter a valid location
8.2	Pressing search button with valid input	All carparks within a 1km radius is displayed in a list to the user, by default filtered by lots available	All carparks within a 1km radius is displayed in a list to the user, by default filtered by lots available
8.3	Pressing “filter by distance”	“filter by distance” button will turn green “filter by lots available” button will turn grey All carparks within a 1km radius is displayed in a list to the user, filtered by distance	“filter by distance” button will turn green “filter by lots available” button will turn grey All carparks within a 1km radius is displayed in a list to the user, filtered by distance
8.4	Pressing “filter by lots available”	“filter by distance” button will turn grey “filter by lots available” button will turn green All carparks within a	“filter by distance” button will turn grey “filter by lots available” button will turn green

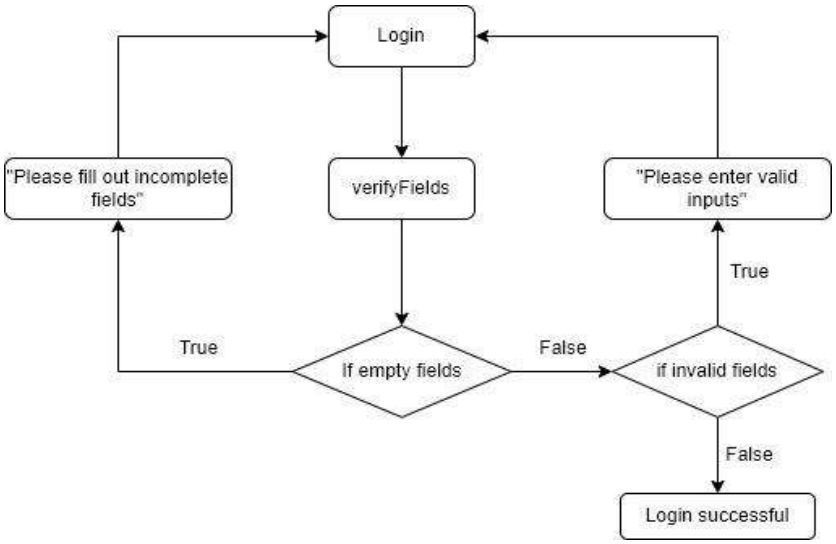
		1km radius is displayed in a list to the user, filtered by lots available	All carparks within a 1km radius is displayed in a list to the user, filtered by lots available
8.5	Pressing “Show on map” button	Markers are placed on the map corresponding to the carpark location	Markers are placed on the map corresponding to the carpark location

4.2 White-Box Testing

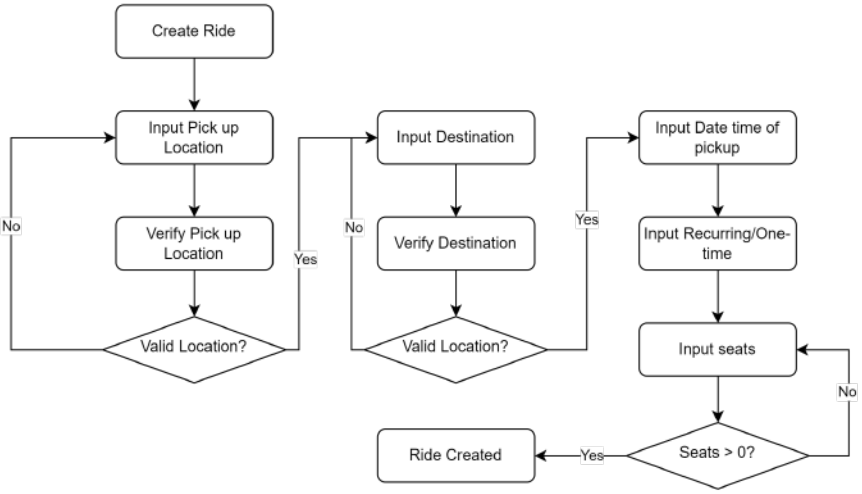
4.2.1 Functionality: Sign-up



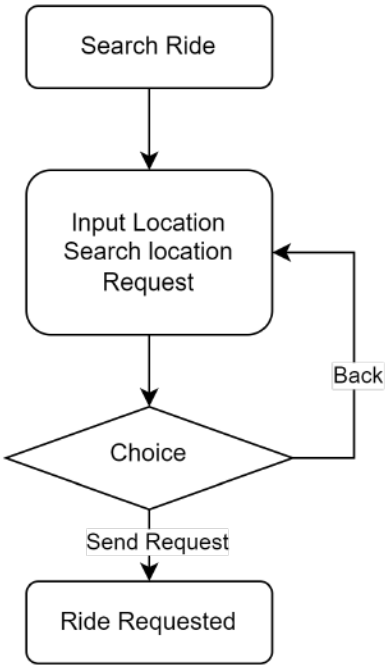
4.2.2 Functionality: Login



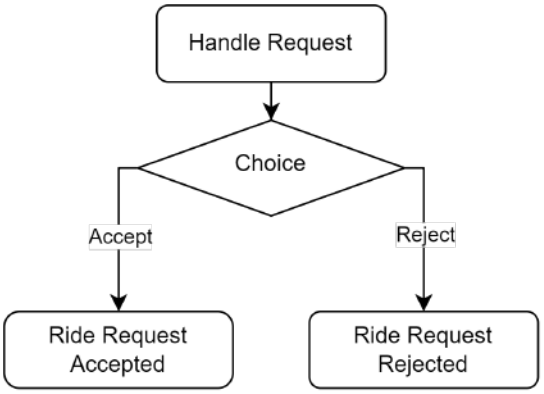
4.2.3 Functionality: Create Ride



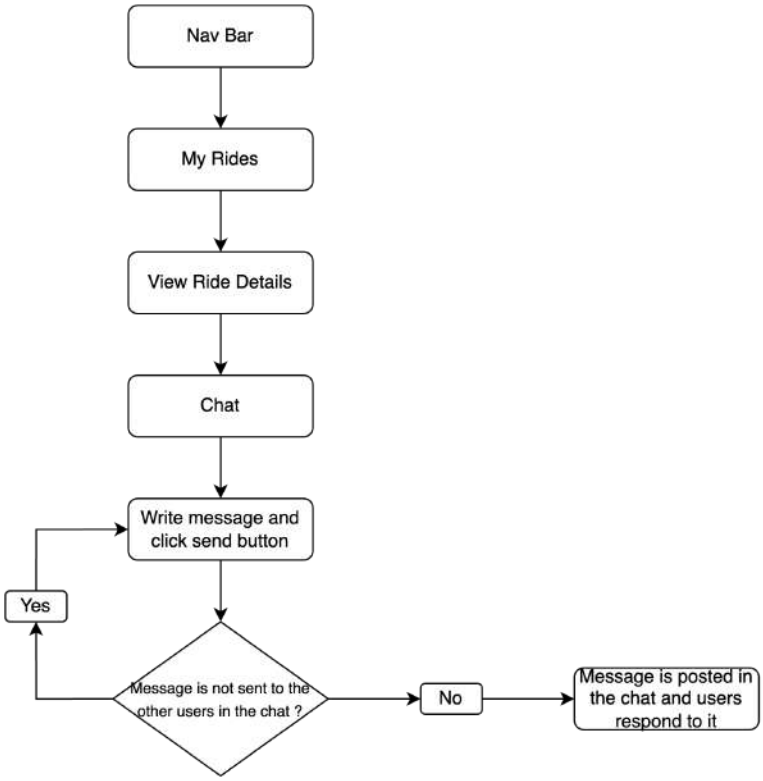
4.2.4 Functionality: Search Ride



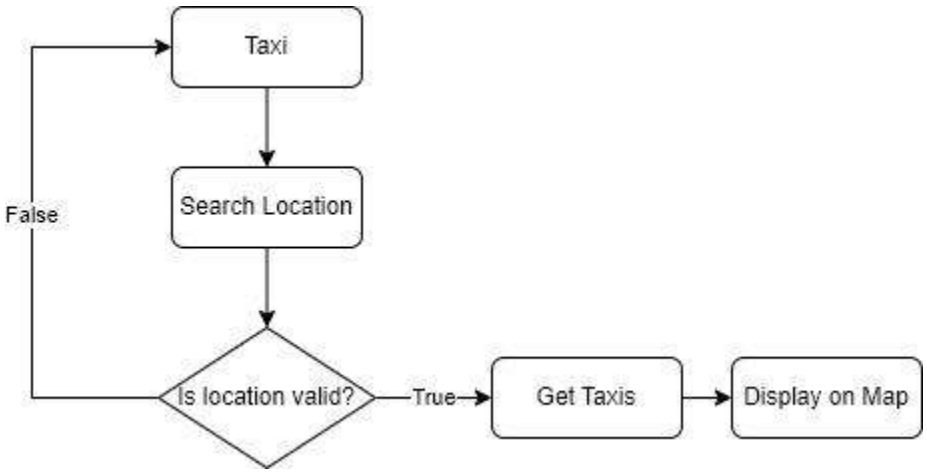
4.2.5 Functionality: Handle Request



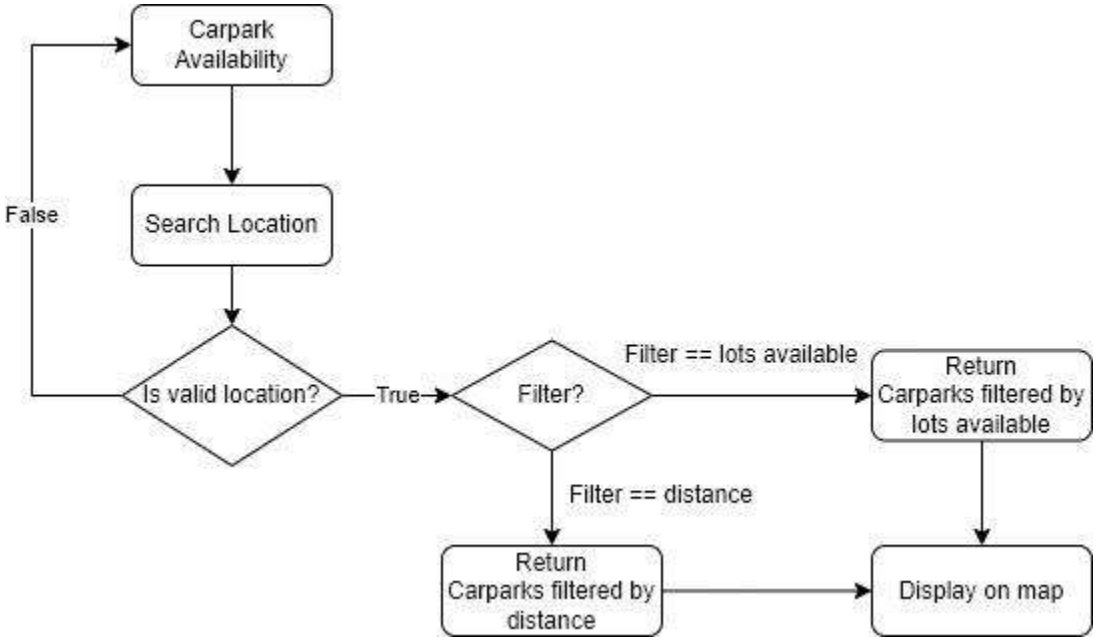
4.2.6 Functionality: Chat



4.2.7 Functionality: Taxi Availability



4.2.8 Functionality: Carpark Availability



5. References

Bruegge, & Dutoit, A. H. (2010). Object-oriented software engineering : using UML, patterns, and Java (3rd ed.). Prentice Hall.

Sommerville, I. (2016). Software Engineering. 10th Edition, Pearson Education Limited, Boston.

Aryan Sethi. (2023, April 16). Wanderful [Video]. YouTube. https://www.youtube.com/watch?v=x5ggTytjw38

Aryan Sethi. (2023). Backend-wanderer. GitHub. https://github.com/AryanSethi20/Backend-wanderer

Wei Yuan. (2023). Software-Eng. Github https://github.com/weiyuan12/Software-Eng

6. Appendix

6.1 Meeting Minutes

Week | Date:	**Task:**
Week 1 | 30-01-2023	☑ Define Problem Statement & Project Features ☑ Appoint a Group Leader (Jarel) ☑ Created Document for Lab 1 Assignment Deliverables: ☑ Documentation of Functional Requirements **(Jonathan)** ☑ Documentation of Non-Functional Requirements **(Jarel)** ☑ Data Dictionary **(Aryan)** ☑ Initial Use Case Model ☑ Use Case Diagram **(Aryan)** ☑ Use Case Descriptions **(Wei Yuan)** ☑ UI Mockups **(Dann)**
Week 2 | 06-02-2023	**Weekly Review 1:** Online Zoom Meeting to discuss the following: ☑ Refine documentation of Functional Requirements ☑ Refine documentation of Non-Functional Requirements ☑ Complete Data Dictionary ☑ Refine Use Case Diagram and Descriptions ☑ Feedback on UI Mockup Design
Week 3 | 13-02-2023	☑ Discuss Lab Deliverables and Split the workload: ☑ Update & Complete Use Case Diagram **(Jarel)** ☑ Update & Complete Use Case Descriptions **(Jarel)** ☑ Create Class Diagrams of Entity Classes **(Jonathan)** ☑ Create Key Boundary & Control Classes **(Aryan)** ☑ Sequence Diagrams of 5 Use Cases **(Wei Yuan)** ☑ Create Dialog Map **(Aryan)** ☑ Update UI Mockup as per feedback **(Dann)**

Week 4 | 20-02-2023	**Weekly Review 2:** Online Zoom Meeting to discuss the following ☑ Use Case Diagram and Descriptions ☑ Class Diagrams ☑ Boundary & Control Class Diagrams ☑ Sequence Diagrams ☑ Dialog Map ☑ UI Mockups
Week 5 | 27-02-2023	☑ Discuss Lab Deliverables and Split the workload: ☑ Complete Use Case Model **(Jarel)** ☑ Finalise Design Models: ☑ Class Diagrams of Entity Classes ☑ Key Boundary & Control Classes ☑ Sequence Diagrams of 5 Use Cases ☑ Dialog Map ☑ Update UI Mockup as per feedback **(Dann)** ☑ System Architecture Diagram **(Dann)** ☑ Discuss Source Code Structure and Stack that is going to be used ☑ Start on Source Code **(Aryan, Wei Yuan, Jonathan)**
Week 6 | 06-03-2023	**Weekly Review 3:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend) ☑ Clarify the structure and refine diagrams ☑ System Architecture Diagram
Week 7 | 13-03-2023	**Weekly Review 4:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend) ☑ Refine Diagrams

Week 8 | 20-03-2023	**Weekly Review 5:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend)
Week 9 | 27-03-2023	**Weekly Review 6:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend)
Week 10 | 03-04-2023	**Weekly Review 7:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend)
Week 11 | 10-04-2023	**Weekly Review 8:** Online Zoom Meeting to discuss the following ☑ Source Code (Frontend) ☑ Source Code (Backend) ☑ Refine FINALISED Diagrams matching latest source code ☑ Craft Demo Script ☑ Develop Test Cases and Testing Results ☑ Do up Presentation Slides
Week 12 | 16-04-2023 **(Submission Date)**	☑ **Final troubleshoot of Source code** ☑ **Test and Run the Source Code fully** ☑ **Rehearse Demo Script** ☑ **Presentation & Live Demo on 12th April 2023** ☑ **Complete SRS Documentation** ☑ **Film Demo Video** ☑ **Finalise all Deliverables & Submit!**