In this document, replace words/phrases enclosed by { } with your own words/phrases and remove ALL { } and instructions written in blue.

**iCar Rental Analysis**

**Software Requirements Specification Document for**

**iCar Rental Service**

**Version 3.0**

A black text on a white background

Description automatically generated

**SOFTWARE ANALYSIS & DESIGN**

Year 2 (2024/25), Semester 3

SCHOOL OF INFOCOMM TECHNOLOGY

Diploma in Information Technology

**ASSIGNMENT 2**

**Duration:**  **15 Jul – 11 Aug 2024**

**Weightage:**  **40% of overall grade**

**Individual – 65%**

**Team – 35%**

**Format:** **Project, Report, Presentation and Demo**

**Deadline:**  **11 Aug 2024 (Sun), 23:59hrs**

**Penalty for late**  10% per day (including Saturday, Sunday

**submission:** and public holiday)

No submission will be accepted after

**16 Aug 2024 (Fri), 23:59hrs**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Module Group:** | **P03** |  |
|  | **Student Number** | **Student Name** | **Grade** |
| **1** | S10258457 | Yeo Jin Rong |  |
| **2** | S10257347 | Ng Jing Zhan Garrett |  |
| **3** | S10258272 | Liang Dingxuan |  |
| **4** | S10262552 | Ng Kai Huat Jason |  |
| **5** | S10262567 | Tan Guo Zhi Kelvin |  |

# Revision History

Describes what are the changes made between each version / iteration under the ‘Description’ column.

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| **25/05/2024** | **1.0** | **First Draft** |
| **09/06/2024** | **1.1** | **Final Submission for Stage 1** |
| **31/07/2024** | **2.0** | **First Draft for Stage 2 Submission** |
| **02/08/2024** | **2.1** | **Minor adjustments, added Jin Rong Content** |
| **06/08/2024** | **2.2** | **Consolidated report, removing “Student ID” labels for classes** |
| **09/08/2024** | **2.3** | **Checked use case descriptions, specifications and added sequence diagrams** |
| **10/08/2024** | **2.4** | **Checked all the classes and added attributes, multiplicities and methods** |
| **10/08/2024** | **3.0** | **Checked all classes, updated any errors** |

# Table of Contents

[Revision History 3](#_Toc174223223)

[Table of Contents 4](#_Toc174223224)

[1. Introduction 5](#_Toc174223225)

[1.1 Purpose 5](#_Toc174223226)

[1.2 Background 5](#_Toc174223227)

[1.3 Scope 7](#_Toc174223228)

[1.4 Definitions, Acronyms, and Abbreviations 9](#_Toc174223229)

[1.5 References 12](#_Toc174223230)

[2. Requirements 13](#_Toc174223231)

[2.1 Functional Requirements 13](#_Toc174223232)

[2.1.1 Functional Requirements by Yeo Jin Rong 13](#_Toc174223233)

[2.1.2 Functional Requirements by Ng Jing Zhan Garrett 18](#_Toc174223234)

[2.1.3 Functional Requirements by Liang Dingxuan 22](#_Toc174223235)

[2.1.4 Functional Requirements by Ng Kai Huat Jason 25](#_Toc174223236)

[2.1.5 Functional Requirements by Tan Guo Zhi Kelvin 31](#_Toc174223237)

[2.2 Non-Functional Requirements 35](#_Toc174223238)

[2.2.1 Non-Functional Requirements by Yeo Jin Rong 35](#_Toc174223239)

[2.2.2 Non-Functional Requirements by Ng Jing Zhan Garrett 40](#_Toc174223240)

[2.2.3 Non-Functional Requirements by Liang Dingxuan 43](#_Toc174223241)

[2.2.4 Non-Functional Requirements by Ng Kai Huat Jason 46](#_Toc174223242)

[2.2.5 Non-Functional Requirements by Tan Guo Zhi Kelvin 49](#_Toc174223243)

[3. Use Case Model 52](#_Toc174223244)

[3.1 Use Case Diagram 52](#_Toc174223245)

[3.2 Use Case Descriptions 53](#_Toc174223246)

[3.2.1 Use Case Descriptions by Yeo Jin Rong 53](#_Toc174223247)

[3.2.2 Use Case Descriptions by Ng Jing Zhan Garrett 56](#_Toc174223248)

[3.2.3 Use Case Descriptions by Liang Dingxuan 59](#_Toc174223249)

[3.2.4 Use Case Descriptions by Ng Kai Huat Jason 61](#_Toc174223250)

[3.2.5 Use Case Descriptions by Tan Guo Zhi Kelvin 64](#_Toc174223251)

[3.3 Use Case Specifications 66](#_Toc174223252)

[3.3.1 Use Case Specification by Yeo Jin Rong 66](#_Toc174223253)

[3.3.2 Use Case Specification by Ng Jing Zhan Garrett 84](#_Toc174223254)

[3.3.3 Use Case Specification by Liang Dingxuan 94](#_Toc174223255)

[3.3.4 Use Case Specification by Ng Kai Huat Jason 106](#_Toc174223256)

[3.3.5 Use Case Specification by Tan Guo Zhi Kelvin 117](#_Toc174223257)

[4. Class Model 126](#_Toc174223258)

[4.1 Design Class Diagram 126](#_Toc174223259)

[4.2 Classes 128](#_Toc174223260)

[5. Interaction Model 160](#_Toc174223261)

[5.1 Sequence Diagram by Yeo Jin Rong 160](#_Toc174223262)

[5.2 Sequence Diagram by Ng Jing Zhan Garrett 162](#_Toc174223263)

[5.3 Sequence Diagram by Liang Dingxuan 164](#_Toc174223264)

[5.4 Sequence Diagram by Ng Kai Huat Jason 165](#_Toc174223265)

[5.5 Sequence Diagram by Tan Guo Zhi Kelvin 166](#_Toc174223266)

[Appendices 167](#_Toc174223267)

[Appendix A – Other Tools Used 167](#_Toc174223268)

# 1. Introduction

## Purpose

The purpose of the iCar Rental platform is to create a seamless, user-friendly, and secure car-sharing marketplace that connects car owners with potential renters. By providing a diverse fleet of vehicles and offering flexible rental options set by Car Owners, iCar aims to meet the varied needs and budgets of its user.

## Background

Describes the background of the project and the client profile. You can include all the stakeholders of the project here.

Platform is conceived in response to the demand for flexible and cost-effective transportation solutions. Traditional car rental services/transportation services have rigid pricing structures, limited vehicle choice and inconvenient, inflexible renting processes. Recognizing these flaws in current available systems, iCar rental platform aims to revolutionize the car rental market by leveraging on ideas and inspirations from popularized platforms such as ‘Uber’ and ‘Grab’. Our platform empowers the user one step further, instead of just ridesharing, now renters can fully rent the vehicle for their own usage.

Our stakeholders are:

1) Car Owners:

**Individual Car Owners:** Private individuals who own one or more vehicles and are looking for additional sources of income passively. These owners benefit from having complete control over rental rates, availability and the ability to manage bookings.

**Small Business Owners:** Small businesses with a fleet of vehicles that are underutilized can list them on the iCar platform to generate additional revenue. These owners can benefit from the ease of managing the bookings of multiple vehicles and tracking their earnings.

2) Renters:

**Regular Renters:** Individuals who need a car for short-term use, for example: a day trip, weekend getaway or occasional errands or emergencies. These renters appreciate the flexibility, variety of vehicle options and ease of booking offered by the iCar platform, which is suited to their needs.

**Prime Renters:** Individuals who are more committed to renting cars with a minimum of $300 per month. Prime renters enjoy additional benefits such as 10% discount on monthly rental fees and 50% discount on roadside assistance, making the platform more attractive for users that have recurring needs

3) iCar Administrators:

**Platform Administrators:** Responsible for overseeing the operations of the iCar platform, such as verification of renters, managing listings as well as ensuring the overall safety and quality of the rental experience

## Scope

Describes the scope of the project, what is included and what is NOT included in the development of the project.

**Included in the Development:**

* + **User Registration and Verification:**
    - Development of a registration system for car owners and renters.
    - Implementation of a verification process for renters, including driver's licence validation and background checks.
  + **Car Listing and Management:**
    - Interface for car owners to list vehicles with detailed information (make, model, year, mileage, photos).
    - Tools for car owners to manage bookings, set rental rates, manage availability schedules, and track earnings through a dashboard.
    - Flexibility for car owners to withdraw their vehicles from the platform at any time.
  + **Search and Booking System:**
    - Advanced search functionality for renters to filter vehicles based on preferences such as car type, price range, location, and availability.
    - User-friendly booking system allowing renters to reserve vehicles by specifying the start and end date & time and the chosen car's brand and model.
  + **Payment Integration:**
    - Secure online payment system supporting multiple payment methods, including credit/debit cards and digital wallets.
    - Option for renter to enter discount code.
    - Automation of payment processing and financial transactions between renters and car owners.
  + **Pick-up and Return Management:**
    - Options for renters to choose pick-up and return locations, including iCar stations and home delivery/return services.
    - Implementation of additional fees for delivery and off-location returns.
  + **Prime Membership Features:**
    - Development of a Prime membership system with minimum rental requirements and benefits, including discounts on rental fees and roadside assistance.
  + **Safety and Quality Assurance:**
    - Pre-rental inspection, with mandated inspection reports to ensure all listed vehicles meet safety and quality standards.
    - Integration with third-party insurance providers by Car Owners to offer comprehensive coverage for rentals.

**NOT Included in the Development:**

* **Vehicle Maintenance Services:**
* Direct vehicle maintenance and repair services are not provided by the platform. Car owners are responsible for maintaining their vehicles.
* **Third-Party Insurance Policy Management:**
* Management and provision of insurance policies are handled by partnered third-party insurance companies, not directly by the iCar rental platform.
* **Legal and Regulatory Compliance:**
* The development scope does not cover the legal advisory services for compliance with local, state, or national regulations regarding car rentals. Stakeholders are expected to seek legal guidance independently to ensure compliance.

## Definitions, Acronyms, and Abbreviations

Describes any specialized terms used.

Within any organisation there will be terminology that is understood by the people working in a particular area, and often this terminology is very precise in conveying information. As a result, it is important that the requirements are able to use this terminology, but this can present a problem for the analysts and reviewers who lack familiarity with it. A glossary of terms overcomes this problem and provides a central source of terminology definitions.

**Car Owner:**  
An individual or business entity that lists their vehicle(s) on the iCar platform for rental purposes. Car owners can set rental rates, manage availability, and track earnings.

**Renter:**

An individual who registers on the iCar platform to rent vehicles. Renters must verify their eligibility and can choose from a variety of vehicles based on their preferences.

**Prime Renter:**

A category of renters who commit to renting cars worth a minimum of $300 per month. Prime renters receive benefits such as a 10% discount on monthly rental fees and a 50% discount on roadside assistance.

**Regular Renter:**

A category of renters without specific rental commitments or additional benefits. Regular renters follow the standard rental process and charges.

**User Registration Process:**

The process by which Users can register on iCar rental platform system by providing details such as Full Name, Contact No., DateOfBirth, Address, EmailAddress, Password. ContactNo and EmailAddress must be unique and valid.

**Car Registration Process:**

The process by which CarOwner(s) can register their cars on the iCar rental platform by providing details pertaining to their vehicle such as Vehicle, Registration Number, Make, Model, Year, Mileage, Photos, Fuel Type and RentalRate.   
  
**Email Verification Process:**

The process by which iCar rental platform system checks the validity of a user’s email by sending a unique 6 digit-code embedded within an email. The system waits for input of the corresponding 6 digit-code for email verification.

**Verification Process:**

The process by which iCar rental platform system checks the validity of a renter's driver's licence and conducts a background check to ensure the renter meets the necessary requirements to drive vehicles rented through the platform.

**Booking:**

The process through which a renter reserves a vehicle for a specified period, including the start and end date & time and the chosen car’s brand and model.

**Pick-up and Return Locations:**

Designated spots where renters can collect and return rented vehicles. These can be iCar stations or renter-specified locations for an additional fee.

**iCar Station:**

A designated location managed by iCar where renters can pick up or return vehicles. These stations are conveniently located throughout the city.

**Insurance Coverage:**

A comprehensive insurance policy provided through a third-party car insurance company, covering both car owners and renters against unforeseen accidents or damage during the rental period.

**Rental Rate:**

The price set by the car owner for renting their vehicle. This can vary based on factors such as vehicle type, duration of rental, and market demand.

**Availability Schedule:**

The schedule managed by administrators on the iCar rental platform indicates when their vehicle is available for rent.

**Payment Methods:**

Various options provided by the iCar rental platform for renters to pay for their bookings, including credit/debit cards and digital wallets. Discount codes can be applied to reduce rental cost.

**Background Check:**

A process undertaken by iCar to verify a renter’s history and ensure they meet the platform's safety and eligibility standards. This includes checking their personal information is legitimate, as well as verifying email and contact number are unique and legitimate

**Driver’s Licence Validation:**

A process undertaken by iCar to verify a renter’s driver's licence and ensure that it meets the eligibility criteria, such as their driver’s licence not being voided, and the driver’s licence is not a probationary licence (no longer p-plate).

**Inspection Protocol:**

Procedures followed by iCar to check the safety and quality of vehicles before each rental to ensure they meet required standards.

**Reservation Modification:**

The ability for renters to change the details of their booking, such as start and end dates or vehicle selection, through the iCar platform.

**Late Return Penalty:**

A fee charged to renters who return the vehicle later than the specified end date and time of the original booking.

**Home Delivery/Return Service:**

An additional service offered by iCar where the rented vehicle is delivered to or picked up from the renter’s specified location for an extra fee.

## References

Describes any references made to external sources such as textbooks, websites, software or technologies… etc. You can read the Library’s resources on how to cite sources at <https://np-sg.libguides.com/c.php?g=846507&p=6051941>

*Getting a driving licence & Learner driver rules in Singapore - SingaporeLegalAdvice.com*. (2024, February 27). SingaporeLegalAdvice.com.

https://singaporelegaladvice.com/law-articles/what-would-happen-if-a-learner-driver-commits-a-traffic-offence-while-learning/#:~:text=You%20will%20be%20required%20to,licence%20for%20the%20subsequent%20time

*LTA | Standard Registration*. (n.d.). https://onemotoring.lta.gov.sg/content/onemotoring/home/buying/vehicle-types-and-registrations/car/standard\_registration.html

# 2. Requirements

## 2.1 Functional Requirements

Write ***the functional requirements of the project.***

### 2.1.1 Functional Requirements by Yeo Jin Rong

Student Name: Yeo Jin Rong

Student ID: S10258457

**Functional Requirements for "Register as a Renter"**

When a user selects to register as a Renter, the system first displays a form for inputting personal information, including full name, date of birth, contact number, address, and email address. The user provides these details, which the system captures and records. The user is then prompted to confirm that the entered information is correct. If confirmed, the system checks for duplicate entries in the database, ensuring the contact number and email address are unique.

Next, the system prompts the user to choose a membership type, either Prime or Regular. Upon selecting Regular membership, the user is asked to confirm this choice, with the system notifying them that membership cannot be changed until the end of the month. Once confirmed, the system displays a form for entering driver’s licence information, including the number of demerit points, class of licence, licence ID number, date of issue, and certificate of merit eligibility. The user must confirm the accuracy of this information, after which the system validates it using an external API to ensure it is legitimate and non-duplicate.

The user is then presented with the Terms and Conditions, which they must accept to proceed. Following acceptance, the system marks the registration status as "Pending Email Verification" and sends a verification email containing a six-digit code. The user must enter this code to verify their email address. Upon successful verification, the system notifies the admin to review the registration. The admin conducts a background check and licence validation before either approving or rejecting the registration. The system notifies the user of their registration status, and if approved, grants full access to the platform. If rejected, the system informs the user of the reasons and possible steps to take.

**Input:**

1. **Personal Information:**
   * Full Name
   * Date of Birth
   * Contact Number
   * Address
   * Zip code
   * Email Address
2. **Membership Type Selection:**
   * Choice between Prime or Regular Membership
3. **Driver’s Licence Information:**
   * Number of Demerit Points
   * Class of Licence
   * Licence ID Number
   * Date of Issue
   * Certificate of Merit Eligibility
4. **Verification Code:**
   * Six-digit code received via email

**Output:**

1. **Captured and Recorded User Information:**
   * Full Name
   * Date of Birth
   * Contact Number
   * Address
   * Zip code
   * Email Address
2. **Confirmation of Information:**
   * Notification if the entered information is correct.
   * Confirmation of unique contact number and email address.
3. **Membership Confirmation:**
   * Notification about the non-changeable nature of membership until the end of the month.
4. **Driver's Licence Confirmation:**
   * Prompt for the user to confirm the accuracy of entered driver's licence information.
   * Validation of driver's licence details using an external API.
5. **Terms and Conditions Acceptance:**
   * User’s acceptance recorded in the system.
6. **Verification Email:**
   * Email sent to the user with a six-digit verification code.
   * Update of registration status to "Pending Email Verification."
7. **Admin Notification and Review:**
   * Notification sent to admin for registration review.
8. **Registration Status Notification:**
   * Notification sent to the user about the registration status (Approved/Rejected).
   * If approved, full access to the platform is granted.
   * If rejected, reasons for rejection and next steps are provided.

**Process:**

1. **Display Registration Form:**
   * System displays a form for inputting personal information.
2. **Capture and Confirm Personal Information:**
   * User provides personal details, confirms accuracy, and checks for duplicates.
3. **Membership Type Selection:**
   * User selects Prime or Regular membership.
4. **Driver's Licence Information Input:**
   * User enters driver's licence details.
5. **Validate Driver's Licence Information:**
   * System validates licence details using an external API.
6. **Acceptance of Terms and Conditions:**
   * User accepts terms and conditions.
7. **Verification Email Sent:**
   * System sends a verification email to the user.
8. **Email Verification:**
   * User verifies their email address using the received code.
9. **Admin Review:**
   * Admin conducts background check and licence validation.
10. **Registration Status Notification:**
    * System notifies the user of their registration status.

**Data Recorded:**

* **User's Personal Information:**
  + Full Name
  + Date of Birth
  + Contact Number
  + Address
  + Email Address
* **Membership Type:**
  + Prime or Regular
* **Driver's Licence Details:**
  + Number of Demerit Points
  + Class of Licence
  + Licence ID Number
  + Date of Issue
  + Certificate of Merit Eligibility
* **Verification Information:**
  + Verification Code
  + Email Status (verified/unverified)

**Functional Requirements for "Verify Renter"**

An iCar Platform administrator conducts background and driver's licence checks separately based on the personal and driver's licence information submitted during registration. Initially, the system conducts a background check to assess the eligibility of the Renter. This check involves verifying information such as identity and criminal background pertaining to traffic violations. Following this, the system conducts a driver's licence check to ensure that the Renter possesses a valid and up-to-date driver's licence. Upon completion of each check, the system updates the verification status of the Renter in the database to either "Verified" or "Rejected." Subsequently, the system notifies the Renter about their verification status. Recorded in the database are the results of both the background and driver's licence checks, ensuring transparency and accuracy in the verification process.

**Input:**

* Renter's Personal Information and Driver's Licence information, taken from record created in Registration

**Output:**

* Renter’s verification status updated
* Notification sent to the Renter about their verification status

**Process:**

1. **Conduct Background Check:**
   * An iCar Platform administrator verifies the Renter's identity, any criminal background pertaining to traffic violations, through a background check, either through verification through Land Transport Authority system, Singapore Police Force and/or other governmental bodies.
2. **Conduct Driver's Licence Check:**
   * An iCar Platform administrator verifies the validity and authenticity of the Renter's driver's licence, by cross-referencing driver licence information with Land Transport Authority of Singapore or other relevant governmental bodies
3. **Update Verification Status:**
   * Based on the results of the background and driver's licence checks, the system updates the verification status of the Renter in the database to either "Verified" or "Rejected."
4. **Notify Renter:**
   * Once the verification status is updated, the system sends a notification to the Renter informing them about their verification status.

**Data Recorded:**

* Verification status of the Renter is recorded in the database.
* Results of the background check (e.g., identity verification, criminal history).
* Results of the driver's licence check (e.g., validity of the licence).

### 2.1.2 Functional Requirements by Ng Jing Zhan Garrett

Student Name: Ng Jing Zhan Garret

Student ID: S10257347  
 **Functional Requirements for "Make Payment"**

The system shall provide a payment function to allow users to make payment for services on the iCar Platform, including but not limited to paying a rental fee, and settling penalties.

When a renter chooses to make payment, the system first checks whether the renter is a ‘prime’ renter, updating the total fee accordingly. Optionally, renters will also have the option to enter a discount code, and the discounted cost will be automatically reflected on the cost summary. The system then prompts the renter to select their preferred payment method, displaying the various options for payment through debit/credit card, digital wallet and PayNow options.

Once the payment method is chosen, the system must be able to prompt for the necessary payment details, allowing the user to input card payment details such as card number, name on card, expiry date and CVV for debit/credit card payment and the system will also prompt for a billing address. System will allow for digital wallet payment processing and also allow for PayNow payments as well.

Renter will review the payment information and confirm payment. The system will validate the entered information to ensure it meets the required format. Once validated, the system will process the payment by transmitting and communicating with the selected payment gateway.

Upon payment verification, the system should update payment status to "confirmed". System displays a message confirming successful payment and booking reservation. Furthermore, booking confirmation emails will be sent to the renter and the car owner in the case of a rental fee payment. An invoice will also be sent to the renter. If payment fails, an error message will be displayed, and the user will be prompted to enter their payment details again or choose an alternative payment method.

**Input**:

1. **Discount Code**
   * Optional Code to be inputted (6-character string)
2. **Payment method**
   * Credit Card/Debit Card
   * PayPal
   * Digital Wallet
3. **Payment Details (Credit/Debit card)**
   * Card Number (16-digit code)
   * Name on card
   * Expiry Date
   * CVV (3-digit code)
   * Billing Address

**Output:**

1. **Discount code validation**
   * Discount applied to cost summary and message notifies user of successful discount code usage
2. **Payment success message**
   * Message confirms successful payment
3. **Payment Confirmation**
   * Payment status updated to ‘confirmed’
4. **Booking confirmation email (Rental Fee Payment)**
   * Booking summary is sent to renter and vehicle owner.
5. **Invoice**
   * Invoice is sent to the renter through email

**Process:**

1. **System checks for renter status**
   * System applies 'prime' discount to cost summary if applicable
2. **Renter inputs discount code**
   * User optionally applies a discount code, granting a discount that is updated in cost summary
3. **Renter selects payment method**
   * User selects payment method and inputs required payment details.
4. **Renter confirms payment**
   * Renter confirms to make payment after checking all the necessary fields have been inputted correctly
5. **Validate Payment Confirmation:**
   * System transmits billing information to respective payment gateway, to process payment.
6. **Booking Confirmation Email Sent:**
   * System sends a booking confirmation details to the renter and car owner via email
7. **Invoice Sent:**
   * System sends an invoice of confirmed payment to the renter via email

**Data Recorded:**

1. **New Payment Record:**
   * Payment record with amount and payment method is generated and payment status is marked as "confirmed"

**Functional Requirements for "Review Vehicle"**

The system shall provide an option for the renters to provide feedback on their rental experience. When renter chooses to provide feedback, the system displays past rentals that the renter has made. Renter is able to choose which experience to give feedback on. The system will then validate that the renter has passed the rental period for the vehicle.

The system will then prompt the user to select a rating out of 5, and optionally the renter is able to write a description up till 200 words as well. System validates the input, before recording it into the database, associating it with the given vehicle. The review section for the vehicle will be updated with the new feedback. System will display a confirmation message stating that the feedback has been saved and car owners are notified via email that the review has been posted.

Alternatively, after renter returns a car, they will be prompted to leave a review about the car they have returned, moving straight to the review page.

**Input:**

1. **Rating:**
   * 1–5-star rating scale
2. **Review Description:**
   * Optional text input with a maximum of 200 words
3. **Vehicle Information:**
   * Vehicle ID
4. **User Information:**
   * User ID
5. **Booking Information:**
   * Booking ID

**Output:**

1. **Confirmation Message:**
   * States successful submission of review
2. **Review Section Update:**
   * New review is displayed on the review section
3. **Notification to Car Owner:**
   * Car owner is emailed about the review made

**Process:**

1. **Display Past Rentals**
   * System displays rental history
2. **Selected rental experience:**
   * Renter selects which rental experience to write a review on
3. **Validate rental period:**

* System validates that rental period has already passed

1. **System prompts for rating:**
   * Renter selects rating and optionally writes description for further feedback
2. **Updated Review Section:**
   * Review section is updated when renter confirms feedback
3. **Notification email sent:**
   * Car owner is notified of the review by the renter

**Data Recorded:**

1. **Review Details:**
   * Renter ID
   * Vehicle ID
   * Rating
   * Description
   * Datetime of review submission
2. **Details for identification** 
   * User ID
   * Booking ID
   * Vehicle ID

### 2.1.3 Functional Requirements by Liang Dingxuan

Student Name: Liang Dingxuan

Student ID: S10258272

**Functional Requirements for “Return Vehicle”**

The system for iCar must provide an option in the navigation menu for users to select the return vehicle option. The system then checks if the user has any vehicles that are currently booked, then the system outputs all vehicles for display and retrieves all vehicles booked by the user, displaying all vehicles on rent for the renter to select.

Users can choose a specific vehicle to return from the list of rented vehicles, with a maximum of 10 cars being displayed concurrently, and the system retrieves the booking details and displays the booking details. The system must then prompt the user to choose how they want to return the car, either to a designated iCar Station location or via delivery service. If the designated location is selected, the system must display all available iCar Station locations and prompt for a location selection. If delivery service is selected, the system must prompt the user to input the location for the delivery service. The system must then prompt the user for confirmation to return the car to account for any user errors. Finally, the System prompts for any damages and user must input any damage sustained during the rental period.

**Input:**

1. **Booking Information:**
   * Start date & time
   * End date & time
   * Brand & model of the vehicle
   * Booking ID
   * User ID
2. **Cost Calculation:**
   * Additional fees
3. **Return Car Information:**
   * Return method (designated location or pickup)
   * Return location
   * Damage sustained (if any)
   * Damage description (if any)
   * Cost of rental, accumulated from base rental and additional charges incurred from late fees

**Output:**

1. **Return Vehicle Confirmation:**
   * Confirmation of the vehicle return
2. **Total Charges Incurred**
   * Display and notify total charges

**Process:**

1. **Display vehicles**
   * System displays all cars currently being rented
2. **Select vehicle**
   * User selects vehicle to return
3. **Display iCar locations**
   * System displays designated iCar locations
4. **Selected method of return**
   * Renter selects preferred method of return (delivery service or drop off at selected iCar Station)
5. **Selected location**
   * Renter inputs location
6. **Input damages**
   * System must prompt for input of damages

**Data Recorded:**

1. **Cost Calculation:**
   * Additional fees
2. **Return Car Information:**
   * Return method (designated location or pickup)
   * Return location
   * Damage sustained (if any)

**Functional Requirements for “Pick up Vehicle”**

The system for iCar must prompt User for what type of Pick up they want, and User will then enter the option of pickup Designated iCar location or delivery Service. If the user selects pick up, System prompts user to enter pick up location and user has to input location. Else if user selects Designated iCar location System must display all designated iCar locations and user must select a location. After User selects location, system must display opening times

**Input:**

**1. Selected designated iCar station or delivery**

* + Type of pick up user has selected either iCar station pickup or delivery to location

**2. Location of designated iCar station or location of delivery**

* + Location of selected choice of pickup either pickup or delivery

**Output:**

1. **Designated iCar locations**
   * All the locations of iCar that could be the designated return station
   * Confirmation that renter has picked up designated vehicle

**Process:**

1. **Display iCar locations**
   * System displays designated iCar locations
2. **Selected method of pickup**
   * Renter selects preferred method of pickup
3. **Selected location**
   * Renter inputs location

**Data recorded:**

1. **Selected type of pickup user selected**
   * Type of pick-up user has selected.

(iCar Station Pickup / Delivery to Location)

1. **Location of pickup**
   * Location of selected choice of pickup either pickup or delivery
   * Date & Time of vehicle pick-up by Renter or when delivery is completed

### 2.1.4 Functional Requirements by Ng Kai Huat Jason

Student Name: Ng Kai Huat Jason

Student ID: S10262552

**Functional Requirements for "Register Vehicle"**

When Car Owner selects " **Add Vehicle**", the system will display a form to allow the input of the new car details which consists of the **Vehicle Registration Number**, **Make** , **Model**, **Maximum Seating Capacity, Year** of Manufacturing**, Mileage** in total kilometres travelled, any **Photos, Rental Rate** (by $/km)and **Fuel Type(Gas, Electric or Hybrid).** Based on the Fuel Type entered, the system will then prompt for additional information on the Fuel Type. For Gas Fuel Type, the system will prompt for information such as Preferred Petrol Kiosk, Petrol Type and Vehicle Maximum Fuel Capacity. For Electric Fuel Type, the system will prompt for information such as Preferred Charging Station, Vehicle Current Type, Vehicle Charging Rate and Vehicle Maximum Charging Capacity. For Hybrid Fuel Type, both the inputs for Electric Fuel Type and Gas Fuel Type will be prompted.

The system will prompt the user for confirmation to proceed to the next step. The system will verify that the car has not been registered within the database, being a unique and non-duplicate Vehicle Registration Number.

Lastly, the system will display a form to allow users to add an availability slot,  
This form will allow users to select the **Day**, **Start and** **End Time** in **24-hour format** for that day to be added to the car's Availability Schedule as a **List of Availability Slots.**

After all the inputs, the system will record all the information and store the details as a **Vehicle Object** and tie it to the respective user and update the Car Owner's display accordingly.

**Inputs:**

1. **Car Details:**
   * Vehicle Registration Number
   * Make
   * Model
   * Maximum Seating Capacity
   * Year
   * Mileage
   * Photos
   * Rental Rate ($/km)
   * Fuel Type (Gas, Electric, Hybrid)
2. **Fuel Type Information (as needed)**
   * Preferred Petrol Kiosk (E.g. Shell, Esso, Caltex, etc.)
   * Petrol Type (E.g. Shell FuelSave 95, Esso Synergy Supreme+, etc.)
   * Vehicle Maximum Fuel Tank Capacity
   * Preferred Charging Station (E.g. Shell Recharge, Bluecharge, CDG EngiePetrol)
   * Vehicle Current Type (AC or DC)
   * Vehicle Charging Rate (in kwH)
   * Vehicle Maximum Charging Capacity
3. **Availability Slots:**
   * Day of Week
   * Start Time (24-hour format)
   * End Time (24-hour format)

**Output:**

1. **Captured Vehicle Information:**
   * The system stores the new Vehicle details, including Vehicle Registration Number, Make, Model, Maximum Seating Capacity, Year, Mileage, Photos, Fuel Type, Rental Rate, and Availability Slots (as a List of Availability Slots called "Availability Schedule") and ties it to the respective Car Owner. The Car Owner is prompted for confirmation of adding vehicle to their account.
2. **Car Owner Dashboard Updated:**
   * The system updates the Car Owner's dashboard to display the newly registered vehicle, and any changes made.

**Process:**

1. **Access Add Vehicle:**
   * The Car Owner selects the "Add Vehicle" option on the iCar platform.
2. **Input Vehicle Details:**
   * The system displays a form for the Car Owner to input the new car details: Vehicle Registration Number, Make, Model, Year, Mileage, Photos, Rental Rate and Fuel Type.
3. **Input Fuel Type Details:**
   * The system displays a form for the Car Owner to input the Fuel Type information, based on what Fuel Type was provided.
     1. For Gas Fuel Type, input will be Preferred Petrol Kiosk, Petrol Type and Vehicle Maximum Fuel Capacity.
     2. For Electric Fuel Type, input will be Preferred Charging Station, Vehicle Current Type, Vehicle Charging Rate and Vehicle Maximum Charging Capacity.
     3. For Hybrid Gas Type, both Electric and Gas Fuel Type inputs will be prompted
4. **Confirm Car Details:**
   * The system prompts the Car Owner for confirmation to proceed to the next step after verifying the information.
5. **Input Availability Slots:**
   * The system displays a form for the Car Owner to input the availability slots, Start Day & Time, and End Date & Time in yyyy-mm-dd hh:mm (example: 2024-08-02 14:30)
6. **Store Car Information:**
   * The system records all the entered information and stores it as a Vehicle Object associated with the respective Car Owner.
7. **Update Dashboard:**
   * The system updates the Car Owner's dashboard to reflect the new vehicle, and any changes made.

**Data Recorded:**

1. **Vehicle Details:**
   * Vehicle Registration Number
   * Make
   * Model
   * Maximum Seating Capacity
   * Year
   * Mileage
   * Photos
   * Fuel Type
   * Rental Rate
2. **Availability Slot:**
   * Start Date & Time
   * End Date & Time
3. **A New Vehicle Record**

**Functional Requirements for “Manage Vehicles”**

When Car Owner selects “Manage Vehicles”, the system will retrieve all registered vehicles of the Car Owner and display them. The Car Owner can modify the **Mileage, Photos, Rental Rate, and Availability Slots** of the selected vehicle. The system will prompt for user confirmation before applying any changes.

**Inputs:**

1. **Car Details:**
   * Vehicle Registration Number
   * Mileage
   * Photos
   * Rental Rate
2. **Availability Slots:**
   * Start Date & Time (yyyy-mm-dd hh:mm format example: 2024-08-02 14:30
   * End Date & Time (yyyy-mm-dd hh:mm format example: 2024-08-02 14:30)

**Output:**

1. **Captured New Vehicle Information:**
   * The system stores the updated car details, including Mileage, Photos, Rental Rate, and Availability Slots (placed into a List of Availability Slots called "Vehicle Availability") and updates the respective vehicle.
2. **Car Owner Vehicles Update:**
   * The system updates the Car Owner's dashboard to display the updated vehicle information according to any changes made.
   * Confirmation of updated information for Car Owner

**Process:**

1. **Access Manage Vehicles:**
   * The Car Owner selects the "Manage Vehicles" option on the iCar platform.
2. **Retrieve Registered Vehicles:**
   * The system retrieves all registered vehicles of the Car Owner and displays them.
3. **Modify Vehicle Details:**
   * The Car Owner selects a vehicle and modifies the Mileage, Photos, Rental Rate, and Availability Slots.
   * The system prompts the Car Owner for confirmation before applying any changes.
4. **Store Updated Information:**
   * The system records all the updated information and updates the associated Vehicle under respective Car Owner.
5. **Update Dashboard:**
   * The system updates the Car Owner's dashboard to reflect the updated vehicle information, and any changes made.

**Data Recorded:**

1. **Updated Car Details:**
   * Mileage
   * Photos
   * Fuel Type
   * Rental Rate
2. **Updated Availability Slot:**
   * Start Date & Time
   * End Date & Time

**Functional Requirements for “Delete Vehicle”**

When Car Owner selects “Delete Vehicle”, they will be prompted to input the **Vehicle Registration Number** of the vehicle they want to delete. The system will than retrieve the respective vehicle record and prompt a user confirmation before deleting the vehicle. The system will than update the display accordingly.

**Input:**

1. **Vehicle Registration Number**

**Output:**

1. **Vehicle Record Deleted:**
   * The system will delete the respective vehicle record from the database.
2. **Car Owner Vehicle Updated:**

* The system updates the Car Owner's dashboard to reflect the deletion.
* Confirmation of updated information for the Car Owner.

**Process:**

1. **Access Delete Vehicle:**
   * The Car Owner selects the "Delete Vehicle" option on the iCar platform.
2. **Retrieve Registered Vehicles:**
   * The system retrieves all registered vehicles of the Car Owner and displays them.
3. **Select Vehicle to Delete:**
   * The Car Owner selects the vehicle they wish to delete.
   * The system prompts the Car Owner to input the Vehicle Registration Number of the vehicle they want to delete.
4. **Confirm Deletion:**
   * The system retrieves the respective vehicle record based on the entered Vehicle Registration Number.
   * The system prompts the Car Owner for confirmation before deleting the vehicle.
5. **Delete Vehicle:**
   * Upon confirmation, the system deletes the vehicle record from the database.
6. **Update Dashboard:**
   * The system updates the Car Owner's dashboard to reflect the deletion.
   * The Car Owner is notified of the successful deletion.

**Data Recorded:**

**Null**

### 2.1.5 Functional Requirements by Tan Guo Zhi Kelvin

Student Name: Tan Guo Zhi Kelvin

Student ID: S10262567

**Functional Requirements for "Book Vehicle"**

When a user decides to reserve a vehicle as a Renter on the iCar rental platform, the process begins after the user has browse a car from the available listings. The user specifies the start and end date & time of the booking period, and the brand & model of the chosen car. The system captures this booking information, checks the availability of the car for the specified period, and prepares the details for user validation, including the option to choose pickup at an iCar station or doorstep delivery.

The system then computes the total rental cost by processing the booking details and applicable rates, displaying a detailed summary with a cost breakdown. The user is prompted to review and confirm the booking details. The system presents the "Booking Terms and Conditions" to the User, requiring confirmation to ensure they agree to the terms before proceeding. Once confirmed, the system finalizes the booking, marking it as confirmed and ready for payment processing in a separate step.

**Input:**

1. **Booking information selection**
   * Start date & time of the booking period
   * End date & time of the booking period
   * Brand & model of the chosen car
   * Pickup or Delivery

**Output:**

1. **Cost calculation**
   * Detailed booking summary
   * Cost breakdown
   * Confirmation message for Renter to signify successful reservation

**Process:**

1. **Specify Booking Details**

* User selects the start and end date & time of the booking period.
* User selects the brand & model of the chosen car.
* User selects the pickup or delivery option.

1. **Capture Booking Information**

* The system captures the specified booking details.
* The system checks the availability of the car for the specified period.

1. **Prepare Details for Validation**

* The system prepares the booking details for user validation.
* The system includes the option for pickup at an iCar station or doorstep delivery.

1. **Compute Total Rental Cost**

* The system calculates the total rental cost based on booking details and applicable rates.
* The system displays a detailed summary with a cost breakdown.

1. **Review and Confirm Booking**

* The user reviews the booking summary and cost breakdown.
* The system presents the "Booking Terms and Conditions" to the user.
* The user confirms acceptance of the terms and conditions.

1. **Finalize Booking**

* The system marks the booking as confirmed.
* The system prepares the booking for payment processing in a separate step.

**Data Recorded:**

1. **Captured booking information**
   * Booking dates and times
   * Car specification
   * Pickup or Delivery selection
2. **Booking Terms and Conditions acceptance**
   * User’s acceptance recorded in the system
3. **New Rental Record**

**Functional Requirements for "Manage Booking"**

When a Renter decides to manage their current bookings on the iCar rental platform, the system allows them to modify their reservations. Reservations have a 24-hour grace window where booking(s) are non-cancellable and non-refundable. The Renter inputs the booking ID and the updated booking details. The system captures these changes, records them in the database, and updates the Renter's booking history accordingly.

**Input:**

1. **Booking ID**
2. **Updated Booking Details**
   * Booking date
   * Availability slot
   * Locations (pick-up of vehicle and return of vehicle)

**Output:**

1. **Booking Updates**

* Booking details are updated in the database.
* Notifications are sent to the Renter and the Car Owner regarding the booking changes.

**Process:**

1. **Enter Booking ID**

* The Renter inputs the booking ID for the booking they wish to modify.

1. **Specify Updated Booking Details**

* The Renter inputs the updated booking details, including new booking dates, availability slot, and pick up and return locations.

1. **Capture Updated Information**

* The system captures the updated booking details provided by the Renter.

1. **Validate Updates**

* The system validates the updated booking details to ensure they are feasible and do not conflict with existing bookings.

1. **Update Booking in Database**

* The system updates the booking details in the database with the new information of the booking as requested.

1. **Send Notifications**

* The system sends notifications to the Renter and the Car Owner regarding the changes made to the booking.

**Data Recorded:**

1. **Locations (pick up and return)**
   * New booking dates and times
   * New car availability slot
   * New pick up or return locations

**Functional Requirements for “Delete Booking”**

When a Renter selects “Delete Booking”, they will be prompted to input the **Booking ID** of the booking they want to delete. The system will then retrieve the respective booking record and prompt for confirmation before deleting the booking. The system will then update and display accordingly.

**Input:**

1. **Booking ID**

**Output:**

1. **Booking Record Deleted:**
   * The system will delete the respective vehicle record from the database.

**Process:**

1. **Access Delete Booking:**
   * The Car Owner selects the "Delete Booking" option on the iCar platform
2. **Retrieve Registered Vehicles:**
   * The system retrieves and displays all booking records for the specific Renter.
3. **Select Booking to Delete:**
   * The Renter selects the booking they wish to delete.
   * The system prompts the Renter to input the Booking ID of the booking record they want to delete.
4. **Confirm Deletion:**
   * The system retrieves their respective booking record based on the entered Booking ID.
   * The system prompts the Renter for confirmation before deleting the booking.
5. **Delete Booking:**
   * Upon confirmation, the system deletes the booking record from the database.

**Data Recorded:**

* + **Null**

## 2.2 Non-Functional Requirements

Write the non-Functional Requirements for the project.

### 2.2.1 Non-Functional Requirements by Yeo Jin Rong

Student Name: Yeo Jin Rong

Student ID: S10258457

**Non-Functional Requirements for "Register as Renter"**

**1. Reliability:**

* iCar System should have minimal downtime to ensure continuous availability for registration, in the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
* iCar System able to handle high volumes of registration without performance degradation, with the capacity to handle 200 concurrent registrations and connections successfully with no errors

**2. Performance:**

* iCar System should validate personal and driver’s licence information within 10s-time frame to ensure smooth user experience
* Email verification and admin approval process able to be completed within a reasonable timeframe (<24h) to avoid user frustration
* iCar System should efficiently manage database queries, with no status 500 (database error) responses for registration to prevent bottlenecks during high traffic

**3. Supportability:**

* Maintainability:
  + iCar System code should be modular and well-documented to facilitate easy maintenance and updates.
  + Logs should be maintained for all registration and verification activities to aid in troubleshooting and auditing
* Testability:
  + iCar System should be designed to support automated testing, enabling regular and thorough testing without extensive manual intervention by developers
* Configurability:
  + iCar System should be scalable to handle the increasing number of users and data entries. It should also be able to downsize to reduce server costs.
  + iCar System should be able to accommodate future enhancements without major conflicts or requiring extraordinary amounts of bug fixing

**4. Security:**

* Data Protection:
  + iCar System should encrypt all personal and sensitive information, both in transit and at rest to ensure data privacy and security. NRIC especially is handled with care in accordance with Singapore’s Personal Data Protection Act 2012
* Resistance To Attacks:
  + iCar Systems should implement robust security measures such as firewalls, intrusion detection systems as well as encryption as stated above to protect against cyber-attacks and data-breaches. 256-AES is strongly recommended for encryption, Cloudflare for Distributed Denial-of-Service Protection.
* Ability to Detect Attacks:
  + Car System should have real-time monitoring and alerting mechanisms to alert critical staff such as key software engineers, to detect and respond to potential security threats hastily.

**5. Usability:**

* Registration process should be intuitive and easy for users to interpret without ambiguity, aligned with common user workflows to minimise learning curves or user error
* Measurable Objectives:
  + Project should aim for user success rate of 95% for completing registration without external assistance
  + Average time to complete registration process should be approximately 3 minutes
  + iCar System should ensure an error rate of less than 2% for data entry and validation
* Characteristics of User:
  + Designed for users with basic competency in computer skills in web navigation and familiarity with online forms and email.
* Tasks User Undertake:
  + Users should be able to complete tasks such as entering personal information, confirming data accuracy and completing email verification without ambiguity, having clear prompts and forms to signal for User Input
* Situational Factors:
  + Registration process should be accessible and responsive on a multitude of devices and popular browsers (Chrome, Safari, Edge, Firefox, Opera, Brave, etc.), as well as having proper UI/UX implementation to accommodate different user environments
* Acceptance Criteria:
  + iCar System should be tested with prototypes and user feedback to ensure it meets usability standards before full deployment

**Non-Functional Requirements for "Verify Renter"**

1. **Reliability:**  
   * The verification process must be reliable, ensuring consistent and accurate results for all renters. Consistent audit checks must be made by administrators to ensure no false verification of Users are made.
   * The iCar system should have minimal downtime to maintain continuous availability for verification tasks, ensuring renters can proceed without delays. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
2. **Performance:**
   * Background and driver's licence checks should be completed efficiently within a reasonable timeframe to provide seamless user experience.
   * Notifications regarding verification status should be sent promptly to renters upon completion of the process to keep them informed.
3. **Supportability:**
   * The verification process should be well-documented with clear procedures and guidelines for administrators to follow, with protocols to enact once discrepancies are found, such as police reports being made if identity theft or fraud is detected.
   * Regular audits and reviews should be conducted to ensure accuracy and reliability, with any necessary updates implemented promptly.
   * iCar system must be able to run and accommodate different operating systems (Windows, MacOS, Linux, Android, iOS) and various devices ranging from desktops to laptops as well as mobile devices.
4. **Security:**
   * User data collected during verification must be securely stored and protected to maintain confidentiality and prevent unauthorised access.
   * Robust security measures should be implemented to safeguard against data breaches or cyber-attacks, ensuring the integrity of renter information.
5. **Usability:**
   * The verification process should be user-friendly and intuitive, having no ambiguity for prompting of user input, with clear instructions and minimal steps required.
   * Notifications sent to renters about their verification status should be action-oriented, providing clear guidance on next steps for accessing the iCar platform.

### 2.2.2 Non-Functional Requirements by Ng Jing Zhan Garrett

Student Name: Ng Jing Zhan Garrett

Student ID: S10257347

**Non-Functional Requirements for "Make Payment"**

1. **Reliability:**

* Availability:
  + iCar system should have minimal downtime on payment processing. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
* Data consistency:
  + Payment transactions and booking information must be processed and updated accurately, ensuring data integrity and avoiding errors in booking confirmation. Acceptable error rate is 0.01% of all payment transactions.

1. **Performance:**

* Payment processing speed:
  + Payment processing should be completed within 5 secs after confirmation, alleviating renter anxiety and concern.
* System response time:
  + Page should respond to the user within 2 secs to provide a smooth user experience and avoid frustration.
* Scalability:
  + iCar system should be scalable to accommodate increased user and transaction volume to ensure smooth operations during payment processing.
* Email delivery time:
  + System should be able to send the confirmation of booking details to the renter and car owner’s email within 15 - 30 secs after payment processing.

1. **Supportability:**

* Maintainability:
  + Code for payment processing functions should be modular and well documented to facilitate troubleshooting and updates.
* Error logging:
  + iCar system should log errors related to payment processing for diagnosis and to identify potential problems for better problem resolution.

1. **Security:**

* Encryption:
  + All payment information should be encrypted, during storage and transmission to ensure payment information remains secure.
* Payment gateway integration:
  + iCar system should comply with both PCI - DSS standards and standards established by the chosen payment gateway providers to securely handle credit/debit card information.

1. **Usability:**

* Intuitive interface:
  + iCar payment page should be simple to navigate and no ambiguity when prompting for user input, with clear instructions and icons of payment methods as a guide.
* Responsiveness:
  + iCar payment page must be accessible and responsive to mobile and desktop view, while also supporting a multitude of browsers to accommodate renters' convenience.

**Non-Functional Requirements for "Review Vehicle" Use Case**

**1. Reliability:**

* Data integrity:
  + iCar system should store data that is related to feedback securely, preventing data corruption.

**2. Performance:**

* System response time:
  + iCar system should display the list of past rentals to the user within 2 secs requesting selection to provide a review.
* Feedback Submission:
  + iCar system should be able to store feedback and update the respective review section in a reasonable timeframe (30secs)

**3. Supportability:**

* Error logging:
  + iCar system should log errors related to providing of feedback for diagnosis and to identify potential problems for better problem resolution.

**4. Security:**

* Audit Trails:
* The iCar system should maintain comprehensive audit logs of all actions related to the reviews, enabling tracking and accountability.

**5. Usability:**

* Intuitive interface:
  + iCar review page should be intuitive to navigate, without ambiguity to allow renters to be able to provide feedback without much trouble.
* Responsiveness:
  + iCar review page must be accessible and responsive to mobile and desktop view, while also supporting a multitude of browsers (such as Google Chrome, Opera GX, Firefox, Brave, Safari, Microsoft Edge) to accommodate renters' convenience.

### 2.2.3 Non-Functional Requirements by Liang Dingxuan

Student Name: Liang Dingxuan

Student ID: S10258272

**Non-Functional Requirements for "Return Car"**

1. **Reliability:**
   * The system for iCar must be available for 99.99% of the time to allow for user access the return functionality when needed. In the event of system failure, downtime can be no longer than 3hr, with an overall 99.9% uptime rate. If system failure is encountered, instructions must be provided to the Renter(s) on protocols to follow.
2. **Performance:**
   * The system for iCar should be able to respond to user actions with a reasonable time frame, with a maximum of 0.5s delay from client to server.
   * The system for iCar should be able to support multiple users without significant slowing of the system, with a maximum of 0.5s delay from client to server.
3. **Supportability:**
   * The system for iCar must be compatible with multiple types of devices
4. **Security:**
   * Only User which have been authenticated by strong passwords should be able to use the program for iCar, for example the password must contain a lowercase letter, an uppercase letter, a number and a special case character (such as !,@,#,$).
   * Sensitive user data for iCar must be protected such as user pickup location by encrypting the data
5. **Usability:**
   * User interface for iCar should be user-friendly, having no ambiguity when prompted for user input, providing clear instructions so that there is no room for ambiguity

**Non-Functional Requirements for "Pick Up Vehicle"**

1. **Reliability**
   * The system for iCar must be highly reliable, ensuring minimal downtime and quick recovery in case of failures to support vehicle pick-up use case. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
2. **Performance**
   * The system for iCar should respond promptly to user actions, for any operation related to vehicle pick-up, with a maximum of 0.5s delay from client to server.
   * The system for iCar should be able to handle high volumes of simultaneous user requests, especially during peak hours, without degradation in performance. This entails to a concurrent connection capacity of up to 200 users.
3. **Supportability**
   * The system for iCar must support various device platforms, including smartphones, tablets, and desktops, ensuring a consistent experience across all devices.
   * The system for iCar must support various operating systems, including but not limited to Android, Windows, MacOS, Linux and iOS
   * The system for iCar should be designed to allow easy updates and maintenance without significant downtime, ensuring continuous improvement and adaptation to new requirements.
4. **Security**
   * Only authenticated users with strong passwords should be able to access the vehicle pick-up functionality.
   * Sensitive user data such as payment information and personal details must be encrypted and securely stored to prevent unauthorized access and data breaches.
5. **Usability**
   * The user interface for iCar should be intuitive, providing no ambiguity for users when prompted for input, guiding users seamlessly through the vehicle pick-up process.
   * The system for iCar should provide real-time feedback, such as confirmation messages after a vehicle is successfully picked up, and clear instructions at each step.
   * The system for iCar should offer personalized features, such as saving user preferences for vehicle types and pick-up locations, to enhance user convenience and satisfaction.

### 2.2.4 Non-Functional Requirements by Ng Kai Huat Jason

Student Name: Ng Kai Huat Jason

Student ID: S10262552

**Non-Functional Requirements for "Register Car"**

1. **Reliability:**
   * The iCar system should ensure high availability to allow car owners to register their cars at any time without interruptions, this will allow them to register their vehicles anywhere and at any time. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
   * The iCar system should maintain data integrity, ensuring that all car details are accurately saved and retrievable without data loss or corruption.
2. **Performance:**
   * The iCar system should process and store the car registration details within 5s to provide a smooth registration experience.
   * **The iCar system should update the Car Owner’s display accordingly to feature the any Vehicle(s) added promptly, the vehicles owned by the Car Owner being updated from the database within 5s.**
3. **Supportability:**
   * The iCar System should logged any car registration activities to aid in any troubleshooting and auditing.
   * The system for iCar must support various device platforms, including smartphones, tablets, and desktops, ensuring a consistent experience across all devices.
   * The system for iCar must support various operating systems, including but not limited to Android, Windows, MacOS, Linux and iOS
4. **Security:**
   * The iCar system should have vehicle information transmitted with end-to-end encryption to ensure that no vehicle information that can be used for User Identification such as Vehicle Registration Number being eavesdropped.
5. **Usability:**
   * The registration process should be accessible and responsive on various devices and popular browsers (Chrome, Safari, Edge, Firefox, etc.) to ensure that a broad range of Car Owners are able to interact with the iCar System.

**Non-Functional Requirements for "Manage Vehicles"**

1. **Reliability:**
   * The vehicle management feature must be available 24/7 with a target uptime of 99.9% to ensure car owners can manage their vehicles throughout the day
   * The iCar system must ensure that all modifications to vehicle data are accurately recorded and maintained without loss or corruption.
2. **Performance:**
   * The system should retrieve and display the list of registered vehicles within 3 seconds and apply any modifications within 5 seconds to ensure a smooth user experience.
3. **Supportability:**
   * The iCar System should log any management activities to allow for testing and auditing if necessary to prevent system errors.
   * The system for iCar must support various device platforms, including smartphones, tablets, and desktops, ensuring a consistent experience across all devices.
   * The system for iCar must support various operating systems, including but not limited to Android, Windows, MacOS, Linux and iOS
4. **Security:**
   * The iCar System should ensure that the respective Car Owners are navigated to their respective dashboards to prevent any unauthorized access or changes to other Car Owner’s dashboards.
5. **Usability:**
   * The vehicle management process should be user-friendly and intuitive, leaving no room for ambiguity when user is prompted for input, with clear instructions and prompts to minimize user errors.
   * The iCar System should ensure that any errors encountered during the modification process are communicated clearly to the Car Owner to prevent user frustration.
   * Ensure the vehicle management process is accessible and responsive on various devices (e.g., desktops, tablets, smartphones) and popular browsers (Chrome, Safari, Edge, Firefox), this is to ensure that Car Owners can manage their vehicles from anywhere.

**Non-Functional Requirements for "Delete Vehicle"**

**1. Reliability:**

* The vehicle management feature must be available 24/7 with a target uptime of 99.9% to ensure car owners can manage their vehicles throughout the day
* The iCar system must ensure that all modifications to vehicle data are accurately recorded and maintained without loss or corruption. Consistent and frequent backups must be created to store all vehicle information

**2. Performance:**

* The system should update and display the new list of registered vehicles within 5s of successful deletion
* The system should retrieve and display the list of registered vehicles within 3 seconds and apply any modifications within 5 seconds to ensure a smooth user experience.

**3. Supportability:**

* The iCar System should log any management activities to allow for testing and auditing if necessary to prevent system errors.

**4. Security:**

* The iCar System should ensure that the respective Car Owners are navigated to their respective dashboards to prevent any unauthorized access or changes to other Car Owner’s dashboards.
* To prevent accidental deletions, the iCar System must have confirmation dialog before any vehicle is removed from the system.

**5. Usability:**

* The vehicle deletion process should be user-friendly and intuitive, leaving no room for ambiguity when user is prompted for input, with clear instructions and prompts to minimize user errors.
* The process should be accessible and responsive on various devices (e.g, desktops, tablets, smartphones) and popular browsers (Chrome, Safari, Edge, Firefox, etc.).
* Any errors encountered during the deletion process should be communicated clearly to the Car Owner to prevent user frustration.

### 2.2.5 Non-Functional Requirements by Tan Guo Zhi Kelvin

Student Name: Tan Guo Zhi Kelvin

Student ID: S10262567

**Non-Functional Requirements for "Create Booking"**

1. **Reliability:**
   * The iCar booking system must maintain a high uptime to ensure uninterrupted access to booking functionalities. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
2. **Performance:**
   * The iCar system must respond to booking-related actions (e.g., specifying booking period, selecting car model) quickly to ensure a smooth and efficient booking experience, with a maximum of 0.5s delay from client to server.
   * The iCar system must be able to scale out to accommodate a total capacity of 1000 bookings for a Renter without performance issues, after exceeding capacity of 1000 bookings, the oldest booking will be automatically deleted and transferred to an archival database instead of the active database.
3. **Supportability:**
   * The iCar system must be designed for easy updates and maintenance, with minimized downtime (<1 hour) and clear, well-documented code.
4. **Security:**
   * Sensitive iCar booking data must be encrypted using industry-standard encryption (e.g., AES-256) during transmission and storage.
5. **Usability:**
   * The iCar booking interface must be user-friendly leaving no room for ambiguity on user prompts, enabling Renters to easily navigate, select car details, specify booking periods, and complete the booking process smoothly.

**Non-Functional Requirements for "Manage Booking"**

1. **Reliability:**
   * iCar system must ensure that any changes or cancellations are accurately recorded, and that the system can recover from failures such as database errors, without loss of booking management data. System must have routine backup(s) as a failsafe measure.
   * The iCar booking management functionality must be available most of the time, ensuring that users are able to manage their booking. In the event of system failure, downtime can be no longer than 6hr, with an overall 99.5% uptime rate.
2. **Performance:**
   * The iCar system must update booking status in real-time to reflect changes made by Renters.
   * The iCar system must respond to booking modification requests quickly, with a maximum of 0.5s delay from client to server.
3. **Supportability:**
   * The iCar system must provide a comprehensive audit log for booking management activities, tracking all changes and updates.
   * The iCar booking management functionality must support multiple updates for bookings, allowing Renters to modify multiple bookings simultaneously.
4. **Security:**
   * The iCar system must monitor booking management activity for suspicious behaviour and alert administrators of potential security incidents.
5. **Usability:**
   * The iCar booking management interface must be user-friendly leaving no room for ambuigity for user input, enabling Renters to easily navigate, modify booking details, and cancel bookings easily.
   * The iCar system must provide clear instructions and real-time feedback to guide Renters on how to modify their booking.

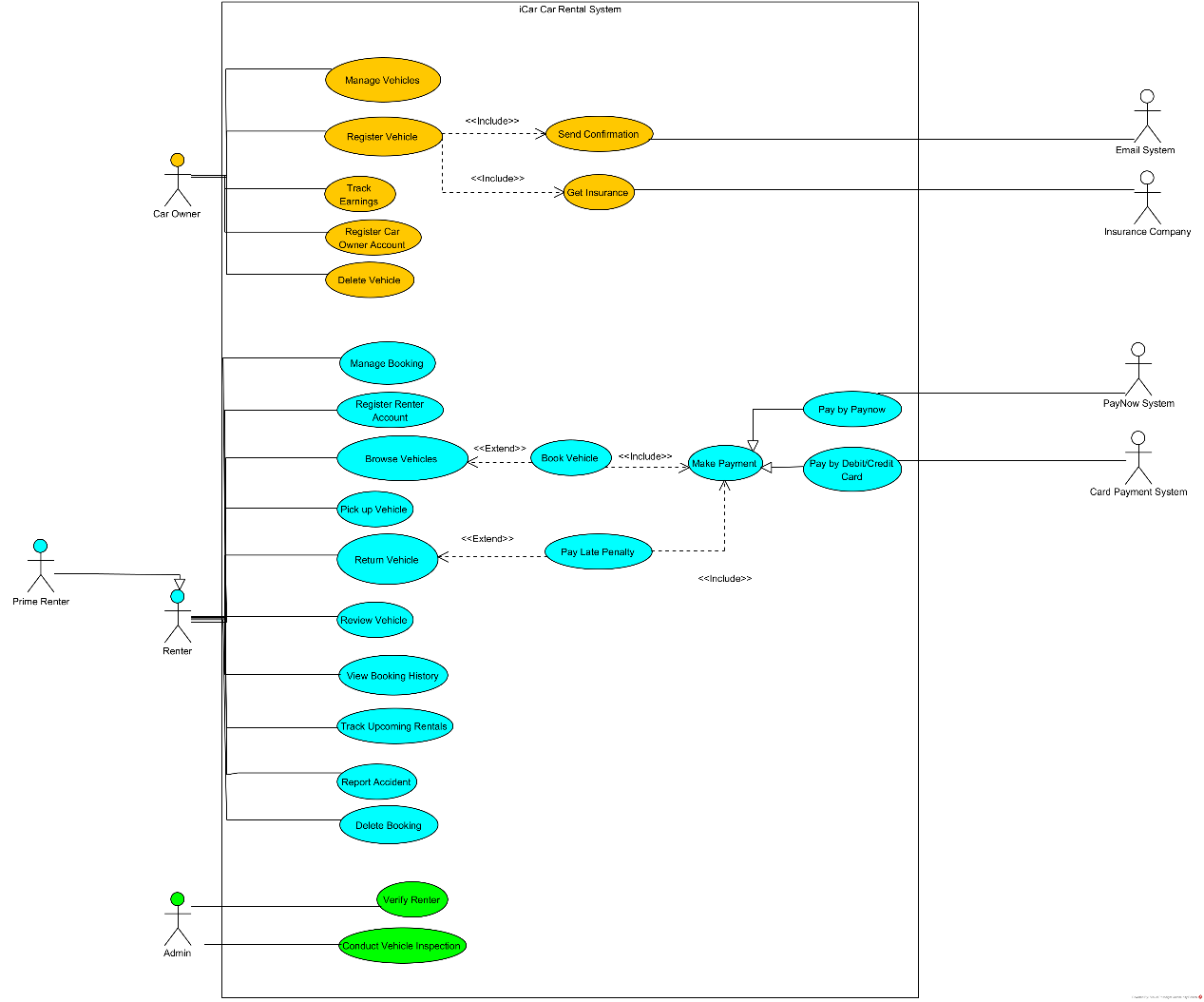
**Non-Functional Requirements for "Delete Booking"**

1. **Reliability:**
   * The iCar system must ensure that when a booking is deleted, the operation is reliably performed without any data inconsistency or loss. If a deletion process fails due to a system error, it must be automatically retried within 5 minutes, with error logs generated for manual review.
2. **Performance:**
   * The iCar system must process booking deletion requests within 0.3 seconds from client to server to ensure a swift user experience.
3. **Supportability:**
   * The iCar system must maintain an audit trail for all deletion operations, capturing details such as the user initiating the deletion, timestamp, and booking details, to ensure traceability and accountability.
4. **Security:**
   * The iCar system must verify the identity of the user before allowing the deletion of a booking record, ensuring that only authorized users can perform deletion.
5. **Usability:**
   * The iCar system must provide a confirmation prompt with clear information about the implications of deletion before finalizing the booking deletion, ensuring that users do not accidentally delete bookings records.

# 3. Use Case Model

## 3.1 Use Case Diagram

Draw your use case diagram.



## 3.2 Use Case Descriptions

ALL use cases as shown in your use case diagram are to be described here.   
The student’s name and id should be clearly indicated for each use case description section.

### 3.2.1 Use Case Descriptions by Yeo Jin Rong

Student Name: Yeo Jin Rong

Student ID: S10258457

**Use Case Description: Register as Renter**

This use case allows the User to register as a renter on the ICar platform provided that the user’s personal and driver’s licence information is valid, unique and available. This information is recorded and input into a database, creating an account for the User. The user’s registration status is updated to “Pending Email Verification” and the user is sent a verification email

* **Input:**
* User's personal information (Full Name, Contact Number, Date of Birth, Address, Email Address) and driver's licence information (Number of Demerit Points, Class of Licence, Licence ID Number, Date of Issue, Certificate of Merit Eligibility)
* **Condition:**
* User's personal and driver's licence information must be valid, unique, and available.
* **Recorded:**
* User's personal and driver's licence information is recorded in the database.
* **Output:**
* User's registration status is updated to "Pending Email Verification," and a verification email is sent.

**Use Case Description: Verify Renter**

This use case allows the Admin to verify the eligibility of users registering as renters on the iCar platform by conducting background and driver's licence checks. The system records the verification status of the Renter, to either "Verified" or "Rejected". Renter's verification status is updated, and Renter is notified on the outcome of the verification process.

* **Input:**
  + Renter ID
* **Condition:**
  + Renter has submitted personal and driver's licence information during registration.
  + Renter ID is valid and exists within the database
* **Recorded:**
  + The verification status of the Renter is recorded in the database.
* **Output:**
  + Renter’s verification status is updated, and the Renter is notified about their verification status.

**Use Case Description: View Booking History**

This use case allows the Renter to view their past bookings on the iCar platform and view the relevant booking information. Renters can input filters such as start date and end date to narrow down their Booking History

* **Input:**
  + Start Date and End Date if necessary
* **Condition:**
  + Renter must be registered and logged into the iCar platform and have made at least one booking.
* **Recorded:**
  + N/A
* **Output:**
  + Renter can view details of their past bookings.

**Use Case Description: Register Car Owner Account**This use case allows the User to register as a Car Owner on the iCar platform provided that the user’s personal information is valid, unique. The system records this information into a database, creating an account for the User. The user’s registration status is updated to “Pending Email Verification” and the user is sent a verification email

* **Input:**
  + User's personal information (Full Name, Contact Number, Date of Birth, Address, Email Address)
* **Condition:**
  + User's personal information must be valid
* **Recorded:**
  + User's personal information is recorded in the database.
  + A new Car Owner record is captured in the iCar System
* **Output:**
  + User's registration status is updated to "Pending Email Verification," and a verification email is sent.
  + A confirmation message is sent to the email address after the Car Owner record is successfully created

### 3.2.2 Use Case Descriptions by Ng Jing Zhan Garrett

Student Name: Ng Jing Zhan Garrett

Student ID: S10257347

**Use Case Description: Make Payment**

This use case allows the renter to make payments on the iCar rental platform. Renters can pay for their bookings using various payment methods, including credit/debit cards and digital wallets, or PayNow. They have the option to enter discount codes during the payment process. The system validates payment details, processes transactions through the payment gateway, and updates the reservation status upon successful payment. Both the renter and the car owner are informed of the successful booking for booking payment. The system records the booking and payment confirmation in its database.

* **Input:**
  + Optional Discount Code
* **Condition:** 
  + Renter has to make payment for rental fee or settle penalty fee.
* **Recorded:** 
  + A new payment record is captured in the iCar System
  + Status on payment record is recorded as "confirmed"
* **Output:** 
  + User receives booking confirmation and invoice.
  + Car owner is notified of the booking

**Use Case Description: Pay By Debit/Credit Card**

This specialized use case, derived from the "Make Payment" use case allows renters to make payment using debit/credit cards and their digital wallet. The system will handle the transaction through the payment gateway and updates the payment status to "confirmed".

* **Input:**
  + Card Number
  + Name on Card
  + Expiry Date
  + CVV
  + Billing Address
* **Condition:**
  + Renter has selected to make card payment.
* **Recorded:**
  + Status on payment record is recorded as "confirmed"
* **Output:**
  + N/A

**Use Case Description: Pay By PayNow**

This specialized use case, derived from the "Make Payment" use case allows renters to make payment using PayNow. The system will handle the transaction through the payment gateway and updates the payment status to "confirmed".

* **Input:**
  + N/A
* **Condition:**
  + Renter has selected to make PayNow payment.
* **Recorded:**
  + Status on payment record is recorded as "confirmed"
* **Output:**
  + N/A

**Use Case Description: Review Vehicle**

This use case allows renters to write a review for a vehicle after the rental period has ended. Feedback will include a rating out of 5 and an optional written review. The system will record the review associated to the vehicle and update the review section accordingly.

* **Input:**
* 1–5-star rating scale and Review Description with a maximum word count of 200
* **Condition:**
* Renter must be registered and logged into the iCar platform. The renter must have completed the rental period for the respective vehicle.
* **Recorded:**
* Review details are recorded in the database and associated to the respective vehicle
* **Output:**
* The review is added to the respective vehicle

**Use Case Description: Track Upcoming Rentals**

This use case allows the Renter to track the details of their upcoming rentals on the iCar platform. The details of the upcoming rentals will be displayed with relevant information, such as booking information, and allows renters to filter them by upcoming week and month.

* **Input:**
* Renter ID
* **Condition:**
* Renter must be registered and logged into the iCar platform and have at least one upcoming rental.
* **Recorded:**
* N/A
* **Output:**
* Details of upcoming rentals are displayed to the renter.

### 3.2.3 Use Case Descriptions by Liang Dingxuan

Student Name: Liang Dingxuan

Student ID: S10258272

**Use Case Description: Return car**

This use case allows the Renter to return a car as a renter on the iCar rental platform provided that the user has already booked a car, enabling them to return a car they have rented by choosing a designated iCar location to return it to or opt for pickup. The user then inputs damages if there is any and the system prompts extra payment if the car is returned late. After the car is returned the car owner is notified.

* **Input:**
  + User's return location
  + Renter ID
  + Booking ID
* **Condition:** 
  + **Use** must be logged in
  + User must have already rented a car
* **Recorded:** 
  + Car return date and time
* **Output:** 
  + User's booking is stated as complete in database

**Use Case Description: Pick up Vehicle**

This use case allows the Renter to pick up the rented vehicle from a designated location or arrange for delivery. The system

* **Input:**
  + Renter ID
  + Booking ID
* **Condition:**
  + BookingStatus is confirmed and active.
  + Renter must be registered and logged into the iCar platform.
  + Renter is opting to pick-up the Vehicle during the allocated booking period
* **Recorded:**
  + isPickedUp status in respective booking record is updated to "True"
  + Date and Time of successful Pick-Up
* **Output:**
  + Booking record is updated to record that vehicle has been picked up
  + Notifications are sent to the Renter and Car Owner confirming the pickup.

**Use Case Description: Report Accident**

This use case allows the Renter to report an accident that occurred during the rental period on the iCar platform. Users will input information pertaining to the accident such as accident date, time, location and description. The system will record the accident details.

* **Input:**
  + Accident details (date, time, location, description)
  + Renter ID
  + Booking ID
* **Condition:**
  + Renter must be registered and logged into the iCar platform.
  + An accident has occurred during the rental period and is reported
  + Renter must have an active booking
* **Recorded:**
  + Accident report details are recorded in the database.
  + A new accident record has been captured in the iCar System
* **Output:**
  + Message displayed for confirmation of submission of an accident

**Use Case Description: Conduct Vehicle Inspections**

This use case allows the Administrator to conduct thorough inspections of vehicles listed on the iCar platform to ensure they meet safety and quality standards. Admins will input information for inspection such as inspection date, name of inspector (usually the admin himself), details of inspection as well as the status of inspection as "Passed" or "Failed". The system records the inspection details.

* **Input:**
  + Inspection Details such as Inspection Date, Name of Inspector, Details of Inspection (such as what aspects have failed) and status of inspection.
* **Condition:** 
  + Vehicle is registered on the iCar platform, and inspection is scheduled or requested.
* **Recorded:** 
  + Inspection report(s) are recorded in the database.
  + Respective vehicle record is updated based on the inspection results
* **Output:** 
  + Message displayed for successful submission of an inspection

### 3.2.4 Use Case Descriptions by Ng Kai Huat Jason

Student Name: Ng Kai Huat Jason

Student ID: S10262552

**Use Case Description: Register Vehicle as Car Owner**

This use case allows the User to register a Vehicle on the iCar platform provided that the user is registered as "Car Owner" and the user's Car Details inclusive of the **Vehicle Registration Number, Make, Model, Year, Mileage, Photos, and Fuel Type (Gas, Electric, Hybrid)** are valid, unique and available. The system will also prompt the user for the **Rental Rate and Availability Slot.** This information is recorded and input into a database as a Vehicle Composite Object associated with the User.

* **Input:** 
  + Vehicle Registration Number, Make, Model, Year, Mileage, Photos, and Fuel Type (Gas, Electric, Hybrid), Rental Rate, Availability Slots
  + Fuel Type Information based on Fuel Type
    - If Gas, Preferred Petrol Kiosk, Petrol Type and Maximum Fuel Capacity
    - If Electric, Preferred Charging Station, Current Type, Charging Rate and Maximum Charging Capacity
    - If Hybrid, both gas and electric fuel type information will be inputted.
* **Condition**:
* Car Owner must be registered and logged into the iCar platform.
* User's Vehicle Details must be valid, unique and available.
* **Recorded:**
* User's Car Details are stored in the Vehicle object
* Date & Time car has been added to the iCar System
* A new car record is captured in the iCar System and added to the Car Owner
* **Output:**
* Message displayed for confirmation of submission of a new Vehicle

**Use Case Description: Manage Vehicles**

This use case allows the Car Owner to manage their listed vehicles on the iCar platform, searching by inputting Vehicle Registration Number, including updating vehicle details, setting availability, if necessary. The updates are recorded in the database and reflected in the Car Owner's dashboard.

* **Input:**
  + Vehicle Registration Number of a specific vehicle
* **Condition:**
  + Car Owner must be registered and logged into the iCar platform.
* **Recorded:**
  + Updated Vehicle details and availability settings are recorded in the database.
* **Output:**
  + Message displayed for successful update of a Vehicle

**Use Case Description: Delete Vehicle**

This use case allows the Car Owner to **delete** their listed vehicles on the iCar platform by searching the **Vehicle Registration Number** of a specific vehicle. Associated availability slots will also be deleted in conjunction with the vehicle. Any deletion will be recorded in the database and reflected in the Car Owner’s dashboard

* **Input:**
  + Vehicle Registration Number of a specific vehicle
* **Condition:**
  + Car Owner must be registered and logged into the iCar platform.
* **Recorded:**
  + Specified vehicle record is deleted from the database
  + Associated availability slot record is deleted from the database
* **Output:**
  + Message displayed for successful deletion of a Vehicle

**Use Case Description: Track Earnings**

This use case allows the Car Owner to track their earnings from rented vehicles through the iCar platform, input the Vehicle Registration Number of a specific vehicle if necessary. Earnings data is retrieved from the database and displayed to the Car Owner in a summarised and detailed format.

* **Input:**
  + Vehicle Registration Number of a specific vehicle if necessary
* **Condition:**
  + Car Owner must be registered and logged into the iCar platform and have at least one vehicle rented out.
* **Recorded:**
  + N/A
* **Output:**
  + Earnings report is displayed to the Car Owner.

### 3.2.5 Use Case Descriptions by Tan Guo Zhi Kelvin

Student Name: Tan Guo Zhi Kelvin

Student ID: S10262567

**Use Case Description:** **Create Booking for a Vehicle as Renter**

This use case allows the User to create a vehicle booking on the iCar platform, enabling them to rent a vehicle for a specified period and choose to pick up at a designated iCar station or opt for doorstep delivery. After viewing detailed vehicle information, the User specifies the start and end date & time of the booking period, and the brand & model of the chosen vehicle. The system then checks the availability of the vehicle for the specified period. If available, the User input and confirm their booking details, selects the pickup or delivery option, agrees to the booking terms and conditions, and receives a detailed cost summary to finalise the booking.

* **Input:** 
  + Booking details (e.g. start date & time, end date & time, and the make and model of chosen vehicle).
* **Condition:** 
  + User logged in as Renter.
  + Renter account status is “Active”.
* **Recorded:** 
  + User's car booking information.
  + A new Booking Record is added to the iCar System
* **Output:** 
  + System confirms the booking awaiting payment.

**Use Case Description: Browse Vehicles**

This use case allows the Renter to browse through available vehicles on the iCar platform. They can input filters to narrow down their search, such as car type, price range, location, availability. Vehicles will then be displayed with their respective vehicle information.

* **Input:**
  + Search criteria (car type, price range, location, availability)
* **Condition:**
  + Renter must be registered and logged into the iCar platform.
* **Recorded:**
  + N/A
* **Output:**
  + List of vehicles is displayed based on the Renter’s filters and preferences, with detailed information available for each vehicle.

**Use Case Description: Manage Booking**

This use case allows the Renter to manage their current bookings on the iCar platform, including modifying the reservation. The changes are recorded in the database and reflected in the Renter's booking history.

* **Input:**
  + Booking ID
  + Updated booking details (e.g. New BookingDate, New AvailabilitySlot, New Locations)
* **Condition:**
  + Renter must be registered and logged into the iCar platform.
  + Booking must be an upcoming booking
  + Booking must have more than a 24h period before commencement (that is a booking cannot be modified if there is less than 24hrs to its Start Time)
  + Booking cannot be modified more than 3 times
* **Recorded:**
  + Updated booking details are recorded in the database
* **Output:**
  + Booking details such as BookingDate, Availability Slot and Location are updated if necessary.
  + Notifications are sent to the Renter and Car Owner regarding the booking changes.

**Use Case Description: Delete Booking**

This use case allows the Renter to delete their booking on the iCar platform. The Renter provides the booking ID for the reservation they wish to delete. The system verifies the request, removes the booking record from the database, and updates the booking history accordingly.

* **Input:**
  + Booking ID
* **Condition:**
  + Renter must be registered and logged into the iCar platform.
* **Recorded:**
  + The booking record has been removed.
* **Output:**
  + The booking record is deleted in the database.

## 3.3 Use Case Specifications

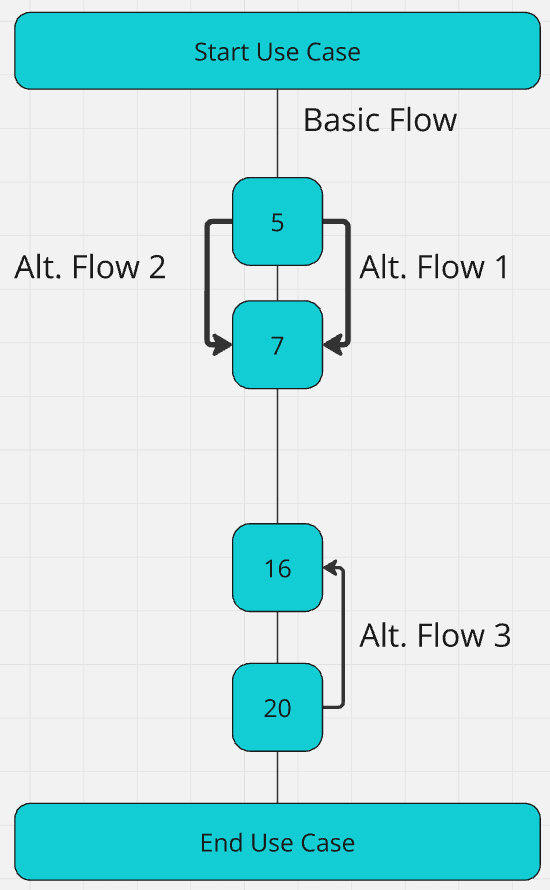
You should use the Use Case Specification template given in Appendix D of the assignment.   
The student’s name and id should be clearly indicated for each use case specification.   
Use Case Specification is only required for Assignment 2.

### 3.3.1 Use Case Specification by Yeo Jin Rong

Student Name: Yeo Jin Rong

Student ID: S10258457

|  |  |
| --- | --- |
| **Use Case ID** | UC01 – Jin Rong |
| **Use Case Name** | Register Vehicle |
| **Brief Description** | This use case allows a registered user with the role of "Car Owner" to register their car on the ICar platform. The process involves entering car details, uploading photos, providing insurance details, setting a rental rate, and selecting availability slots. The information is stored in the database as a Vehicle Object associated with the Car Owner. |
| **Actor(s)** | Car Owner (Primary)  iCar Platform System (Supporting)  LTA System (Supporting)  Insurance System (Supporting) |
| **Pre-condition(s)** | The user must be registered and logged in as a “Car Owner”  The user has all relevant information pertaining to the vehicle valid and ready for input, this is inclusive of Vehicle Details, Fuel Type Details, Photos, Insurance and Availability Slot(s) of the Vehicle |
| **Post-condition(s)** | The vehicle details, insurance information, rental rate and availability slots are validated and stored in database  The vehicle is registered on the iCar platform and associated with respective Car Owner  Car Owner is notified of successful registration |
| **Basic Flow** | 1. This use case starts when a Car Owner selects to register a new Vehicle 2. System prompts for vehicle information input 3. Car Owner inputs vehicle information    * **Vehicle Registration Number**    * **Make [String]**    * **Model [String]**    * **Rental Rate [Float]**    * **Maximum Seating Capacity [Int]**    * **Year (YYYY) [Date]**    * **Mileage (km) [String]** 4. System prompts for Fuel Type (Gas, Electric or Hybrid) 5. Car owner selects Gas Fuel Type 6. Car Owner inputs Fuel Type Information for Fuel Type Gas    * **Preferred Petrol Kiosk (Shell, Esso, Caltex, etc.) [String]**    * **Petrol Type (Shell FuelSave 95, Esso Synergy Supreme+, etc.) [String]**    * **Vehicle Maximum Fuel Tank Capacity [Float]** 7. System prompts for confirmation that vehicle information is correct 8. System validates uniqueness of Vehicle Registration Number and that all required fields are filled 9. System prompts Car Owner to upload Photos for Vehicle 10. Car Owner uploads Photo object 11. System prompts Car Owner to continue 12. Car Owner chooses to continue 13. System prompts for Insurance details for Vehicle 14. Car Owner inputs insurance information     * **Coverage Details**     * **Insurance Start Date**     * **Insurance End Date** 15. System validates insurance details are legitimate 16. System prompts Car Owner to select availability slot 17. Car Owner selects availability slot     * **Day of Week**     * **Start Time (24h format)**     * **End Time (24h format)** 18. System validates that availability slot input is valid 19. System prompts Car Owner to continue 20. Car Owner selects to continue 21. System stores record as a Vehicle Object, with composite objects VehicleAvailability and Insurance, associated with Car Owner 22. System displays Vehicle Information, that is all Vehicle Details, Photos, Fuel Type Information, Insurance Details and Availability Slots 23. System prompts Car Owner to confirm that information is correct 24. Car Owner chooses that information is correct 25. System prompts Car Owner to complete vehicle registration or register another vehicle 26. Car Owner selects complete vehicle registration 27. System notifies Car Owner of successful registration and lists all registered vehicles by Vehicle Register Number 28. This Use case ends   References:   * <https://www.carsbruh.com/petrol-prices/> * <https://carbuyer.com.sg/where-to-charge-electric-vehicle-singapore/> |
| **Alternate Flow** | **AF1** 5.1) Car Owner selects Electric Fuel Type  5.1.1) Car Owner inputs Fuel Information for Fuel Type Electric   1. Preferred Charging Station (Shell Recharge, Bluecharge, CDG EngiePetrol) [String] 2. Vehicle Current Type (Alternate or Direct) [String] 3. Vehicle Charging Rate (22, 30, 43, 50 kwH) [Float] 4. Vehicle Maximum Fuel Tank Capacity [Float]   5.1.2) Use case continues at Step 7  **AF2** 5.2) Car Owner selects Hybrid Fuel Type  5.2.1) Car Owner inputs Fuel Information for Fuel Type Hybrid   1. Preferred Petrol Kiosk (Shell, Esso, Caltex, etc.) [String] 2. Petrol Type (Shell FuelSave 95, Esso Synergy Supreme+, etc.) [String] 3. Vehicle Maximum Fuel Tank Capacity [Float] 4. Preferred Charging Station (Shell Recharge, Bluecharge, CDG EngiePetrol) [String] 5. Vehicle Current Type (Alternate or Direct) [String] 6. Vehicle Charging Rate (22, 30, 43, 50 kwH) [Float] 7. Vehicle Maximum Fuel Tank Capacity [Float]   5.2.2) Use case continues at Step 7  **AF3** 20.1) Car Owner selects to add another availability slot  20.1.1 Car Owner selects availability slot   * Start Date & Time (24h format) * End Date & Time (24h format)   20.1.2 System validates that availability slot input is not duplicate and valid  20.1.3 Use case resumes at Step 16 |



Test Case(s):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sequence** | **Flow** |  |  |  |  |
| **1** | **Basic Flow** |  |  |  |  |
| **2** | **Basic Flow** | **AF 1** | **Basic Flow** |  |  |
| **3** | **Basic Flow** | **AF 2** | **Basic Flow** |  |  |
| **4** | **Basic Flow** | **AF 3** | **Basic Flow** |  |  |
| **5** | **Basic Flow** | **AF 1** | **Basic Flow** | **AF3** | **Basic Flow** |
| **6** | **Basic Flow** | **AF 2** | **Basic Flow** | **AF3** | **Basic Flow** |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC01 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Gas |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| **2.** | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGX1234A** * Make: **Toyota** * Model: **Corolla** * Rental Rate: **50.0** * Maximum Seating Capacity: **5** * Year: **2020** * Mileage: **20000** * **Fuel Type:**  **“1” for Gas** | System Prompts for fuel type information |
| **3.** | Car Owner inputs fuel type information for Gas. | * Preferred Petrol Kiosk: **Shell** * Petrol Type: **Shell FuelSave 95** * Vehicle Maximum Fuel Tank Capacity: **60.0** | System prompts for confirmation that vehicle information is correct. |
| **4.** | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| **5.** | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| **6.** | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| **7.** | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| **8.** | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| **9.** | Car Owner inputs insurance information. | Coverage Details: **Full Coverage**  Insurance Start Date: **2024-10-26**  Insurance End Date: **2025-08-26** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| **10.** | Car Owner inputs availability slot. | Start DateTime: **2024-08-26 08:00**  End DateTime:  **2024-08-26 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| **13.** | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| **14.** | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| **15.** | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC02 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Electric |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. The Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| **2.** | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGA9876X** * Make: **Tesla** * Model: **Model 3** * Rental Rate: **75.0** * Maximum Seating Capacity: **5** * Year: **2022** * Mileage: **500** * Fuel Type: **“2” For Electric** | System Prompts for fuel type information |
| **3.** | Car Owner inputs fuel type information for Electric | * Preferred Charging Station: **Tesla Supercharger** * Vehicle Current Type: **Direct** * Vehicle Charging Rate: **50.0** * Vehicle Maximum Fuel Tank Capacity: **75.0** | System prompts for confirmation that vehicle information is correct. |
| **4.** | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| **5.** | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| **6.** | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| **7.** | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| **8.** | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| **9.** | Car Owner inputs insurance information. | Coverage Details**: Full Coverage**  Insurance Start Date: **2024-10-10**  Insurance End Date: **2025-01-01** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| **10.** | Car Owner inputs availability slot. | Start DateTime: **2024-10-10 08:00**  End DateTime: **2024-10-10 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| **13.** | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| **14.** | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| **15.** | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC03 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Hybrid |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. The Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| 1. | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| 2. | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGX5678B** * Make: **Toyota** * Model: **Prius** * Rental Rate: **100.0** * Maximum Seating Capacity: 5 * Year: **2022** * Mileage: **8000** * Fuel Type: **“3” for Hybrid** | System Prompts for fuel type information |
| 3. | Car Owner inputs fuel type information for Electric | * Preferred Petrol Kiosk: **Shell** * Petrol Type: **Shell FuelSave 95** * Vehicle Maximum Fuel Tank Capacity (Petrol): **45.0** * Preferred Charging Station: **Shell Recharge** * Vehicle Current Type: **Alternate** * Vehicle Charging Rate: **22.0** * Vehicle Maximum Fuel Tank Capacity (Electric): **10.0** | System prompts for confirmation that vehicle information is correct. |
| 4. | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| 5. | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| 6. | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| 7. | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| 8. | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| 9. | Car Owner inputs insurance information. | Coverage Details: **Full Coverage**  Insurance Start Date: **2024-10-10**  Insurance End Date: **2025-01-01** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| 10. | Car Owner inputs availability slot. | Start DateTime: **2024-10-10 08:00**  End DateTime: **2024-10-10 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| 13. | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| 14. | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| 15. | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC04 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Gas and adds multiple availability slots |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| 1. | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| 2. | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGX1234A** * Make: **Toyota** * Model: **Corolla** * Rental Rate: **50.0** * Maximum Seating Capacity: **5** * Year: **2020** * Mileage: **20000** * Fuel Type: “**1**” for Gas | System Prompts for fuel type information |
| 3. | Car Owner inputs fuel type information for Gas. | * Preferred Petrol Kiosk: **Shell** * Petrol Type: **Shell FuelSave 95** * Vehicle Maximum Fuel Tank Capacity: **60.0** | System prompts for confirmation that vehicle information is correct. |
| 4. | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| 5. | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| 6. | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| 7. | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| 8. | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| 9. | Car Owner inputs insurance information. | Coverage Details: **Full Coverage**  Insurance Start Date: **2024-10-26**  Insurance End Date: **2025-08-26** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| 10. | Car Owner inputs availability slot. | Start DateTime: **2024-08-26 08:00**  End DateTime: **2024-08-26 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| 11. | Car Owner chooses to add another availability slot. | “**Yes**” as Input | System prompts for availability slot. |
| 12. | Car Owner inputs another availability slot. | Start DateTime: **2024-01-02 08:00**  End DateTime: **2024-01-02 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| 13. | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| 14. | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| 15. | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC05 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Electric and adds multiple availability slots |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| 1. | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| 2. | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGA9876X** * Make: **Tesla** * Model: **Model 3** * Rental Rate: **75.0** * Maximum Seating Capacity: **5** * Year: **2022** * Mileage: **500** * Fuel Type: “**2**” for Electric | System Prompts for fuel type information |
| 3. | Car Owner inputs fuel type information for Gas. | * Preferred Charging Station: **Tesla** **Supercharger** * Vehicle Current Type: **Direct** * Vehicle Charging Rate: **50.0** * Vehicle Maximum Fuel Tank Capacity: **75.0** | System prompts for confirmation that vehicle information is correct. |
| 4. | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| 5. | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| 6. | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| 7. | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| 8. | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| 9. | Car Owner inputs insurance information. | Coverage Details: **Full Coverage**  Insurance Start Date: **2024-10-26**  Insurance End Date: **2025-08-26** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| 10. | Car Owner inputs availability slot. | Start DateTime: **2024-08-26 08:00**  End DateTime: **2024-08-26 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| 11. | Car Owner chooses to add another availability slot. | “**Yes**” as Input | System prompts for availability slot. |
| 12. | Car Owner inputs another availability slot. | Start DateTime: **2024-01-02 08:00**  End DateTime: **2024-01-02 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| 13. | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| 14. | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| 15. | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC06 - Yeo Jin Rong** |
| **Test Description** | **:** | This test case tests whether a Car Owner can successfully register a vehicle with Fuel Type: Hybrid and adds multiple availability slots |
| **Use Case ID** | **:** | UC01 |
| **Use Case Name** | **:** | Register Vehicle |
| **Pre-Condition(s)** | **:** | 1. Car Owner is logged in and selects to register a new vehicle 2. Car Owner has required Vehicle Information readily available for input and is all valid. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Car Owner selects to register a new vehicle | N/A | System prompts for vehicle information input |
| **2.** | Car Owner inputs vehicle information | * Vehicle Registration Number: **SGX5678B** * Make: **Toyota** * Model: **Prius** * Rental Rate: **100.0** * Maximum Seating Capacity: **5** * Year: **2022** * Mileage: **8000** * Fuel Type: “**3**” for Hybrid | System Prompts for fuel type information |
| **3.** | Car Owner inputs fuel type information for Gas. | * Preferred Petrol Kiosk: **Shell** * Petrol Type: **Shell FuelSave 95** * Vehicle Maximum Fuel Tank Capacity (Petrol): **45.0** * Preferred Charging Station: Shell Recharge * Vehicle Current Type: **Alternate** * Vehicle Charging Rate: **22.0** * Vehicle Maximum Fuel Tank Capacity (Electric): **10.0** | System prompts for confirmation that vehicle information is correct. |
| **4.** | Car Owner confirms vehicle information. | “**Yes**” as Input | System validates the uniqueness of the Vehicle Registration Number and that all required fields are filled. System prompts the Car Owner to upload photos for the vehicle. |
| **5.** | Car Owner uploads a photo | Photo ID: **1234** | Photo is added to the vehicle. System prompts to upload another photo |
| **6.** | Car Owner chooses to upload another photo | “**Yes**” as Input | System prompts to upload photo |
| **7.** | Car Owner uploads another photo. | Photo ID: **12345** | Photo is added to the vehicle. System prompts to upload another photo |
| **8.** | Car Owner chooses not to add more photos. | “**No**” as Input | System prompts for insurance details for the vehicle. |
| **9.** | Car Owner inputs insurance information. | Coverage Details: **Full Coverage**  Insurance Start Date: **2024-10-26**  Insurance End Date: **2025-08-26** | System validates insurance details and adds them to the vehicle. System prompts for Availability Slot |
| **10.** | Car Owner inputs availability slot. | Start DateTime: **2024-08-26 08:00**  End DateTime: **2024-08-26 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| **11.** | Car Owner chooses to add another availability slot. | “**Yes**” as Input | System prompts for availability slot. |
| **12.** | Car Owner inputs another availability slot. | Start DateTime: **2024-01-02 08:00**  End DateTime: **2024-01-02 20:00** | Availability slot is validated and added to the list of Availability Slots (Vehicle Availability) in Vehicle. System prompts to add another availability slot. |
| **13.** | Car Owner chooses not to add more availability slots. | “**No**” as input | System displays vehicle information and prompts for final confirmation. |
| **14.** | Car Owner confirms that the information is correct. | “**Yes**” as Input | System prompts the Car Owner to complete vehicle registration or register another vehicle. |
| **15.** | Car Owner completes vehicle registration. | “**Yes**” as Input | System notifies the Car Owner of successful registration. |

### 3.3.2 Use Case Specification by Ng Jing Zhan Garrett

Student Name: Ng Jing Zhan Garrett

Student ID: S10257347

|  |  |
| --- | --- |
| **Use Case ID** | UC02 - Garrett |
| **Use Case Name** | Make Payment |
| **Brief Description** | This use case allows registered renters to make payments for rental fees or penalties on the iCar platform. Renters can choose from various payment methods, including debit/credit card, digital wallet and PayNow. The process includes an optional step for applying a discount code. Upon successful payment, the system updates the payment status, sends confirmation receipts to the renter, and communicates booking details to the relevant stakeholders. |
| **Actor(s)** | Renter (Primary)  Card Payment System (Supporting)  PayNow System (Supporting)  Email System (Supporting)  iCar Rental Platform (Supporting) |
| **Pre-condition(s)** | The renter must be a registered user of the iCar platform.  The renter has selected to pay for a rental or has a penalty fee to settle. |
| **Post-condition(s)** | Payment confirmation receipts are sent to the renter.  Payment status is confirmed in the database.  For rental fee payments, renter and the vehicle owner receive booking details through email. |
| **Basic Flow** | 1. This use case starts when the renter selects to make rental payment on the iCar platform. 2. The system checks for renter type. 3. The renter is a normal renter, no discount is applied 4. The system prompts renter to input valid optional discount code 5. The renter chooses to skip inputting discount code [alt 1] 6. The system calculates the total cost, including base price and any applied discounts. 7. The system displays the cost to the renter. 8. The system prompts the renter to select their preferred payment method. 9. The renter selects to pay by Debit/Credit card [alt 2] 10. The system executes the Pay By Debit/Credit Card use case to proceed with payment. [alt 3] 11. The system records the payment status as ‘confirmed’. 12. The system displays a message indicating successful payment. 13. The system sends a confirmation email to the renter with an attached receipt. 14. The system sends a summary of booking details to the car owner and the renter. 15. The system thanks renter for using payment service. 16. Use case ends |
| **Alternate Flows** | **AF1: Discount Code Input**  5.1 The renter chooses to input a discount code. 5.2 The system prompts the renter to input the discount code. 5.3 The renter inputs the discount code. 5.4 The system checks the validity of the discount code. 5.5 The use case resumes at step 6.  **AF2: Alternative Payment**  9.1 Renter selects “Pay by PayNow”  9.2 System executes "Pay By PayNow" use case  9.3 Use case resumes at step 11 when the payment is confirmed.  **AF3: Payment failure**  10.1 System notifies the renter of the payment failure  10.2 The renter is prompted to try again with the same or different payment method.  10.3 Use case resumes at step 8. |

|  |  |
| --- | --- |
| **Use Case ID** | UC02-01 - Garrett |
| **Use Case Name** | Pay By Debit/Credit Card |
| **Brief Description** | This use case allows registered renters to make debit/credit card payments on the iCar platform, renters will be able to input their payment details. |
| **Actor(s)** | Renter (Primary)  Card Payment System (Supporting)  iCar Rental Platform (Supporting) |
| **Pre-condition(s)** | The renter must be a registered user of the iCar platform.  The renter has selected to pay for a rental or has a penalty fee to settle.  The renter has selected the Pay By Debit/Credit card option. |
| **Post-condition(s)** | Payment status is updated as "confirmed" in the database as a payment record. |
| **Basic Flow** | 1. This use case starts when the renter selects to make debit/credit card payment on the iCar platform. 2. The system prompts the renter to choose between debit/credit card or digital wallet payment. 3. The renter chooses to make debit/credit card payment. [alt 1] 4. The system prompts the renter for card details and billing address, displaying a form for input.  * Card Number * Name on Card * Expiry Date (MM/YY) * CVV * Billing Address  1. The renter inputs their card details and billing address. 2. The system validates the card details to ensure that they are entered with the correct format. [alt 2] 3. The system transmits card details to the selected payment gateway to process the payment. 4. The system receives payment verification from the payment gateway. 5. Make Payment use case resumes at step 11, where the system records payment status as "confirmed" |
| **Alternate Flows** | **AF1: Digital Wallet Payment**  3.1 The renter chooses to make digital wallet payment.  3.2 Use case resumes at step 7.  **AF2: Invalid Card Details**  6.1 The system detects invalid card details  6.2 The system prompts the renter to re-enter the correct card details or choose a different payment method.  6.3 The renter chooses to re-renter card details.  6.4 Use case resumes at step 4. |

**Make Payment: Pay By Debit/Credit card:**

A diagram of a flow chart

Description automatically generatedA diagram of a flowchart

Description automatically generated

**Make Payment** Test Cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sequence** | **Flow** |  |  |  |
| **1** | **Basic Flow** |  |  |  |
| **2** | **Basic Flow** | **AF 1** | **Basic Flow** |  |
| **3** | **Basic Flow** | **AF 2** | **Basic Flow** |  |
| **4** | **Basic Flow** | **AF 3** | **Basic Flow** |  |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC01 – Ng Jing Zhan Garrett** |
| **Test Description** | **:** | This test case is to test for whether the renter can make payment on the iCar platform |
| **Use Case ID** | **:** | UC02 |
| **Use Case Name** | **:** | Make Payment |
| **Pre-Condition(s)** | **:** | Renter has rental fee to pay. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to make rental payment | | NA | System checks for renter type. |
| **2.** | System checks for renter type | Renter type: Normal | System identifies the renter as a normal renter, no discount is applied. |
| **3.** | |  | | --- | | System prompts for optional discount code | | |  | | --- | | NA | | |  | | --- | | System prompts renter to input valid optional discount code. | |
| **4.** | Renter chooses to skip discount code input | **'n'** as input | System proceeds to calculate the total cost. |
| **5.** | System calculates the total cost | Base price: $70.00 | System calculates the total cost as $70.00 and displays it to the renter. |
| **6.** | |  | | --- | |  |  |  | | --- | | System prompts for preferred payment method | | NA | System displays options for payment methods: Debit/Credit Card, PayNow, Exit. |
| **7.** | Renter selects to pay by Debit/Credit card | "**1**" | System executes the Pay By Debit/Credit Card use case. |
| **8.** | System prompts renter for credit card payment. | **"1"** | System displays options for payment methods:  Manual input, Digital Wallet, Back. |
| **9.** | Renter enters card details | Card Number: **1234567812345678** Name on Card: **John Ng** Expiry Date: **12/25** CCV: **123** | System validates card details and processes the payment. |
| **10.** | Payment processing is successful | NA | System records the payment status as ‘confirmed’. |
| **11.** | System displays payment success message | NA | System displays a message indicating successful payment. |
| **12.** | System sends confirmation email to renter | NA | |  | | --- | |  |  |  | | --- | | System sends a confirmation email with the receipt to the renter. | |
| **13.** | System sends summary of booking details | NA | System sends a summary of booking details to both the renter and the car owner. |
| **14.** | System thanks renter for using payment service | NA | System thanks renter for using the payment service. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC02 – Ng Jing Zhan Garrett** |
| **Test Description** | **:** | This test case is to test for whether the renter can use a discount code on the payment service |
| **Use Case ID** | **:** | UC02 |
| **Use Case Name** | **:** | Make Payment |
| **Pre-Condition(s)** | **:** | Renter has rental fee to pay. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to make rental payment | | NA | System checks for renter type. |
| **2.** | System checks for renter type | Renter type: Normal | System identifies the renter as a normal renter, no discount is applied. |
| **3.** | |  | | --- | | System prompts for optional discount code | | |  | | --- | | NA | | |  | | --- | | System prompts renter to input valid optional discount code. | |
| **4.** | Renter chooses to input discount | **'y'** as input | System prompts renter to input the discount code. |
| **5.** | |  | | --- | |  |  |  | | --- | | Renter inputs valid discount code | | **"DIST10"** | System checks validity of discount code. |
| **6.** | Discount code is valid | Discount applied: $7 | System applies the discount and proceeds to calculate the total cost. |
| **7.** | System calculates the total cost | Base price: $70.00  Discount: $7.00 | System calculates the total cost as $63.00 and displays it to the renter. |
| **8.** | |  | | --- | |  |  |  | | --- | | System prompts for preferred payment method | | NA | System displays options for payment methods: Debit/Credit Card, PayNow, Exit. |
| **9.** | Renter selects to pay by Debit/Credit card | "**1**" | System executes the Pay By Debit/Credit Card use case. |
| **10.** | System prompts renter for credit card payment. | "**1**" | System displays options for payment methods:  Manual input, Digital Wallet, Back. |
| **11.** | Renter enters card details | Card Number: **1234567812345678** Name on Card: **John Ng** Expiry Date: **12/25** **CCV: 123** | System validates card details and processes the payment. |
| **12.** | Payment processing is successful | NA | System records the payment status as ‘confirmed’. |
| **13.** | System displays payment success message | NA | System displays a message indicating successful payment. |
| **14.** | System sends confirmation email to renter | NA | |  | | --- | |  |  |  | | --- | | System sends a confirmation email with the receipt to the renter. | |
| **15.** | System sends summary of booking details | NA | System sends a summary of booking details to both the renter and the car owner. |
| **16.** | System thanks renter for using payment service | NA | System thanks renter for using the payment service. |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC03 – Ng Jing Zhan Garrett** |
| **Test Description** | **:** | This test case is to test for whether the renter can make payment with PayNow System |
| **Use Case ID** | **:** | UC02 |
| **Use Case Name** | **:** | Make Payment |
| **Pre-Condition(s)** | **:** | Renter has rental fee to pay. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to make rental payment | | NA | System checks for renter type. |
| **2.** | System checks for renter type | Renter type: Normal | System identifies the renter as a normal renter, no discount is applied. |
| **3.** | |  | | --- | | System prompts for optional discount code | | |  | | --- | | NA | | |  | | --- | | System prompts renter to input valid optional discount code. | |
| **4.** | Renter chooses to skip discount code input | **'n'** as input | System proceeds to calculate the total cost. |
| **5.** | System calculates the total cost | Base price: $70.00 | System calculates the total cost as $70.00 and displays it to the renter. |
| **6.** | |  | | --- | |  |  |  | | --- | | System prompts for preferred payment method | | NA | System displays options for payment methods: Debit/Credit Card, PayNow, Exit. |
| **7.** | Renter selects to pay by PayNow | "**2**" | System executes Pay By PayNow use case. |
| **8.** | Payment processing is successful | NA | |  | | --- | |  |  |  | | --- | | System records the payment status as ‘confirmed’. | |
| **9.** | System displays payment success message | NA | System displays a message indicating successful payment. |
| **10.** | System sends confirmation email to renter | NA | |  | | --- | |  |  |  | | --- | | System sends a confirmation email with the receipt to the renter. | |
| **11.** | System sends summary of booking details | NA | System sends a summary of booking details to both the renter and the car owner. |
| **12.** | System thanks renter for using payment service | NA | System thanks renter for using the payment service. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | | **:** | **TC04 – Ng Jing Zhan Garrett** | |
| **Test Description** | | | **:** | This test case is to test whether the system prompts the renter for re-input when payment validation fails. | |
| **Use Case ID** | | | **:** | UC02 | |
| **Use Case Name** | | | **:** | Make Payment | |
| **Pre-Condition(s)** | | | **:** | Renter has rental fee to pay. | |
|  | | |  |  | |
| **Sequence** | **Scenario** | | **Test Data** | **Expected Result** |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to make rental payment | | | NA | System checks for renter type. |
| **2.** | System checks for renter type | | Renter type: Normal | System identifies the renter as a normal renter, no discount is applied. |
| **3.** | |  | | --- | | System prompts for optional discount code | | | |  | | --- | | NA | | |  | | --- | | System prompts renter to input valid optional discount code. | |
| **4.** | Renter chooses to skip discount code input | | **'n'** as input | System proceeds to calculate the total cost. |
| **5.** | System calculates the total cost | | Base price: $70.00 | System calculates the total cost as $70.00 and displays it to the renter. |
| **6.** | |  | | --- | |  |  |  | | --- | | System prompts for preferred payment method | | | NA | System displays options for payment methods: Debit/Credit Card, PayNow, Exit. |
| **7.** | Renter selects to pay by Debit/Credit card | | "**1**" | System executes the Pay By Debit/Credit Card use case. |
| **8.** | System prompts renter for credit card payment. | | "**1**" | System displays options for payment methods:  Manual input, Digital Wallet, Back. |
| **9.** | Renter enters card details | | Card Number: **12345678123456** Name on Card: **John Ng** Expiry Date: **12/25** CCV: **123** | System is unable to validate card details and cannot continue to process the payment. |
| **10.** | System notifies renter of payment failure | | NA | System prompts renter to try again with the same or different payment method. |
| **11.** | Renter tries the same payment method again | | Card Number: **1234567812345678** Name on Card: **John Ng** Expiry Date: **12/25** CCV: **123** | System validates card details and processes the payment. |
| **12.** | Payment processing is successful | | NA | System records the payment status as ‘confirmed’. |
| **13.** | System displays payment success message | | NA | System displays a message indicating successful payment. |
| **14.** | System sends confirmation email to renter | | NA | |  | | --- | |  |  |  | | --- | | System sends a confirmation email with the receipt to the renter. | |
| **15.** | System sends summary of booking details | | NA | System sends a summary of booking details to both the renter and the car owner. |
| **16.** | System thanks renter for using payment service | | NA | System thanks renter for using the payment service. |

### 3.3.3 Use Case Specification by Liang Dingxuan

Student Name: Liang Dingxuan

Student ID: S10258272

|  |  |
| --- | --- |
| **Use Case ID** | UC03 - Dingxuan |
| **Use Case Name** | Return Car |
| **Brief Description** | This use case allows renter to return cars they rented, involving selecting car to return, then renter will have to make any extra |
| **Actor(s)** | Renter |
| **Pre-condition(s)** | The user has logged in as renter  The renter has a car already rented |
| **Post-condition(s)** | The car is returned, and the system records the return.    Any additional charges are calculated and notified to the renter.  Any damages reported are recorded in the system. |
| **Basic flow** | 1. Use case starts when renter selects return car in navigation menu 2. System retrieves cars user has rented 3. System displays all cars available for return 4. System prompts user to select a car 5. Renter chooses specific car to return 6. System retrieves booking details 7. System displays booking details 8. System prompts for user confirmation to return car 9. Renter selects confirm to return car 10. System prompts for return method 11. Renter Selects return to designated icar station method. 12. System displays all available designated icar station for return 13. System prompts Renter for designated icar station location 14. Renter inputs designated Icar station location 15. System validates input. 16. System retrieves current time 17. System checks if car is returned later than the original end time 18. System detects car is returned on time 19. System checks for charges 20. System detects no charges 21. System notifies Renter no charges 22. System prompts for any damages done to the car if any 23. Renter inputs no damages 24. System notes no reported damages 25. Use case ends here |
| **Alternate Flows** | **AF1 User selects pickup instead of return**  9.1 User selects pickup  9.2 System prompts renter to input pickup location  9.3 Renter inputs pickup location  9.4 Use case resumes at step 13  **AF2 System detects charges**  19.1 System detects additional charges  19.2 System calculates additional charges  19.2 System notifies renter of charges  19.3 system prompts renter to make payment use case (use case)  19.4 Use case resumes at step 22  **AF6 Renter reports damages**  22.1 Renter inputs damages  22.2 System notes the reported damages  22.3Use Case resumes at step 25 |

A diagram of a flowchart

Description automatically generated

Test Cases:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Flow** |  |  |
| **1** | **Basic Flow** |  |  |
| **2** | **Basic Flow** | **AF1** | **Basic Flow** |
| **3** | **Basic Flow** | **AF2** | **Basic Flow** |
| **4** | **Basic Flow** | **AF3** | **Basic Flow** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | | **:** | **TC01 – Liang Dingxuan** | | |
| **Test Description** | | | **:** | This test case is to test for whether renter can return car on the Icar platform | | |
| **Use Case ID** | | | **:** | UC03 | | |
| **Use Case Name** | | | **:** | Return car as renter | | |
| **Pre-Condition(s)** | | | **:** | The user has logged in as renter  The renter has a car already rented | | |
| **Sequence** | **Scenario** | | | **Test Data** | **Expected Result** | |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to Return Vehicle | | | | NA | System retrieves List of bookings | |
| **2.** | System displays all cars available for return | | | NA | List of cars available for return is shown. | |
| **3.** | System prompts renter to select a car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **4.** | Renter chooses a specific car to return | | | |  | | --- | | Booking ID: "**1**" |  |  | | --- | |  | | |  | | --- | | Selected booking details are retrieved. |  |  | | --- | |  | | |
| **5.** | System displays booking details | | | NA | |  | | --- | | Booking details, including car and rental information, are shown. |  |  | | --- | |  | | |
| **6.** | System prompts for renter confirmation to return the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **7.** | Renter confirms to return the car | | | NA | |  | | --- | | System prompts for the return method. |  |  | | --- | |  | | |
| **8.** | Renter selects "Return to designated Icar station" method | | | NA | |  | | --- | | System displays all available designated Icar stations for return. |  |  | | --- | |  | | |
| **9.** | System prompts renter for designated Icar station location | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **10.** | Renter inputs designated Icar station location | | | |  | | --- | | Location: "**123 Main St**" |  |  | | --- | |  | | |  | | --- | | System validates the input location. |  |  | | --- | |  | | |
| **11.** | System validates input | | | NA | |  | | --- | | System retrieves the current time and checks if the car is returned later than the original end time. |  |  | | --- | |  | | |
| **12.** | System detects the car is returned on time | | | NA | |  | | --- | | System checks for any charges. |  |  | | --- | |  | | |
| **13.** | System detects no charges | | | NA | |  | | --- | | System notifies the renter that there are no charges. |  |  | | --- | |  | | |
| **14.** | System prompts for any damages done to the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **15.** | Renter inputs "No damages" | | | NA | |  | | --- | | System notes no reported damages. |  |  | | --- | |  | | |
| **16.** | |  | | --- | | Use case ends here |  |  | | --- | |  | | | | NA | The car is successfully returned, and the system records the return. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | **:** | **TC02 – Liang DIngxuan** | | |
| **Test Description** | | **:** | This test case is to test for whether renter can return car on the Icar platform Using pickup instead of return | | |
| **Use Case ID** | | **:** | UC03 | | |
| **Use Case Name** | | **:** | Return car as renter | | |
| **Pre-Condition(s)** | | **:** | The user has logged in as renter  The renter has a car already rented | | |
| **Sequence** | **Scenario** | | | **Test Data** | **Expected Result** | |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to Return Vehicle | | | | NA | System retrieves List of bookings | |
| **2.** | System displays all cars available for return | | | NA | List of cars available for return is shown. | |
| **3.** | System prompts renter to select a car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **4.** | Renter chooses a specific car to return | | | |  | | --- | | Booking ID: "**1**" |  |  | | --- | |  | | |  | | --- | | Selected booking details are retrieved. |  |  | | --- | |  | | |
| **5.** | System displays booking details | | | NA | |  | | --- | | Booking details, including car and rental information, are shown. |  |  | | --- | |  | | |
| **6.** | System prompts for renter confirmation to return the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **7.** | Renter confirms to return the car | | | NA | |  | | --- | | System prompts for the return method. |  |  | | --- | |  | | |
| **8.** | Renter selects "Pickup" method | | | NA | |  | | --- | | System displays all available designated Icar stations for return. |  |  | | --- | |  | | |
| **9.** | System prompts renter for designated Icar station location | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **10.** | Renter inputs the pickup location | | | |  | | --- | | Location: "**123 john’s St**" (Renter’s house) |  |  | | --- | |  | | |  | | --- | | System validates the input location. |  |  | | --- | |  | | |
| **11.** | System validates input | | | NA | |  | | --- | | System retrieves the current time and checks if the car is returned later than the original end time. |  |  | | --- | |  | | |
| **12.** | System detects the car is returned on time | | | NA | |  | | --- | | System checks for any charges. |  |  | | --- | |  | | |
| **13.** | System detects no charges | | | NA | |  | | --- | | System notifies the renter that there are no charges. |  |  | | --- | |  | | |
| **14.** | System prompts for any damages done to the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **15.** | Renter inputs "No damages" | | | NA | |  | | --- | | System notes no reported damages. |  |  | | --- | |  | | |
| **16.** | |  | | --- | | Use case ends here |  |  | | --- | |  | | | | NA | The car is successfully returned, and the system records the return. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | **:** | **TC03 – Liang DIngxuan** | | |
| **Test Description** | | **:** | This test case is to test for whether renter can return car on the Icar platform and make payment when charges apply | | |
| **Use Case ID** | | **:** | UC03 | | |
| **Use Case Name** | | **:** | Return car as renter | | |
| **Pre-Condition(s)** | | **:** | The user has logged in as renter  The renter has a car already rented | | |
| **Sequence** | **Scenario** | | | **Test Data** | **Expected Result** | |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to Return Vehicle | | | | NA | System retrieves List of bookings | |
| **2.** | System displays all cars available for return | | | NA | List of cars available for return is shown. | |
| **3.** | System prompts renter to select a car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **4.** | Renter chooses a specific car to return | | | |  | | --- | | Booking ID: "**1**" |  |  | | --- | |  | | |  | | --- | | Selected booking details are retrieved. |  |  | | --- | |  | | |
| **5.** | System displays booking details | | | NA | |  | | --- | | Booking details, including car and rental information, are shown. |  |  | | --- | |  | | |
| **6.** | System prompts for renter confirmation to return the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **7.** | Renter confirms to return the car | | | NA | |  | | --- | | System prompts for the return method. |  |  | | --- | |  | | |
| **8.** | Renter selects "Return to designated Icar station" method | | | NA | |  | | --- | | System displays all available designated Icar stations for return. |  |  | | --- | |  | | |
| **9.** | System prompts renter for designated Icar station location | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **10.** | Renter inputs designated Icar station location | | | |  | | --- | | Location: "**123 Main St**" |  |  | | --- | |  | | |  | | --- | | System validates the input location. |  |  | | --- | |  | | |
| **11.** | System validates input | | | NA | |  | | --- | | System retrieves the current time and checks if the car is returned later than the original end time. |  |  | | --- | |  | | |
| **12.** | System detects the car is returned on time | | | NA | |  | | --- | | System checks for any charges. |  |  | | --- | |  | | |
| **13.** | System detects Additional Charges | | | NA | |  |  |  | | --- | --- | --- | | |  | | --- | | System calculates the additional charges and notifies the renter. |  |  | | --- | |  | |  |  | | --- | |  | | |
| **14.** | |  | | --- | | System prompts the renter to make a payment |  |  | | --- | |  | | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **15.** | |  | | --- | | Renter makes the payment |  |  | | --- | |  | | | | |  | | --- | | Payment details |  |  | | --- | |  | | |  | | --- | | System processes the payment and confirms the return. |  |  | | --- | |  | | |
| **16.** | System prompts for any damages done to the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **17.** | Renter inputs "No damages" | | | NA | |  | | --- | | System notes no reported damages. |  |  | | --- | |  | | |
| **18.** | |  | | --- | | Use case ends here |  |  | | --- | |  | | | | NA | The car is successfully returned, and the system records the return. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | | **:** | **TC04 – Liang DIngxuan** | | |
| **Test Description** | | **:** | This test case is to test for whether renter can return car on the Icar platform and input damages the car received | | |
| **Use Case ID** | | **:** | UC03 | | |
| **Use Case Name** | | **:** | Return car as renter | | |
| **Pre-Condition(s)** | | **:** | The user has logged in as renter  The renter has a car already rented | | |
| **Sequence** | **Scenario** | | | **Test Data** | **Expected Result** | |
| **1.** | |  | | --- | |  |  |  | | --- | | Renter selects to Return Vehicle | | | | NA | System retrieves List of bookings | |
| **2.** | System displays all cars available for return | | | NA | List of cars available for return is shown. | |
| **3.** | System prompts renter to select a car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **4.** | Renter chooses a specific car to return | | | |  | | --- | | Booking ID: "**1**" |  |  | | --- | |  | | |  | | --- | | Selected booking details are retrieved. |  |  | | --- | |  | | |
| **5.** | System displays booking details | | | NA | |  | | --- | | Booking details, including car and rental information, are shown. |  |  | | --- | |  | | |
| **6.** | System prompts for renter confirmation to return the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **7.** | Renter confirms to return the car | | | NA | |  | | --- | | System prompts for the return method. |  |  | | --- | |  | | |
| **8.** | Renter selects "Return to designated Icar station" method | | | NA | |  | | --- | | System displays all available designated Icar stations for return. |  |  | | --- | |  | | |
| **9.** | System prompts renter for designated Icar station location | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **10.** | Renter inputs designated Icar station location | | | |  | | --- | | Location: "**123 Main St**" |  |  | | --- | |  | | |  | | --- | | System validates the input location. |  |  | | --- | |  | | |
| **11.** | System validates input | | | NA | |  | | --- | | System retrieves the current time and checks if the car is returned later than the original end time. |  |  | | --- | |  | | |
| **12.** | System detects the car is returned on time | | | NA | |  | | --- | | System checks for any charges. |  |  | | --- | |  | | |
| **13.** | System detects no charges | | | NA | |  | | --- | | System notifies the renter that there are no charges. |  |  | | --- | |  | | |
| **14.** | System prompts for any damages done to the car | | | NA | |  | | --- | | System waits for renter input. |  |  | | --- | |  | | |
| **15.** | Renter inputs "Damaged" | | | NA | |  | | --- | | System prompts for damages |  |  | | --- | |  | | |
| **17.** | Renter inputs damage details | | | Damages details: Details: **caved in bumper**  Location: **21 jump street**  Date: **8/3/2024** | |  | | --- | | System records the damages. |  |  | | --- | |  | | |
| **16.** | |  | | --- | | Use case ends here |  |  | | --- | |  | | | | NA | The car is successfully returned, and the system records the return. | |

### 3.3.4 Use Case Specification by Ng Kai Huat Jason

Student Name: Ng Kai Huat Jason

Student ID: S10262552

|  |  |
| --- | --- |
| **Use Case ID** | UC04 – Ng Kai Huat Jason |
| **Use Case Name** | Register as “Renter” |
| **Brief Description** | This use case allows a new user to register as a “Renter” |
| **Actor(s)** | User (Primary)  System (Supporting)  Email System (Supporting)  LTA System |
| **Pre-condition(s)** | User is not logged in.  Users must have internet access.  User must have a valid email address. |
| **Post-condition(s)** | The User is registered as a “Renter” with Registration Status “Pending”  The admin is notified of the new registration request.  The User can view vehicles but is unable to book them until registration is fully approved by an Admin. |
| **Basic Flow** | 1. The Use Case Starts 2. User selects to register as a “Renter” 3. System displays a registration form that requires user’s personal information and prompts for input 4. User inputs their personal information:    * **Full Name (as per NRIC)**    * **Date of Birth (MM/DD/YYYY Format)**    * **Contact Number (unique)**    * **Address**    * **Zip Code**    * **Email Address (unique)** 5. System validates if personal details are correct 6. System prompts for membership type (Prime or Regular) 7. User input Regular membership type 8. System prompts user to confirm choice of membership, notifying that it cannot be altered until end of month 9. User confirms membership choice 10. System displays and prompts for confirmation for inputted personal details and membership type 11. User confirms all details 12. System Creates a Renter without a driver licence 13. System prompts for driver’s license information input 14. User inputs driver’s license information  * **Licence Number** * **Number of demerit points** * **Class of license** * **Date of Issue for License** * **Certificate of Merit Eligibility**  1. System Validates Licences 2. System Creates a Licence 3. System displays created licence details 4. System prompts for confirmation for created licence details 5. User confirms driver’s license is valid 6. System adds created Licence to created Renter 7. System displays “Terms and Service” and prompts for acceptance of "Terms and Service" 8. User agrees to Terms and Service 9. Email System sends a verification email containing a six-digit code to User. 10. System prompts for the six-digit code 11. User enters the received six-digit code to verify their email address 12. System validates whether six-digit code is legitimate 13. System marks registration status as “Pending” 14. System notifies User of new Registration. 15. System notifies Admin to review the registration 16. The Use Case Ends |
|  | **AF1 User selects “Prime” Membership Type** 7.1 User selects “Prime” membership  7.1 System displays relevant information for prime membership  7.2 System prompts user to confirm prime membership  7.3 User confirms prime membership  7.4 Use case proceeds to step 10  **AF2 Verification Code is Invalid, User wants to re-enter the code**  25.1 System displays that verification code is invalid  25.2 System prompts user to request for new code or re-enter code  25.3 User selects to re-enter code  25.4 Use case resumes at step 24  **AF3 Verification Code is Invalid, User wants a new code**  25.1 System displays that verification code is invalid  25.2 System prompts user to request for a new code or re-enter code  25.3 User selects for new verification code  25.4 System displays “New Verification code sent to your email Address.”  25.5 Use case resumes at step 24 |

**A diagram of a flowchart

Description automatically generated**

Test Case(s):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sequence** | **Flow** |  |  |  |
| **1** | **Basic Flow** |  |  |  |
| **2** | **Basic Flow** | **AF 1** | **Basic Flow** |  |
| **3** | **Basic Flow** | **AF 2** | **Basic Flow** |  |
| **4** | **Basic Flow** | **AF 3** | **Basic Flow** |  |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC01 – Ng Kai Huat Jason** |
| **Test Description** | **:** | This test case is to test if a new user can sign up as a regular renter. |
| **Use Case ID** | **:** | UC04 |
| **Use Case Name** | **:** | Register a New Renter |
| **Pre-Condition(s)** | **:** | System is ready for a new renter registration. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | System displays personal detail form | NA | System will display and prompt for personal details |
| **1.** | User enters their personal details when prompted | Full Name: **John Ng**  Contact Number: **9002 1002**  Date of Birth: **01/10/1990**  Address: **Gardens By the Bay #01-01**  Zip Code: **600320**  Email Address: [**john@gmail.com**](mailto:john@gmail.com)  Password: **john123** | System will prompt display and for membership type |
| **2.** | User chooses to be a “Regular” Member | “**regular**” as input | System will prompt for confirmation of membership type |
| **3.** | User confirms membership type | “**y**” as input | System will display and prompt for confirmation of personal details and membership type |
| **4.** | User confirms personal details and membership type | “**y**” as input | System display and prompts for driver licence details |
| **5.** | User enters their driver license details when prompted | License No: **12345678**  Number of Demerit Points: **0**  Class of License: **2A**  Date of Issue: **01/12/2028**  Certificate of Merit Eligibility: **Yes** | System will display and prompt if the inputted driver licence details are correct |
| **6.** | User confirms driver license details | “**y**” as input | System will validate and confirm driver licence details |
| **7.** | System displays and prompts for acceptance of Terms of Services | NA | System will display Terms of Services and prompt for acceptance |
| **8.** | User accepts Terms of Services | “**y**” as input | System will generate a verification code and send to user’s email |
| **9.** | System will send a code to user’s email address | Code: “**ABCDE**” (code is randomly generated) | System prompts user for the code |
| **10.** | User inputs verification code | “**ABCDE**” as input | System will validate the code |
| **11.** | System will notify Admin of new registration | NA | System notifies admin to review the new registration |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC02 – Ng Kai Huat Jason** |
| **Test Description** | **:** | This test case is to test if a new user can sign up as a prime renter |
| **Use Case ID** | **:** | UC04 |
| **Use Case Name** | **:** | Register a New Renter |
| **Pre-Condition(s)** | **:** | System is ready for a new renter registration. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | System displays personal detail form | NA | System will display and prompt for personal details |
| **1.** | User enters their personal details when prompted | Full Name: **John Ng**  Contact Number: **9002 1002**  Date of Birth: **01/10/1990**  Address: **Gardens By the Bay #01-01**  **Zip Code: 600320**  Email Address: [**john@gmail.com**](mailto:john@gmail.com)  Password: **john123** | System will prompt display and for membership type |
| **2.** | **Alternate Flow 1:** User chooses to be a “Prime” Member | “**prime**” as input | System will display prime member info and prompt for confirmation of membership type |
| **3.** | User confirms membership type | “**y**” as input | System will display and prompt for confirmation of personal details and membership type |
| **4.** | User confirms personal details and membership type | “**y**” as input | System display and prompts for driver licence details |
| **5.** | User enters their driver license details when prompted | License No: **12345678**  Number of Demerit Points: **0**  Class of License: **2A**  Date of Issue: **01/12/2028**  Certificate of Merit Eligibility: **Yes** | System will display and prompt if the inputted driver licence details are correct |
| **6.** | User confirms driver license details | “**y**” as input | System will validate and confirm driver licence details |
| **7.** | System displays and prompts for acceptance of Terms of Services | NA | System will display Terms of Services and prompt for acceptance |
| **8.** | User accepts Terms of Services | “**y**” as input | System will generate a verification code and send to user’s email |
| **9.** | System will send a code to user’s email address | Code: “**ABCDE**” (code is randomly generated) | System prompts user for the code |
| **10.** | User inputs verification code | “**ABCDE**” as input | System will validate the code |
| **11.** | System will notify user of new registration | NA | System will display “You have been registered successfully” |
| **12.** | System will notify Admin of new registration | NA | System notifies admin to review the new registration |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC03 – Ng Kai Huat Jason** |
| **Test Description** | **:** | This test case is to test whether the user can re-enter their code if code validation fails and proceed with registration |
| **Use Case ID** | **:** | UC04 |
| **Use Case Name** | **:** | Register a New Renter |
| **Pre-Condition(s)** | **:** | System is ready for a new renter registration. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | System displays personal detail form | NA | System will display and prompt for personal details |
| **1.** | User enters their personal details when prompted | Full Name: **John Ng**  Contact Number: **9002 1002**  Date of Birth: **01/10/1990**  Address: **Gardens By the Bay #01-01**  Zip Code: **600320**  Email Address: [**john@gmail.com**](mailto:john@gmail.com)  Password: **john123** | System will prompt display and for membership type |
| **2.** | User chooses to be a “Regular” Member | “**regular**” as input | System will prompt for confirmation of membership type |
| **3.** | User confirms membership type | “**y**” as input | System will display and prompt for confirmation of personal details and membership type |
| **4.** | User confirms personal details and membership type | “**y**” as input | System display and prompts for driver licence details |
| **5.** | User enters their driver license details when prompted | License No: **12345678**  Number of Demerit Points: **0**  Class of License: **2A**  Date of Issue: **01/12/2028**  Certificate of Merit Eligibility: **Yes** | System will display and prompt if the inputted driver licence details are correct |
| **6.** | User confirms driver license details | “**y**” as input | System will validate and confirm driver licence details |
| **7.** | System displays and prompts for acceptance of Terms of Services | NA | System will display Terms of Services and prompt for acceptance |
| **8.** | User accepts Terms of Services | “**y**” as input | System will generate a verification code and send to user’s email |
| **9.** | System will send a code to user’s email address | Code: “**ABCDE**” (code is randomly generated) | System prompts user for the code |
| **10.** | **Alternate flow 2:** User inputs wrong verification code | “**ZXCVB**” as input | System prompts if user wants to “1.re-enter code” or “2. request a new code” |
| **11.** | User wants to re-enter code | “**1**” | System will re-prompt for verification code |
| **12.** | User inputs correct verification code | “**ABCDE**” | System will validate code |
| **13.** | System will notify user of new registration |  | System will display” You have been registered successfully” |
| **14.** | System will notify Admin of new registration | NA | System notifies admin to review the new registration |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC04 – Ng Kai Huat Jason** |
| **Test Description** | **:** | This test case is to test whether the user can request a new code if code validation fails and proceed with registration |
| **Use Case ID** | **:** | UC04 |
| **Use Case Name** | **:** | Register a New Renter |
| **Pre-Condition(s)** | **:** | System is ready for a new renter registration. |

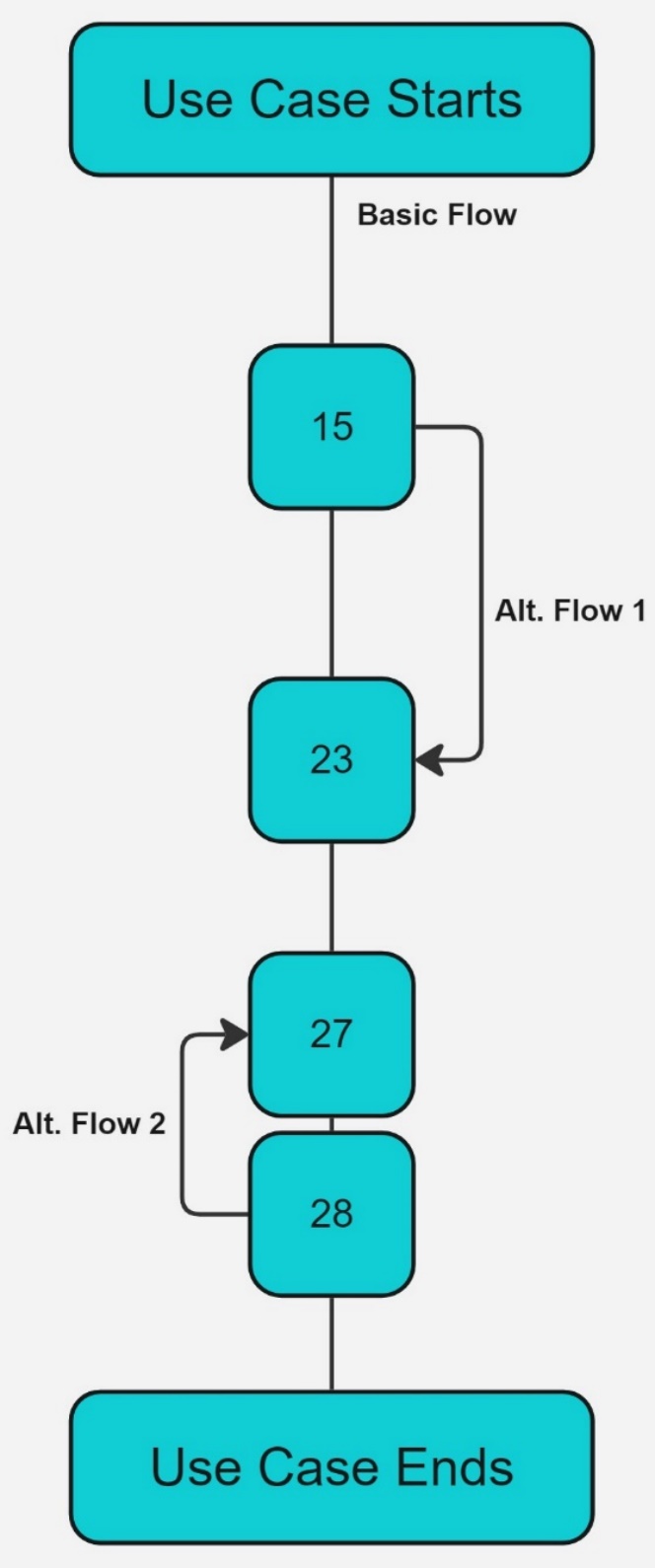
|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | System displays personal detail form | NA | System will display and prompt for personal details |
| **1.** | User enters their personal details when prompted | Full Name: **John Ng**  Contact Number: **9002 1002**  Date of Birth: **01/10/1990**  Address: **Gardens By the Bay #01-01**  Zip Code: **600320**  Email Address: [**john@gmail.com**](mailto:john@gmail.com)  Password: **john123** | System will prompt display and for membership type |
| **2.** | User chooses to be a “Regular” Member | “**regular**” as input | System will prompt for confirmation of membership type |
| **3.** | User confirms membership type | “**y**” as input | System will display and prompt for confirmation of personal details and membership type |
| **4.** | User confirms personal details and membership type | “**y**” as input | System display and prompts for driver licence details |
| **5.** | User enters their driver license details when prompted | License No: **12345678**  Number of Demerit Points: **0**  Class of License: **2A**  Date of Issue: **01/12/2028**  Certificate of Merit Eligibility: **Yes** | System will display and prompt if the inputted driver licence details are correct |
| **6.** | User confirms driver license details | “**y**” as input | System will validate and confirm driver licence details |
| **7.** | System displays and prompts for acceptance of Terms of Services | NA | System will display Terms of Services and prompt for acceptance |
| **8.** | User accepts Terms of Services | “**y**” as input | System will generate a verification code and send to user’s email |
| **9.** | System will send a code to user’s email address | Code: “**ABCDE**” (code is randomly generated) | System prompts user for the code |
| **10.** | **Alternate flow 3:** User inputs wrong verification code | “**ZXCVB**” as input | System prompts if user wants to “1.re-enter code” or “2. request a new code” |
| **11.** | User wants to request a new code | “**2**” | System will re-send a new verification code |
| **12.** | System will regenerate and resend code to user’s email | Code: "E1234” (code is randomly generated) | System will re-prompt for new verification code |
| **12.** | User inputs correct verification code | “**E1234**” | System will validate code |
| **13.** | System will notify user of new registration |  | System will display” You have been registered successfully” |
| **14.** | System will notify Admin of new registration | NA | System notifies admin to review the new registration |

### 3.3.5 Use Case Specification by Tan Guo Zhi Kelvin

Student Name: Tan Guo Zhi Kelvin

Student ID: S10262567

|  |  |
| --- | --- |
| **Use Case ID** | UC05 – Tan Guo Zhi Kelvin |
| **Use Case Name** | Book Vehicle |
| **Brief Description** | This use case allows the User to book vehicle on the iCar platform, enabling them to rent a vehicle for a specified period and choose to pick up at a designated iCar station or opt for doorstep delivery. After viewing detailed vehicle information, the User specifies the start and end date & time of the booking period, and the brand & model of the chosen vehicle. The system then checks the availability of the vehicle for the specified period. If available, the User input and confirm their booking details, selects the pickup or delivery option, agrees to the booking terms and conditions, and receives a detailed cost summary to finalise the booking. |
| **Actor(s)** | Renter |
| **Pre-condition(s)** | User is logged in as Renter and their account status is “Active”. |
| **Post-condition(s)** | System confirms the booking awaiting payment, and a new booking record is added. |
| **Basic Flow** | 1. The use case starts when Renter chooses to book a vehicle 2. System prompts Renter to input start date and time 3. Renter input start date and time 4. System prompts Renter to input end date and time 5. Renter input end date and time 6. System validates both start and end date-times and matches availability slot 7. System prompts Renter to input brand and model of chosen vehicle 8. Renter input brand and model of chosen vehicle 9. System validates that brand and model of chosen vehicle exist 10. System checks for availability of chosen vehicle for the specified period 11. System displays that it is available 12. System prompts Renter to continue with booking 13. Renter confirms to continue with the booking 14. System prompts Renter to select pickup or delivery option 15. Renter selects pickup 16. System displays available pickup location of iCar Station 17. System prompts Renter to select an iCar Station location 18. Renter selects an iCar Station location 19. System validates that iCar Station location exist 20. System displays booking terms and conditions 21. System prompts Renter to confirm booking terms and conditions 22. Renter confirms booking terms and conditions 23. System displays Renter detailed cost summary 24. System prompts Renter to confirm detailed cost summary 25. Renter confirms detailed cost summary 26. System displays a prompt “thank you for booking, please proceed to payment” screen 27. Use case executes the Make Payment use case to proceed with the payment 28. System records payment is successful 29. System records the new booking record 30. Use case ends here |
|  | **AF1 - 15. Renter chooses doorstep delivery**  15.1 System prompts Renter for delivery address, date, and time  15.2 Renter input delivery address, date, and time  15.3 System records the new delivery record  15.4 Use case resumes at step 20  **AF2 - 28. Unsuccessful payment**  28.1 System records payment is unsuccessful  28.2 System displays unsuccessful message  28.3 System prompts Renter to try again  28.4 Use case resumes at step 27  *\*2 most significant alternate flows have been selected* |



Test Case(s):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sequence** | **Flow** |  |  |  |  |
| **1** | **Basic Flow** |  |  |  |  |
| **2** | **Basic Flow** | **AF 1** | **Basic Flow** |  |  |
| **3** | **Basic Flow** | **AF 1** | **Basic Flow** | **AF 2** | **Basic Flow** |
| **4** | **Basic Flow** | **AF 2** | **Basic Flow** |  |  |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC01 – Tan Guo Zhi Kelvin** |
| **Test Description** | **:** | This test case is to test if a Renter can book a vehicle. |
| **Use Case ID** | **:** | UC05 – Tan Guo Zhi Kelvin |
| **Use Case Name** | **:** | Book Vehicle |
| **Pre-Condition(s)** | **:** | User is logged in as Renter and their account status is “Active”. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Renter chooses to book a vehicle | N/A |  |
| **2.** | System prompts for start date & time | **2024-08-20 12:00** | System will prompt for start date & time |
| **3.** | System prompts for end date & time | **2024-08-21 12:00** | System will prompt for end date & time |
| **4.** | System prompts for brand & model of vehicle | **BMW i8** | System will prompt for brand & model of vehicle |
| **5.** | Renter chooses to continue with the booking | **yes** | System displays vehicle available for selected period and prompt Renter to continue with the booking |
| **6.** | System prompt Renter to choose pickup or delivery | **pickup** | System will prompt for pickup or delivery |
| **7.** | System prompt Renter for a pickup location | **S001** | System displays available pickup location and prompt Renter to select a location |
| **8.** | System prompt Renter to confirm Booking Terms and Conditions | **yes** | System displays Booking Terms and Conditions and prompt Reter to agree |
| **9.** | System prompt Renter to confirm detailed cost summary | **yes** | System displays detailed cost summary and prompt Renter to confirm it |
| **10.** | System displays proceed to payment screen | N/A | System confirm payment is successful and records the new booking |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC02 – Tan Guo Zhi Kelvin** |
| **Test Description** | **:** | This test case is to test if a Renter can book a vehicle using the delivery option. |
| **Use Case ID** | **:** | UC05 – Tan Guo Zhi Kelvin |
| **Use Case Name** | **:** | Book Vehicle |
| **Pre-Condition(s)** | **:** | User is logged in as Renter and their account status is “Active”. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Renter chooses to book a vehicle | N/A |  |
| **2.** | System prompts for start date & time | **2024-08-20 12:00** | System will prompt for start date & time |
| **3.** | System prompts for end date & time | **2024-08-21 12:00** | System will prompt for end date & time |
| **4.** | System prompts for brand & model of vehicle | **BMW i8** | System will prompt for brand & model of vehicle |
| **5.** | Renter chooses to continue with the booking | **yes** | System displays vehicle available for selected period and prompt Renter to continue with the booking |
| **6.** | System prompt Renter to choose pickup or delivery | **delivery** | System will prompt for pickup or delivery |
| **7.** | System prompt Renter for delivery details | Delivery Address: **123 test road, #08-112, Singapore 543123**  Delivery date & time: **2024-08-10 11:00** | System prompt Renter to enter delivery details and records the delivery |
| **8.** | System prompt Renter to confirm Booking Terms and Conditions | **yes** | System displays Booking Terms and Conditions and prompt Reter to agree |
| **9.** | System prompt Renter to confirm detailed cost summary | **yes** | System displays detailed cost summary and prompt Renter to confirm it |
| **10.** | System displays proceed to payment screen | N/A | System confirm payment is successful and records the new booking |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC03 – Tan Guo Zhi Kelvin** |
| **Test Description** | **:** | This test case is to test if a Renter can book a vehicle using the delivery option when payment status returns unsuccessful. |
| **Use Case ID** | **:** | UC05 – Tan Guo Zhi Kelvin |
| **Use Case Name** | **:** | Book Vehicle |
| **Pre-Condition(s)** | **:** | User is logged in as Renter and their account status is “Active”. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Renter chooses to book a vehicle | N/A |  |
| **2.** | System prompts for start date & time | **2024-08-20 12:00** | System will prompt for start date & time |
| **3.** | System prompts for end date & time | **2024-08-21 12:00** | System will prompt for end date & time |
| **4.** | System prompts for brand & model of vehicle | **BMW i8** | System will prompt for brand & model of vehicle |
| **5.** | Renter chooses to continue with the booking | **yes** | System displays vehicle available for selected period and prompt Renter to continue with the booking |
| **6.** | System prompt Renter to choose pickup or delivery | **delivery** | System will prompt for pickup or delivery |
| **7.** | System prompt Renter for delivery details | Delivery Address: **123 test road, #08-112, Singapore 543123**  Delivery date & time: **2024-08-20 11:00** | System prompt Renter to enter delivery details and records the delivery |
| **8.** | System prompt Renter to confirm Booking Terms and Conditions | **yes** | System displays Booking Terms and Conditions and prompt Reter to agree |
| **9.** | System prompt Renter to confirm detailed cost summary | **yes** | System displays detailed cost summary and prompt Renter to confirm it |
| **10.** | System displays proceed to payment screen | N/A |  |
| **11.** | System displays payment unsuccessful and prompt Renter to try again | **yes** | System returns payment unsuccessful and prompt Renter to try again |
| **12.** | System displays payment successful and records the new booking | N/A | System confirm payment is successful and records the new booking |

|  |  |  |
| --- | --- | --- |
| **Test Case ID** | **:** | **TC04 – Tan Guo Zhi Kelvin** |
| **Test Description** | **:** | This test case is to test if a Renter can book a vehicle when payment status returns unsuccessful. |
| **Use Case ID** | **:** | UC05 – Tan Guo Zhi Kelvin |
| **Use Case Name** | **:** | Book Vehicle |
| **Pre-Condition(s)** | **:** | User is logged in as Renter and their account status is “Active”. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequence** | **Scenario** | **Test Data** | **Expected Result** |
| **1.** | Renter chooses to book a vehicle | N/A |  |
| **2.** | System prompts for start date & time | **2024-08-20 12:00** | System will prompt for start date & time |
| **3.** | System prompts for end date & time | **2024-08-21 12:00** | System will prompt for end date & time |
| **4.** | System prompts for brand & model of vehicle | **BMW i8** | System will prompt for brand & model of vehicle |
| **5.** | Renter chooses to continue with the booking | **yes** | System displays vehicle available for selected period and prompt Renter to continue with the booking |
| **6.** | System prompt Renter to choose pickup or delivery | **pickup** | System will prompt for pickup or delivery |
| **7.** | System prompt Renter for a pickup location | **S001** | System displays available pickup location and prompt Renter to select a location |
| **8.** | System prompt Renter to confirm Booking Terms and Conditions | **yes** | System displays Booking Terms and Conditions and prompt Reter to agree |
| **9.** | System prompt Renter to confirm detailed cost summary | **yes** | System displays detailed cost summary and prompt Renter to confirm it |
| **10.** | System displays proceed to payment screen | N/A |  |
| **11.** | System displays payment unsuccessful and prompt Renter to try again | **yes** | System returns payment unsuccessful and prompt Renter to try again |
| **12.** | System displays payment successful and records the new booking | N/A | System confirm payment is successful and records the new booking |

# 4. Class Model

## 4.1 Design Class Diagram

Draw your domain class diagram.

A diagram of a computer flowchart

Description automatically generated

## 4.2 Classes

**Class Name – User**

User is an abstract class, with 3 subclasses of Renter, Car Owner and Admin

1. **Attributes:**
   1. **UserID : string**
      * Unique identifier that acts as primary key.
   2. **FullName : string**
      * Full legal name for User as per NRIC
   3. **ContactNumber : string**
      * Contact Number of User, specifically +65 XXXX XXXX, following Singaporean phone number
   4. **DateOfBirth : DateTime**
      * Date of Birth of User in the format of “DD/MM/YYYY”. Must be minimum of 18 years of age as of registration
   5. **Address : string**
      * Address of User, must be a Singaporean Address with postal code
   6. **ZipCode : string**
      * Zip Code of the user.
   7. **EmailAddress : string**
      * Email Address of User, must have a valid domain
   8. **Password : string**
      * Password for User login
   9. **DateJoin : DateTime**
      * Date and time of account creation
2. **Multiplicity:**
   1. **N/A**
3. **Methods:**
   1. **User()**
      * Empty protected constructor utilised by sub classes
   2. **User(fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, dateJoin : DateTime)**
      * Protected constructor utilised by sub classes.
      * Parameterised constructor to initialise a User object with the given parameters.
      * Parameters include:
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : DateTime**
   3. **RetrieveUser() [Yeo Jin Rong]**
      * Returns the current user object.
   4. **DisplayUserInfo() [Yeo Jin Rong]**
      * Abstract Method to be overridden in subclasses
      * Displays user information and returns the current user object.

**Class Name – Renter**

Renter is a subclass of the abstract User

1. **Attributes:**
   1. **RenterType : string**
      * renterType string can either be “Prime” or “Regular” to designate the membership type
   2. **BackgroundCheckStatus : string**
      * backgroundCheckStatus string can either be “Pending” when awaiting verification by admin or “Validified” after validation by admin
   3. **RegistrationStatus : string**
      * registrationStatus string can either be “Pending”, “Approved” or “Rejected”
      * If “Pending”, registration is still pending, which required Background Check, Driver’s Licence Validation and Email Verification. Please refer to section 1.4 for more details.
      * If “Approved”, registration has been fully completed and the Renter is approved for full access to the iCar rental platform
      * If “Rejected”, registration process was not successful, Renter is not allowed to obtain full access to the iCar rental platform
   4. **VerificationEmailSent : boolean**
      * verificationEmailSent is a boolean that can be either “True” or “False”. If “True”, an email with a unique 6-digit code for verification is successfully sent, if “False”, an email was not successfully sent. Refer to section 1.4 for more details
   5. **VerificationStatus : string**
      * verificationStatus is a string that can either be “Unverified” or “Verified”. If the user has successfully keyed in the 6-digit code during registration process, the string will be changed to ‘Verified’
2. **Multiplicity:**
   * + **Renter has Driver Licence**
       - 1 renter can only have 1 driver licence
3. **Methods:**
   1. **Renter()**
      * Empty Constructor
   2. **Renter(userID : string, fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, dateJoin : DateTime, renterType : string, backgroundCheckStatus : string, registrationStatus : string, verificationEmailSent : Boolean, verificationStatus : string) : base(userID : string, fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, dateJoin : DateTime)**
      * Parameterised Constructor for renter subclass
      * Initialises a new instance of the Renter class with the given parameters. Assigns a unique userID using Guid.NewGuid().
      * **DriverLicence** object within Renter set to null
      * **BookingsList** within Renter set to a new List of Booking Object(s)
      * Parameters included in subclass:
        + **userID : string**
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : DateTime**
        + **renterType : string**
        + **backgroundCheckStatus : string**
        + **registrationStatus : string**
        + **verificationEmailSent : Boolean**
        + **verificationStatus : string**
      * Parameters inherited from superclass (User):
        + **userID**
        + **fullName**
        + **contactNumber**
        + **dateOfBirth**
        + **address**
        + **zipCode**
        + **emailAddress**
        + **password**
        + **dateJoin**
   3. **RetrieveBookings() : List<Booking> [Liang Dingxuan]**
      * Returns the list of bookings for the renter.
   4. **CompleteBooking(aBooking : Booking) [Liang Dingxuan]**
      * Removes booking from the booking list once completed
   5. **DisplayUserInfo()**
      * Override method
      * Displays user information and returns the current user object.
   6. **CreateRenter(fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, renterType : string) [Ng Kai Huat Jason]**
      * Static Method
      * Initialises a new instance of the Renter class with parameters and returns the created Renter, assigns userID using Guid.NewGuid()
   7. **AddLicenceToRenter(aLicence : DriverLicence) [Ng Kai Huat Jason]**
      * Add a Driver Licence to a renter, returns void

**Class Name – Admin**

Admin is a subclass of the abstract User

1. **Attributes** 
   1. **Role : string**
2. **Multiplicity**
   1. **Admin to Renter:**
      * 1 Admin verifies 1 Renter
3. **Methods:**
   1. **Admin()** 
      * Empty Constructor
   2. **Admin(userID : string, fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string , password : string, dateJoin : DateTime, role : string) : base(userID : string, fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, dateJoin : DateTime)**
      * Parameterised constructor for admin subclass
      * Initialises a new instance of the Admin class with the given parameters. Assigns a unique userID using Guid.NewGuid().
      * Parameters included in subclass:
        + **userID : string**
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : Date**
        + **role : string**
      * Parameters inherited from superclass (User):
        + **userID : string**
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : Date**
   3. **DisplayUserInfo**
      * Override Method
      * Displays user information and returns the current user object.

**Class Name – CarOwner**

CarOwner is a subclass of the abstract User

1. **Attributes:**
   1. **ProjectedMonthlyRevenue : float**
      * Holds the total value of the earnings of the car owner
2. **Multiplicity:**
   1. **Car Owner to Vehicle:**
      * 1 CarOwner manages 0..\* Vehicle
3. **Methods:**
   1. **CarOwner() :**
      * Empty Constructor
   2. **CarOwner(userID : string, fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode : string, emailAddress : string, password : string, dateJoin : DateTime)** **: base(userID : string , fullName : string, contactNumber : string, dateOfBirth : DateTime, address : string, zipCode :string, emailAddress : string, password : string, dateJoin : DateTime)**
      * Parameterised constructor for CarOwner subclass
      * Initialises a new instance of the CarOwner class with the given parameters. Assigns a unique userID using Guid.NewGuid() and a Vehicles list.
      * **Vehicles** within Car Owner set to a new List of Vehicle Object(s)
      * Parameters included in sub class:
        + **userID : string**
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : DateTime**
      * Parameters inherited from superclass (User)
        + **userID : string**
        + **fullName : string**
        + **contactNumber : string**
        + **dateOfBirth : DateTime**
        + **address : string**
        + **zipCode : string**
        + **emailAddress : string**
        + **password : string**
        + **dateJoin : Date**
   3. **RegisterAVehicle(vehicle : Vehicle) [Yeo Jin Rong]**
      * Adds a vehicle to the Vehicles list and prints a success message.
   4. **DisplayAllVehicleInformation() [Yeo Jin Rong]**
      * Displays information for all registered vehicles. If no vehicles are registered, it prints a message indicating so.
   5. **DisplayVehicleById(vehicleId : string) [Yeo Jin Rong]**
      * Displays information for a specific vehicle by its ID. If the vehicle is not found, it prints a message indicating so.
   6. **DisplayUserInfo()**
      * Override Method
      * Displays user information and returns the current user object.

**Class Name – DriverLicence**

Class for Driver’s Licence, associated with Renter

1. **Attributes:**
   1. **LicenceID : string**
      * Unique identifier for Licence ID
   2. **LicenceNo : string**
      * Vehicle Licence Plate number (usually SBA XXXXX) for Singaporeans, following guidelines from LTA
      * <https://onemotoring.lta.gov.sg/content/onemotoring/home/buying/vehicle-types-and-registrations/car/standard_registration.html>
   3. **NumberOfDemeritPoints : int**
      * Number of demerit points on driver's licence
   4. **ClassOfLicence : string**
      * Class of driver's licence, such as "Class 3CA", as per Singapore Police Force advisory
      * <https://www.police.gov.sg/Advisories/Traffic/Traffic-Matters/Singapore-Driving-Licence>
   5. **DateOfIssue : DateTime**
      * Date of Issue of driver's licence, as stated on the licence itself. Date Of Issue must at least be 1 year before date of registration on iCar rental platform
   6. **CertificateOfMeritEligibility : boolean**
      * Eligibility for a Certificate of Merit, which are drivers who have maintained a continuous demerit-point-free driving record for the past 3 years. Saved as a Boolean for "True" or "False"
2. **Multiplicity:**
   * + **Renter to DriverLicence**
       - 1 Renter must have 1 Driver Licence
3. **Methods:**
   1. **DriverLicence()**
      * Empty Constructor
   2. **DriverLicence(licenceID : string, licenceNo : string, numberOfDemeritPoints : string, classOfLicence : string, dateOfIssue : DateTime, certificateOfMeritEligibility : boolean)**
      * Parameterised constructor for DriverLicence class
      * Initialises a new instance of the DriverLicence class with the given parameters.
      * Parameters include:
        + **licenceID : string**
        + **licenceNo : string**
        + **numberOfDemeritPoints : int**
        + **classOfLicence : string**
        + **dateofIssue : DateTime**
        + **certificateOfMeritEligibility : boolean**

* 1. **CreateLicence(renters : List<Renter>) [Ng Kai Huat Jason]**
     + Creates a licence object with parameters and returns it. Uses ValidateLicence to make sure licences are not duplicated before creation.
     + Returns a DriverLicence object
  2. **ValidateLicence(licenceNo : string, renters : List<Renter>) [Ng Kai Huat Jason]**
     + Static Method
     + This function is used to validate if there are any duplicate licences, if no duplicates, returns true else false

**Class Name – Vehicle**

1. **Attributes:**
   1. **VehicleID : string**
      * Unique Identifier for each vehicle registered by Car Owners
   2. **VehicleRegNo : string**
      * Unique Vehicle Registration Number registered by Car Owners
   3. **Make : string**
      * Name of Company/Manufacturer who made the Vehicle
   4. **Model : string**
      * Car Model Name of the Vehicle
   5. **RentalRate : float**
      * Rental rate amount set by the user
   6. **MaximumSeatingCapacity: int**
      * Rental rate amount set by the user
   7. **Year : int**
      * The year of manufacturing of the Vehicle
   8. **Mileage : float**
      * The Total Distance travelled by the car
   9. **IsVerified : boolean**
      * Boolean value to record if the vehicle has been inspected, thus verified by an Administrator
2. **Multiplicity:**
   1. **CarOwner to Vehicle:**
      * 1 CarOwner can register 0..\* Vehicle
3. **Methods:**
   1. **Vehicle()**
      * Empty Constructor.
   2. **Vehicle(vehicleRegNo : string, make : string, model : string, rentalRate : float, maxSeatingCapacity : int, year : int, mileage : float)** **[Yeo Jin Rong]**
      * Parameterised Constructor for Vehicle Class
      * Initialises a new instance of the Vehicle class with the given parameters. Assigns a unique VehicleID using Guid.NewGuid().
      * **FuelType** within Vehicle is set to null
      * **InsuranceDetails** within Vehicle is set to null
      * **Availability** within Vehicle is set to null
      * **Photos** within Vehicle is set to a List of Photo object(s)
      * **BookingsList** within Vehicle is set to a List of Booking object(s)
      * **AccidentReports** within Vehicle is set to a List of AccidentReport object(s)
      * **InspectionReports** Within is set to a List of InspectionReport object(s)
      * Parameters include:
        + **vehicleRegNo : string**
        + **make : string**
        + **model : string**
        + **rentalRate : float**
        + **maxSeatingCapacity : int**
        + **year : int**
        + **mileage : float**
   3. **CreateVehicle() : Vehicle [Yeo Jin Rong]**
      * Static Method
      * Prompts the user to input vehicle details, validates the input, and creates a new Vehicle instance with the provided details.
   4. **AddFuelType()** **[Yeo Jin Rong]**
      * Prompts the user to select and input details for a fuel type (Gas, Electric, or Hybrid) and assigns it to the vehicle.
   5. **SetFuelType(fuelType : FuelType)** **[Yeo Jin Rong]**
      * Sets the FuelType property for the vehicle.
   6. **ValidateVehicle() : boolean** **[Yeo Jin Rong]**
      * Validates the vehicle's properties (e.g., registration number, make, model, rental rate, seating capacity, year, mileage). Includes specific validation for fuel type properties based on the type of fuel (Gas, Electric, Hybrid).
   7. **ValidateBookedVehicle (vehicles : List<Vehicle>, vehicleAvailabilities : List<VehicleAvailability>, make : string, model : string, startDateTime : DateTime, endDateTime : DateTime, out selectedVehicle : Vehicle) [Tan Guo Zhi Kelvin]**
      * Static Method
      * Checks if a vehicle of the specified make and model is available for the given date/time range, returning true and setting the selected vehicle if available, or false if not.
   8. **AddPhoto(photo : Photo)** **[Yeo Jin Rong]**
      * Adds a photo to the vehicle's Photos list. Prints a message indicating success or failure.
   9. **AddInsurance(insurance : Insurance)** **[Yeo Jin Rong]**
      * Assigns insurance details to the vehicle.
   10. **AddVehicleAvailability(vehicleAvailability : VehicleAvailability)** **[Yeo Jin Rong]**
       * Assigns availability details to the vehicle.
   11. **DisplayVehicleInformation()** **[Yeo Jin Rong]**
       * Displays all relevant information about the vehicle, including fuel type details, insurance details, availability details, and photos.

**Class Name – FuelType**

Abstract class for Fuel Type, with subclasses Hybrid, Electric and Gas

1. **Attributes:**
   1. **N/A**, due to abstract class, no attributes are inherited
2. **Multiplicity:**
   1. 1 Fuel Type belongs to 1 Vehicle
3. **Methods:**
   1. **FuelType()** 
      * Protected Empty constructor, utilised by for subclasses.
   2. **DisplayFuelTypeInformation() [Yeo Jin Rong]**
      * Abstract method to be overridden in subclasses
   3. **CreateFuelType() [Yeo Jin Rong]**
      * Abstract method to be overridden in subclasses

**Class Name – Gas : FuelType**

Subclass of FuelType

1. **Attributes:**
   1. **PreferredPetrolKiosk: string**
      * Car Owner’s preferred petrol kiosk. Examples are Shell, Esso, Caltex, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   2. **PetrolType : string**
      * The vehicle’s type of petrol utilised, predominantly either octane-92, octane-95, octane-98 or specific branded premium alternatives. Examples include Shell FuelSave 95, Esso Synergy Supreme+, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   3. **VehicleMaximumFuelTank : float**
      * The vehicle’s total maximum fuel capacity in litres. This is to provide and approximation for cost of refueling if necessary.
2. **Multiplicity:**
   1. **Fuel Type to Vehicle**
      * 1 Fuel Type belongs to 1 Vehicle
3. **Methods:**
   1. **Gas()**
      * Empty Constructor
   2. **Gas(preferredPetrolKiosk : string, petrolType : string, vehicleMaxFuelCapacity : float)** 
      * Parameterised constructor for Vehicle with gas fuel type.
      * Initialises a new instance of the Gas class with the given parameters.
      * Parameters included in subclass:
        + **preferredPetrolKiosk : string**
        + **petrolType : string**
        + **vehicleMaxFuelCapacity : float**
   3. **DisplayFuelTypeInformation()** **[Yeo Jin Rong]**
      * Override Method
      * Displays information about the gas fuel type, including preferred petrol kiosk, petrol type, and vehicle max fuel capacity.
   4. **CreateFuelType()** **[Yeo Jin Rong]**
      * Override Method
      * Prompts the user to input details specific to gas fuel type: preferred petrol kiosk, petrol type, and vehicle max fuel capacity.

**Class Name – Electric : FuelType**

Subclass of FuelType

1. **Attributes:**
   1. **PreferredChargingStation : string**
      * Car Owner’s preferred brand or type of charging station. Examples are Shell Recharge, Bluecharge, CDG EngiePetrol, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   2. **VehicleCurrentType : string**
      * The vehicle’s charging current type, either as “Alternate” or “Direct”, no other inputs are accepted.
   3. **VehicleChargingRate : float**
      * The vehicle’s recommended charging rate as per manufacturer's instructions, to prevent unintentional overcharge during rental and storage, reducing probability of battery damage for rental vehicles.
   4. **VehicleMaxChargeCapacity: float**
      * Maximum capacity of battery for vehicle, to have an approximation of charging time required as well as to prevent overcharging and unintentional battery damage to the vehicle
2. **Multiplicity:**
   1. **Fuel Type To Vehicle** 
      * 1 Fuel Type belongs to 1 Vehicle
3. **Methods:**
   1. **Electric()**
      * Empty Constructor
   2. **Electric (preferredChargingStation : string, vehicleCurrentType : string, vehicleChargingRate : float, vehicleMaxChargeCapacity: float)**
      * Parameterised constructor for vehicles with electric fuel type.
      * Initialises a new instance of the Electric class with the given parameters.
      * Parameters included in subclass:
        + **preferredChargingStation : string**
        + **vehicleCurrentType : string**
        + **vehicleChargingRate : float**
        + **vehicleMaxChargeCapacity : float**
   3. **DisplayFuelTypeInformation()** **[Yeo Jin Rong]**
      * Override Method
      * Displays information about the electric fuel type, including preferred charging station, vehicle current type, vehicle charging rate, and vehicle max charge capacity.
   4. **CreateFuelType()** **[Yeo Jin Rong]**
      * Override Method
      * Prompts the user to input details specific to electric fuel type: preferred charging station, vehicle current type, vehicle charging rate, and vehicle max charge capacity.

**Class Name – FuelType : Hybrid**

Subclass of FuelType

1. **Attributes:**
   1. **PreferredPetrolKiosk: string**
      * Car Owner’s preferred petrol kiosk. Examples are Shell, Esso, Caltex, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   2. **PetrolType : string**
      * The vehicle’s type of petrol utilised, predominantly either octane-92, octane-95, octane-98 or specific branded premium alternatives. Examples include Shell FuelSave 95, Esso Synergy Supreme+, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   3. **VehicleMaxFuelCapacity: float**
      * The vehicle’s total maximum fuel capacity in litres. This is to provide and approximation for cost of refuelling if necessary.
   4. **PreferredChargingStation : string**
      * Car Owner’s preferred brand or type of charging station. Examples are Shell Recharge, Bluecharge, CDG EngiePetrol, etcetera. These examples are not exhaustive of what is available in a Singaporean Context
   5. **VehicleCurrentType : string**
      * The vehicle’s charging current type, either as “Alternate” or “Direct”, no other inputs are accepted.
   6. **VehicleChargingRate : float**
      * The vehicle’s recommended charging rate as per manufacturer's instructions, to prevent unintentional overcharge during rental and storage, reducing probability of battery damage for rental vehicles.
   7. **VehicleMaxChargeCapacity: float**
      * Maximum capacity of battery for vehicle, to have an approximation of charging time required as well as to prevent overcharging and unintentional battery damage to the vehicle
2. **Multiplicity:**
   1. **Fuel Type to Vehicle**
      * 1 Fuel Type belongs to 1 Vehicle
3. **Methods:**
   1. **Hybrid()**
      * Empty Constructor
   2. **Hybrid(preferredPetrolKiosk : string, petrolType : string, vehicleMaxFuelCapacity : float, preferredChargingStation : string, vehicleCurrentType : string, vehicleChargingRate : float, vehicleMaxChargeCapacity : float)**
      * Parameterised constructor for vehicle with hybrid fuel type.
      * Initialises a new instance of the Hybrid class with the given parameters.
      * Parameters included in subclass:
        + **preferredPetrolKiosk : string**
        + **petrolType : string**
        + **vehicleMaxFuelCapacity : float**
        + **preferredChargingStation : string**
        + **vehicleCurrentType : string**
        + **vehicleChargingRate : float**
        + **vehicleMaxChargeCapacity : float**
   3. **DisplayFuelTypeInformation()** **[Yeo Jin Rong]**
      * Override Method
      * Displays information about the hybrid fuel type, combining details from both gas and electric fuel types: preferred petrol kiosk, petrol type, vehicle max fuel capacity, preferred charging station, vehicle current type, vehicle charging rate, and vehicle max charge capacity.
   4. **CreateFuelType()** **[Yeo Jin Rong]**
      * Override Method
      * Prompts the user to input details specific to hybrid fuel type, including details for both gas and electric components: preferred petrol kiosk, petrol type, vehicle max fuel capacity, preferred charging station, vehicle current type, vehicle charging rate, and vehicle max charge capacity.

**Class Name – Insurance**

1. **Attributes:**
   1. **InsuranceID : string**
      * Unique Identifier for insurance that is covering a particular vehicle
   2. **CoverageDetails : string**
      * Description of the insurance
   3. **InsuranceStartDate : DateTime**
      * Date of when insurance coverage starts
   4. **InsuranceEndDate : DateTime**
      * Date of when insurance coverage ends
2. **Multiplicity:**
   1. **Insurance to Vehicle:**
      * 1 Vehicle covered by 1 Insurance
3. **Methods:**
   1. **Insurance()**
      * Empty Constructor
   2. **Insurance(insuranceID : string, coverageDetails : string, insuranceStartDate : StartDate, insuranceEndDate : EndDate)**
      * Parameterised constructor for Insurance Class
      * Initialises a new instance of the Insurance class with the given parameters.
      * Parameters include:
        + **insuranceID : string**
        + **coverageDetails : string**
        + **insuranceStartDate : StartDate**
        + **insuranceEndDate : EndDate**
   3. **CreateInsurance() : Insurance [Yeo Jin Rong]**
      * Static Method
      * Prompts the user to input insurance details (ID, coverage details, start date, and end date). Returns a new Insurance instance with the provided details, or null if canceled.
   4. **ValidateInsurance() : boolean [Yeo Jin Rong]**
      * Validates the InsuranceID (it must not be empty) and ensures that the InsuranceEndDate is after the InsuranceStartDate. Returns true if the insurance details are valid; otherwise, returns false.
   5. **DisplayInsuranceInformation() [Yeo Jin Rong]**
      * Displays all relevant information about the insurance, including InsuranceID, CoverageDetails, InsuranceStartDate, and InsuranceEndDate.

**Class Name – VehicleAvailability**

1. **Attributes:**
   * + **AvailabilityID : string**
2. **Multiplicity:**
   1. **VehicleAvailability to Vehicle:**
      * **1 Vehicle has 1 VehicleAvailability**
3. **Methods:**
   1. **VehicleAvailability()**
      * Empty Constructor
   2. **VehicleAvailability(availabilityID, : string, availabilitySlots : List <AvailabilitySlot> availabilitySlots)** 
      * Parameterised constructor for VehicleAvailability
      * Initialises a new instance of the VehicleAvailability class with a unique AvailabilityID, empty list of AvailabilitySlots and given parameters.
      * Parameters include:

* **availabilityID : string**
* **availabilitySlots : List <AvailabilitySlot> availabilitySlots**
  1. **CreateVehicleAvailability() : VehicleAvailability [Yeo Jin Rong]**
     + Static Method
     + Creates and returns a new instance of the VehicleAvailability class with a unique AvailabilityID and an empty list of AvailabilitySlots.
  2. **AddAvailabilitySlot(slot : AvailabilitySlot) [Yeo Jin Rong]**
     + Adds a validated AvailabilitySlot to the AvailabilitySlots list. Prints a message indicating success or failure.
  3. **DisplayAvailabilityInformation() [Yeo Jin Rong]**
     + Displays all relevant information about the vehicle's availability, including the AvailabilityID and details of each AvailabilitySlot.
  4. **CheckAvailability(availabilities : List<VehicleAvailability>, startDateTime : DateTime, endDateTime : DateTime) [Tan Guo Zhi Kelvin]**
     + Static Method
     + Validates the date/times and checks if any availability slots cover the requested period, returning true if available and false otherwise.
     + Parameters include:
       - **availabilities : List<VehicleAvailability>**
       - **startDateTime : DateTime**
       - **endDateTime : DateTime**
  5. **ValidateAvailabilitySlots(startDateTime : DateTime, endDateTime : DateTime) [Tan Guo Zhi Kelvin]**
     + Static Method
     + Validates if the start time is before the end time and at least 24 hours from the current time, returning true if valid and false otherwise.
     + Parameters include:
       - **startDateTime : DateTime**
       - **endDateTime : DateTime**

**Class Name – AvailabilitySlot**

1. **Attributes:**
   1. **AvailabilitySlotID : string**
      * Unique identifier for each availability slot.
   2. **StartTime : DateTime**
      * Start time of the slot in YYYY-MM-DD HH:MM format  
        (example: 2024-08-02 14:30)
   3. **EndTime : DateTime**
      * End time of the slot in YYYY-MM-DD HH:MM format  
        (example: 2024-08-02 14:30)
2. **Multiplicity:**
   1. **AvailabilitySlot to VehicleAvailability:**
      * 0..\* AvailabilitySlot has 1 VehicleAvailability
3. **Methods:**
   1. **AvailabilitySlot()**
      * Empty Constructor
   2. **AvailabilitySlot(availabilitySlotID: string, startTime : DateTime, endTime : DateTime)**
      * Parameterised Constructor for AvailabilitySlot object
      * Initialises a new instance of the AvailabilitySlot class with the given parameters.
      * Parameters include:
        + **slotID : string**
        + **startTime : DateTime**
        + **endTime : DateTime**
   3. **CreateAvailabilitySlot()** **[Yeo Jin Rong]**
      * Static Method
      * Prompts the user to enter start and end times. Creates and returns a new AvailabilitySlot with a unique ID and the provided times.
   4. **ValidateAvailabilitySlot()** **[Yeo Jin Rong]**
      * Validates the availability slot
      * Start time is before end time.
      * Start time is at least 24 hours from the current time. Returns true if valid; otherwise, prints an error message and returns false.

**Class Name – Photo**

1. **Attributes:**
   1. **PhotoID : string**
2. **Multiplicity:**
   1. **Photo to Vehicle**
      * **1 Vehicle may have 0..\* Photo**
3. **Methods:**
   1. **Photo()**
      * Empty Constructor
   2. **Photo(photoID : string)**
      * Parameterised Constructor for photo class
      * Initialises a new instance of the Photo class. It has a property PhotoID for storing the photo's unique identifier.
      * Parameters include:
        + **photoID : string**
   3. **CreatePhoto() [Yeo Jin Rong]**
      * Static Method
      * Prompts the user to input a PhotoID, validates the input, and creates a new Photo instance with the provided PhotoID. Returns the Photo instance if the PhotoID is valid; otherwise, it returns null and prints an error message.

**Class Name – Reviews**

1. **Attributes:**
   1. **ReviewID : string**
      * Unique identifier for review, acts as primary key
   2. **StarInt : int**
      * Number of stars given out of 5
   3. **ReviewDesc : string**
      * Description provided for review
2. **Multiplicity:**
   1. **Booking to Reviews:**
      * 1 Booking can receive 0..1 Reviews
   2. **Vehicle to Reviews:**
      * 1 Vehicle can have 0..\* Reviews
3. **Methods:**
   1. **Reviews()**
      * Empty Constructor
   2. **Reviews(reviewID : string, starInt : int, reviewDesc : string)**
      * Parameterised constructor for Reviews class
      * Initialises a new instance of the Reviews class with the given parameters.
      * Parameters include:
        + **reviewID : string**
        + **starInt : int**
        + **reviewDesc : string**

**Class Name – InspectionReport**

1. **Attributes:**
   1. **InspectionID : string**
      * Unique identifier for Inspection Report
   2. **ReportDetails : string**
      * Details of inspection
   3. **InspectionDateTime : DateTime**
      * Date and time of Inspection
   4. **InspectorName : string**
      * Name of the inspection
   5. **Status : string**
      * Status of inspection
2. **Multiplicity:**
   1. **InspectionReport To Vehicle**
      * 1 Vehicle is inspected with 1..\* InspectionReport
3. **Methods:**
   1. **InspectionReport()**
      * Empty Constructor
   2. **InspectionReport(inspectionID : string, reportDetails : string, inspectionDateTime : DateTime, inspectorName : string, status : string)**
      * Parameterised constructor for InspectionReport class.
      * Initialises a new instance of the InspectionReport class with the given parameters.
      * Parameters include:
        + **inspectionID : string**
        + **reportDetails : string**
        + **inspectionDateTime : DateTime**
        + **inspectorName : string**
        + **status : string**

**Class Name – Booking**

Booking is an aggregation with Location, and it has associations with Renter, Vehicle, and Payment.

1. **Attributes:**
   1. **BookingID : string**
      * Unique identifier for each booking made
   2. **BookingDateTime : datetime**
      * The date that the booking was made
   3. **BookingStatus : string**
      * The status of the booking (e.g. pending, confirmed, cancelled, failed, completed)
   4. **PenaltyFee : float**
      * The penalty fee incurred (if any)
   5. **RentalFee : float**
      * The rental fee of booked vehicle
   6. **BookedAvailabilitySlot: AvailabilitySlot**
      * The specific time period selected for the car's availability, indicating when the car can be rented or is available for booking.
2. **Multiplicity:**
   1. **Booking aggregation to Location:**
      * 1 Booking contains 1..2 Location
   2. **Booking to Renter:**
      * 1 Renter makes 1..\* Booking
   3. **Booking to Vehicle:**
      * 1 Vehicle has 0..\* Booking
3. **Methods:**
   1. **Booking()**
      * Empty Constructor
   2. **Booking(bookingID : string, bookingDateTime : DateTime, bookingStatus : string, penaltyFee : float, rentalFee : float, bookedAvailabilitySlot : AvailabilitySlot, pickUpLocation : Location, returnLocation : Location) : Constructor**
      * Parameterised Constructor for Booking Class
      * Initialises a new Booking object with the given parameters.
      * Vehicle object within Booking is set to null
      * Reviews object within Booking is set to null
      * Payment object within Booking is set to null
      * AccidentReport object within Booking is set to null
      * Parameters:
        + **bookingID : string**
        + **bookingDate : DateTime**
        + **vehicle : Vehicle**
        + **bookingStatus : string**
        + **penaltyFee : float**
        + **rentalFee : float**
        + **availabilitySlot : AvailabilitySlot**
   3. **GetDetails() : [Liang Dingxuan]**
      * Returns the details of the booking as a String.
   4. **SetReturnLocation(location : Location) [Liang Dingxuan]**
      * Sets the location for the vehicle return.
   5. **ReturnVehicle() [Liang Dingxuan]**
      * Marks the booking as returned.

**Class Name – Location**

Location is an abstract class for subclass iCarStation and subclass DeliveryService

1. **Attributes:**
   1. **Address: string**
      * Address of either iCarStation or Specific Location
   2. **PickupDateTime : DateTime**
      * Date and Time of pickup
   3. **ReturnDateTime : DateTime**
      * Date and Time of return
2. **Abstract Class:**
   1. **Location generalisation to iCarStation and DeliveryService:**
      * A location is an abstract class and cannot be directly instantiated, must be explicitly downcasted to subclass iCarStation or DeliveryService to be instantiated.
3. **Methods:**
   1. **Location()**
      * Empty protected constructor utilised by subclasses
   2. **Location(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime)**
      * Protected parameterised constructed utilised by subclasses
      * Parameters includes:
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
   3. **GetLocation() [Liang Dingxuan]**
      * Abstract method to be overridden in subclasses

**Class Name – iCarStation**

iCarStation is a specialisation of Location (a.k.a subclass iCarStation of abstract class Location)

1. **Attributes:**
   1. **StationID : string**
      * Unique identifier for iCarStation
   2. **StationName : string**
      * The iCarStation name
   3. **OperatingHrs : string**
      * The operating hours of the iCarStation
   4. **ContactInfo : string**
      * The contact info for the iCarStation (usually a phone number in Singapore format of +65 xxxx xxxx)
2. **Multiplicity:**
   1. **N/A**
3. **Methods:**
   1. **iCarStation()**
      * Empty Constructor
   2. **iCarStation( address : string, pickUpDateTime : DateTime, returnDateTime : DateTime, stationID : string, stationName : string, operatingHrs : string, contactInfo : string) : base(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime)**
      * Parameterised constructor for iCarStation subclass
      * Initialises a new instance of the iCarStation object with the given parameters.
      * Parameters included in subclass
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
        + **stationID : string**
        + **stationName : string**
        + **operatingHrs : string**
        + **contactInfo : string**
      * Parameters inherited from superclass (Location)
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
   3. **ValidatePickupLocation(iCarStations : List<ICarStation>) [Tan Guo Zhi Kelvin]**
      * Validates the entered Station ID and returns the corresponding iCar station if found, otherwise returns null.
      * Parameters include:
        + **iCarStations : List<ICarStation>**
   4. **GetLocation() [Liang Dingxuan]**
      * Override method returns location information on the iCar station as a string

**Class Name – DeliveryService**

DeliveryService is a specialisation of Location (a.k.a subclass DeliveryService of abstract class Location)

1. **Attributes:**
   1. **DeliveryID : string**
      * Unique identifier for the delivery
   2. **DeliveryDateTime : datetime**
      * The date and time of the delivery
   3. **DeliveryFee : float**
      * The delivery fee amount
   4. **DeliveryStatus : string**
      * The status of the delivery (e.g. delivered, pending, in transit, scheduled, cancelled, failed)
2. **Multiplicity:**
   1. **N/A**
3. **Methods:**
   1. **DeliveryService()** 
      * **Empty Constructor**
   2. **DeliveryService(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime, deliveryID : string, deliveryDateTime : DateTime, deliveryFee : float, deliveryStatus : string) : base(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime)**
      * Parameterised constructor for DeliveryService subclass
      * Initialises a new instance of the DeliveryService object with the given parameters.
      * Parameters included in subclass
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
        + **deliveryID : string**
        + **deliveryDateTime : DateTime**
        + **deliveryFee : float**
        + **deliveryStatus : string**
      * Parameters inherited from superclass (Location)
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
   3. **DeliveryService(address : string, deliveryID : string, deliveryDateTime : DateTime, deliveryStatus : string, pickUpDateTime : DateTime, returnDateTime : DateTime) : base(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime)**
      * Parameterised constructor for DeliveryService subclass
      * Initialises a new instance of the DeliveryService object with the given parameters.
      * Parameters included in subclass
        + **address : string**
        + **deliveryID : string**
        + **deliveryDateTime : DateTime**
        + **deliveryStatus : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
      * Parameters inherited from superclass (Location)
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
   4. **DeliveryService(deliveryID : string, deliveryDateTime : DateTime, deliveryFee : float, deliveryStatus : string, address : string) : base(address : string, pickUpDateTime : DateTime, returnDateTime : DateTime)**
      * Parameterised constructor for DeliveryService subclass
      * Initialises a new instance of the DeliveryService object with the given parameters.
      * Parameters included in subclass
        + **deliveryID : string**
        + **deliveryDateTime : DateTime**
        + **deliveryFee : float**
        + **deliveryStatus : string**
        + **address :string**
      * Parameters inherited from superclass (Location)
        + **address : string**
        + **pickUpDateTime : DateTime**
        + **returnDateTime : DateTime**
   5. **GetLocation() [Liang Dingxuan]**
      * Override method Returns Location information on the iCar station as a string
   6. **RecordDeliveryDetails(deliveries : List<DeliveryService>, address : string, deliveryDateTime : DateTime) [Tan Guo Zhi Kelvin]**
      * Static Method
      * Records a new delivery with the specified details and adds it to the list of deliveries.
      * Parameters include:
        + **deliveries : List<DeliveryService>**
        + **address : string**
        + **deliveryDateTime : DateTime**

**Class Name – AccidentReport**

1. **Attributes:**
   1. **ReportID : string**
      * Unique identifier for report
   2. **AccidentDateTime : DateTime**
      * Date and time stored
   3. **AccidentDesc : string**
      * Description of what happened in the accident
   4. **AccidentLocation: string**
      * Location of which the accident occurred.
2. **Multiplicity:**
   1. **Accident Report To Booking**
      * 1 Booking may have 0..1 Accident Report
3. **Methods:**
   1. **Accident()**
      * Empty Constructor
   2. **AccidentReport(reportID : string, accidentDateTime : DateTime, accidentDescription : string, accidentLocation : string)**
      * Parameterised constructor for AccidentReport.
      * Initialises a new instance of the AccidentReport object with given parameters.
      * Parameters included:
        + **reportID : string**
        + **accidentDateTime : string**
        + **accidentDescription : string**
        + **accidentLocation : string**
   3. **ReportAccident(damages : string, location : string, datetime : DateTime) [Liang Dingxuan]**
      * Returns the accident object with updated attributes; damages, Location and datetime

**Class Name – Payment**

1. **Attributes:**
   1. **PaymentID : string**
      * Unique identifier for each payment transaction, acts as primary key
   2. **Amount : float**
      * Cost associated with the car rental booking
   3. **PaymentMethod : string**
      * Payment method utilized in payment (Credit/Debit Card, Digital Wallet or PayNow)
   4. **PaymentStatus : string**
      * Status of payment transaction, set to “Pending” or “Confirmed”
2. **Multiplicity:**
   1. **Booking to Payment:**
      * 1 Booking has 1..2 Payment (due to rental cost and potential penalty fee)
   2. **DiscountCode to Payment:**
      * 1 Booking uses 0..1 discountCode
3. **Methods:**
   1. **Payment()**
      * Empty Constructor
   2. **Payment (amount : float, paymentMethod : string, paymentStatus : string) [Ng Jing Zhan Garrett]**
      * Parameterised Constructor for Payment Class
      * Initialises Payment() with the given parameters. Assigns a unique PaymentID using Guid.NewGuid()
      * Parameters includes:
        + **amount : float**
        + **paymentMethod : string**
        + **paymentStatus : string**
   3. **InitializePayment(amount : float, paymentMethod : string)**

**[Ng Jing Zhan Garrett]**

* + - Creates a payment object with the parameters
    - Returns payment object
  1. **UpdatePaymentStatus(newStatus : string) [Ng Jing Zhan Garrett]**
     + Updates PaymentStatus when payment is validated and confirmed.
  2. **UpdatePaymentMethod(newMethod : string) [Ng Jing Zhan Garrett]**
     + Updates PaymentMethod of use when payment is validated and confirmed.

**Class Name - DiscountCode**

1. **Attributes:**
   1. **DiscountID : string**
      * Unique identifier for discount code, acts as primary key
   2. **DiscCode : string**
      * Unique 6-character string for Renter to key into system
   3. **DiscountType : string**
      * Type of discount (Fixed or Percentage)
   4. **DiscountValue : float**
      * Numerical value for discount
2. **Multiplicity:**
   1. **DiscountCode to Payment:**
      * 1 Booking uses 0..1 discountCode
3. **Methods:**
   1. **DiscountCode()**
      * Empty Constructor
   2. **DiscountCode (discountID : string, discCode : string, discountType : string, discountValue : float)**

**[Ng Jing Zhan Garrett]**

* + - Parameterised constructor for DiscountCode class
    - Initialises DiscountCode object with the given parameters. Assigns a unique DiscountID using Guid.NewGuid().
    - Parameters includes:
      * **discountID : string**
      * **discCode : string**
      * **discountType : string**
      * **discountValue : float**
  1. **ApplyDiscount(amount : float) (Ng Jing Zhan Garrett)**
     + Applies discount from use of discount code based on DiscountType = “Percentage” or “Flat”

# 5. Interaction Model

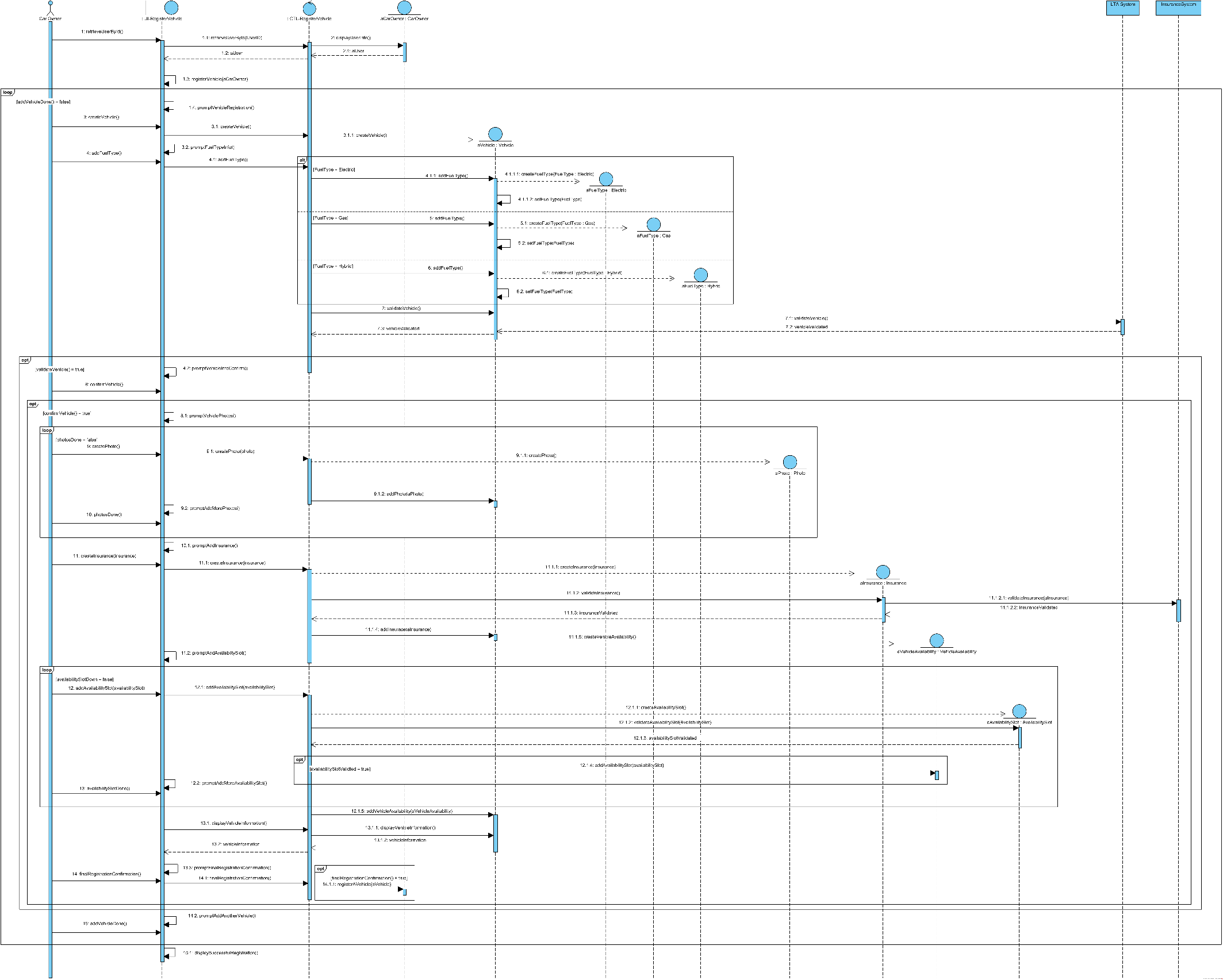
The student’s name and id should be clearly indicated for each sequence diagram.   
Sequence diagram is only required for Assignment 2.

## 5.1 Sequence Diagram by Yeo Jin Rong

Student Name: Yeo Jin Rong

Student ID: S10258457

**Use Case: Register Vehicle**



## 5.2 Sequence Diagram by Ng Jing Zhan Garrett

Student Name: Ng Jing Zhan Garrett

Student ID: S10257347

**Use Case: Make Payment**

A blue and white diagram

Description automatically generated

**Use Case: Pay By Debit/Credit Card**

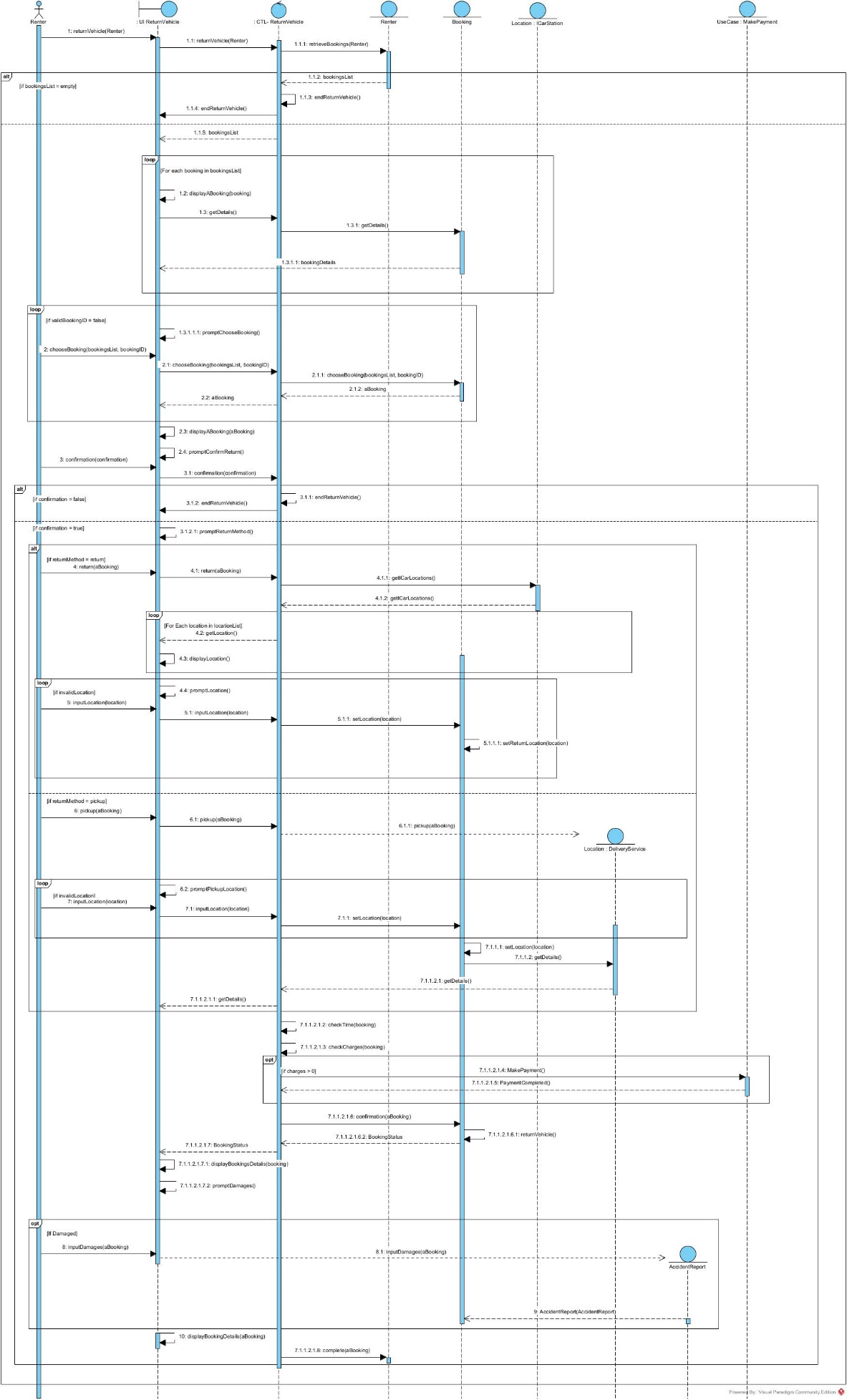
**A diagram of a diagram

Description automatically generated**

## 5.3 Sequence Diagram by Liang Dingxuan

Student Name: Liang Dingxuan

Student ID: S10258272



## 5.4 Sequence Diagram by Ng Kai Huat Jason

Student Name: Ng Kai Huat Jason

Student ID: S10262552

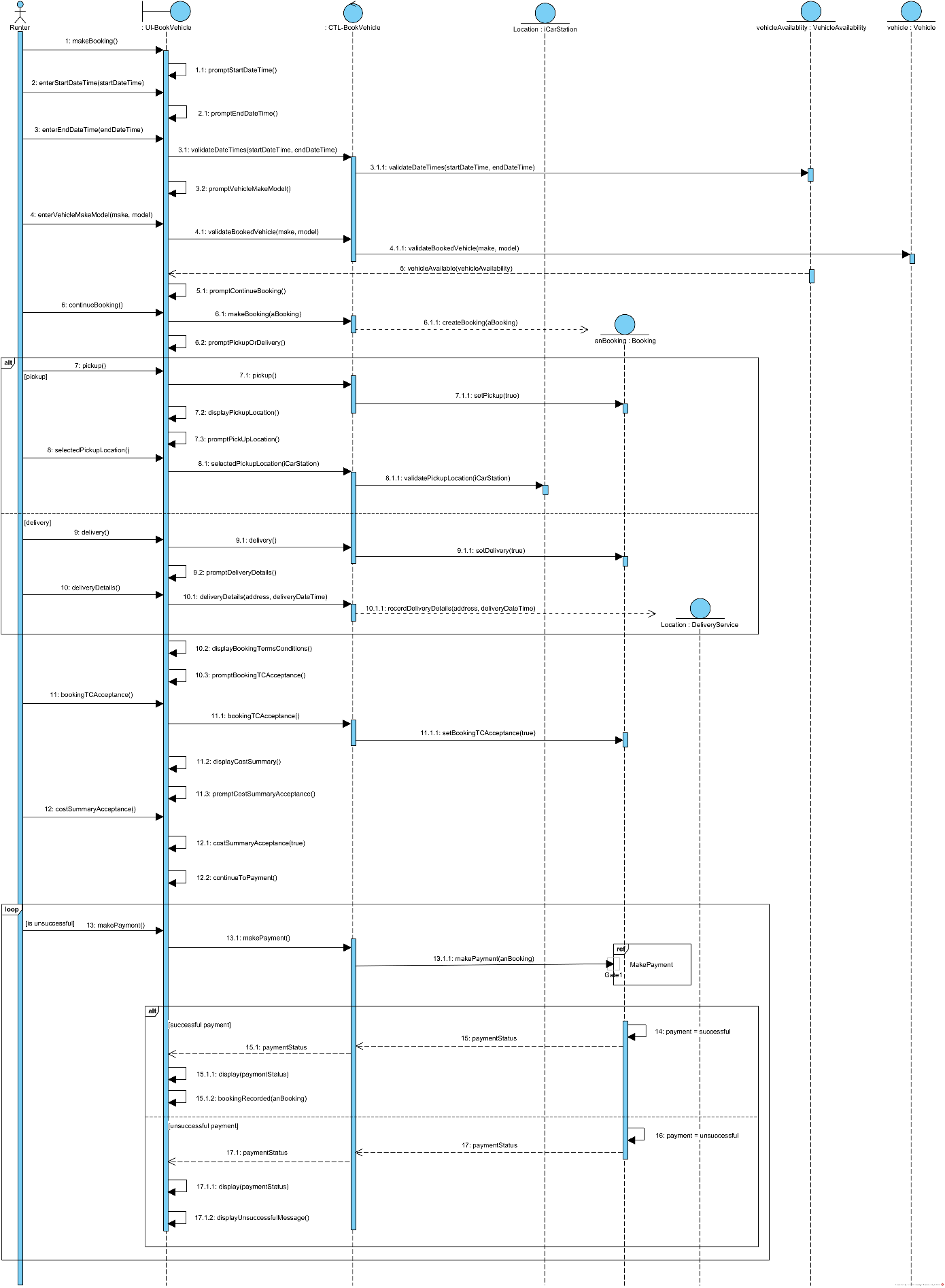
A blue lines on a white background

Description automatically generated

## 5.5 Sequence Diagram by Tan Guo Zhi Kelvin

Student Name: Tan Guo Zhi Kelvin

Student ID: S10262567

**Use Case: Book Vehicle**

# Appendices

# Appendix A – Other Tools Used

Include any GUI of prototypes, test cases, etc. produced during analysis and design of the project.

You should make appropriate references to this appendix from Section 2 – General Project Description