# Software Requirements Specification

Version 2.0

for

# Star Car Rentals

Prepared by

| Name | Student ID | Email |
|---|---|---|
| Adeyinka Areje (Team Lead) | 40088288 | adeyinka.areje@icloud.com |
| Vaibhav Malhotra | 40079373 | malhotra.vaibhav0304@gmail.com |

| | |
|---|---|
| Instructor: | ***Dr. C. Constantinides*** |
| Course: | SOEN 6461 |
| Date: | 8th November 2019 |

**Document history**

| Date | Version | Description | Author |
|---|---|---|---|
| 13th Nov 2019 | 1.0 | | Team 3 |
| 24th Nov 2019 | 2.0 | | Team 3 |
| | | | |

**Table of contents**

**List of figures**

## 1. Introduction

This document describes a Car Rental system that is to be developed and all the requirements (functional and non functional) that will be met to ensure acceptability and ensure continuity.

**Purpose**

The purpose of this document is to present a detailed description of our application. Specific requirements , use cases , system constraints and analysis of the software models will be done in order to achieve a well developed system.

In the first section, we describe the product's perspective, its functions, the user characteristics, system constraints, assumptions and dependencies. In the second section, we investigate requirements of the system like the external interface and functional and non-functional requirements.

Target audience: The document is targeting Dr. Constantinos Constantinides and his TAs as an educational project.

**Scope**

This web application will be used by a car rental company: Star Car Rental to make rentals and create vehicle reservations for the clients. The car rental system enables the clerks to rent a vehicle to a client, perform a vehicle return or create and cancel reservations of vehicles for a client, view and search catalog. An admin will manage the vehicle catalog.

This SRS is also aimed at specifying requirements of software to be developed but it can also be applied to assist similar car renting projects. The standards applied here can be used to create software requirements specifications directly or can be used as a model for defining an organization.

**Definitions, acronyms, and abbreviations**

| Acronym | Abbreviation | Definition |
|---|---|---|
| UML | Unified Modeling Language | It is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying and documenting the artifacts of a software system. |
| CRUD | Create, Read, Update, Delete | Operations that are performed on stored data. |
| UoW | Unit of Work | Framework used to keep track of changes made to the database(creation, updates and deletes) |

**References**

The following documentation has been used as a reference in the project for the front-end:

Bootstrap version 3.3.7- https://getbootstrap.com/docs/3.3/

The UML definition has been referred from:

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

## 2. Overall description

This document contains the requirement presented to us for a course project to develop a car rental web application. It further contains a list of the stakeholders and users of the proposed solution. It also illustrates the needs and wants of the stakeholders that are mentioned in this requirements document. It further lists and briefly describes the major features and a brief description of each of the proposed system.

### Product perspective

The product depends on an external database.

The external database stores Users, Client, Vehicle data and transaction records.

### Product functions

**Clerk:**

The clerk is one of the users of this solution. The clerks are able to view and search through the catalogs using various search criteria and can also filter the results from the search. He can perform actions like managing client, creating car rental for a client, managing the reservation, handling the return of the vehicles.

**Admin:**

The administrator is the second user of this solution  who manages the vehicle record (add, modify, delete) i.e the Vehicle Catalog. He can additionally view the contents of the catalog and perform searches on the catalog using some specified characteristic of a vehicle namely;  the vehicle type,name, model, color etc. The catalog can be viewed in a sorted(ascending or descending order) form as well. The Admin also has access to view any operations done i.e. Transactions that have been performed for a vehicle along with access to view its history.

## User characteristics

Intended users of the system: the Clerk and the Admin of Star Car Rental company.

The users are expected to have the domain knowledge, basic understanding of using a computer system and scanning a website.

## Constraints

Data will be stored in an external SQL server database, bundled with web-server hosting the system.

## Assumptions and dependencies

Assumptions:

- Client can only reserve/rent cars that are available.

Dependencies:

- An external database. The user will need to configure the database.
- The application is platform dependent: It requires a web browser and an internet connection to run(for multi system use).

## 3. Specific requirements

The functional and nonfunctional requirements are listed in this section. The quality attributes are listed according to the ISO/IEC 25010 standard that classifies software quality in a structured set of characteristics and sub-characteristics.

### 3.1 External interfaces

The detailed description of all inputs into the system and all outputs from it:

The inputs to the system will be given by the Clerks and the Admin.

The output of the system will be results of the input specified by the users of the system, namely Clerk or Admin.

Below are detailed interactions and results:

**Clerk Interactions:**

| Input | Output |
|---|---|
| Manage Client<br>● View<br>● Add<br>● Delete<br>● Modify | On success:<br>● Update database with details(except in view)<br>● Display Success confirmation<br><br>On Unsuccessful:<br>● Display Error message |
| Vehicle Catalog<br>● View<br>● Detailed View | ● A list of vehicles will be presented.<br>● In detail view, status of the vehicle can also be seen |
| Vehicle Rentals<br>● View<br>● Rent<br>● Return<br>● Modify<br>● Reserve | On success:<br>● Update database with details<br>● Display Success confirmation<br><br>On Unsuccessful:<br>● Display Error message |

- **Admin Interactions:**

| Input | Output |
|---|---|
| Manage Vehicle Catalog<br>● View<br>● Add<br>● Delete<br>● Modify<br>● Sort<br>● Check status | On success:<br>● In detail view, status of the vehicle can also be seen<br>● Update database with details<br>● Display Success confirmation<br><br>On Unsuccessful:<br>● Display Error message |
| Transactions History<br>● View<br>● Sort | A list of all the transactions will be presented. Admin can search/ sort through the list. |

## 3.2 Functional requirements

Functional requirements capture the intended behaviour of the system. This section contains the *Actor Goal List* and the *Use Case view*.

### *Actor goal list*

| Actor | Goal |
|---|---|
| Clerk | 1. Manage Client Record (CRUD)<br>2. View contents of Vehicle catalog<br>3. Create Vehicle Rental for a client<br>4. Return Vehicle for a client<br>5. Create Reservation for a client<br>6. Cancel Reservation for a client |
| Admin | 1. Manage Vehicle Record (CRUD)<br>2. View detailed contents of Vehicle<br>3. View and search Transaction History |

### *Use case view*

The use case model is shown in Figure 3.

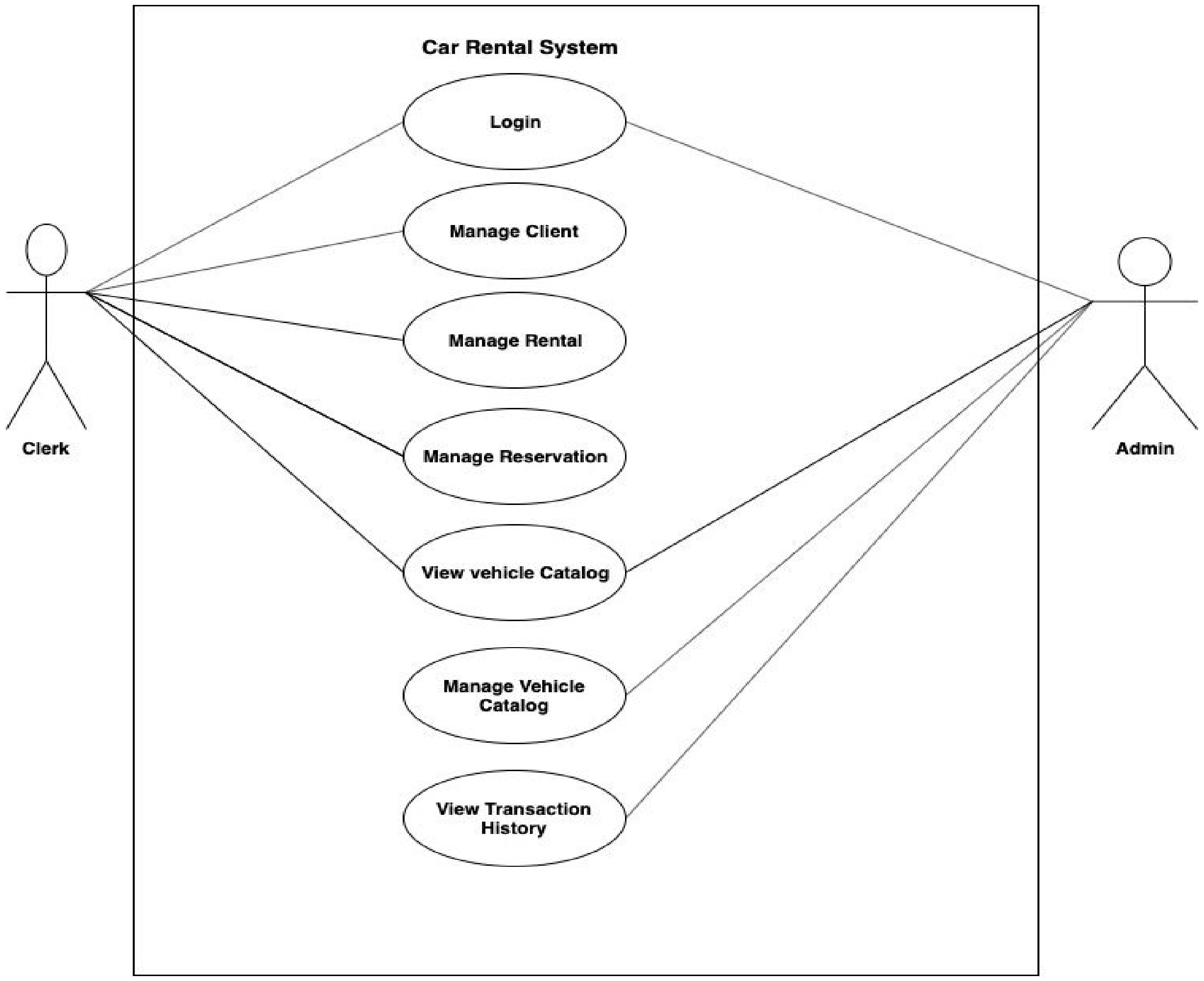


Figure 3. Use case model.

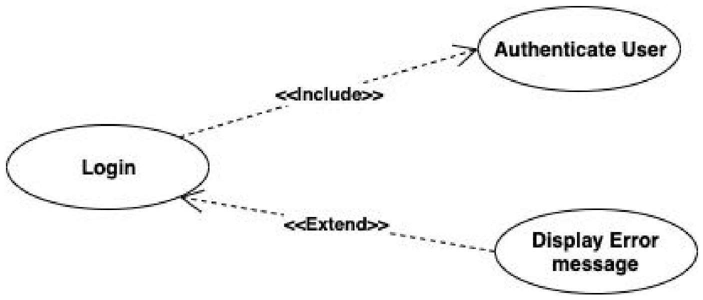


**Figure 3.1 - Login Use case**

- **UseCase 1- Login**: Each User(Admin or clerk ) is required to log in to the car rental system to perform any task. The user inputs his username and password and the system authenticates the user, if the inputted credentials are wrong, an error message is displayed and the user cannot log into the system.

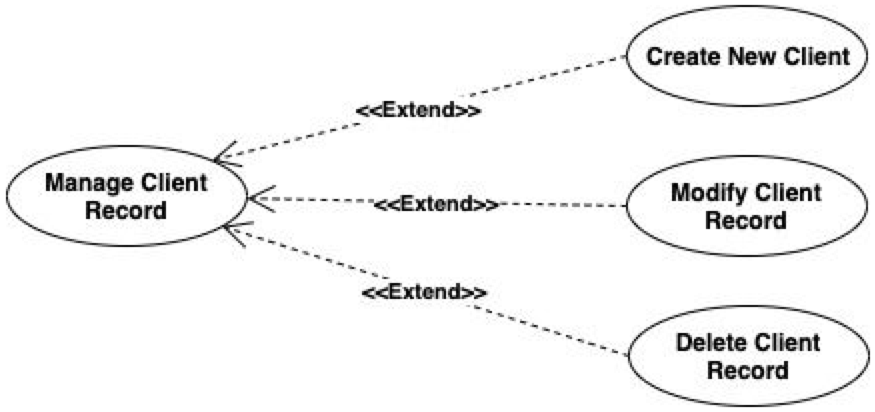


**Figure 3.2 Manage Client Record use case**

- **UseCase2- Manage Client Record:** The clerk is the actor/stakeholder here. He is required to be able to create a client record, modify/update a client record or delete a client's record, depending on the scenario. There must be a record of a client before a rental or reservation action can occur.

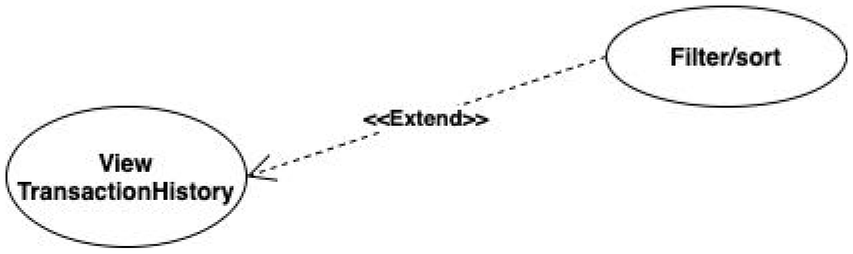


**Figure 3.3 View Transaction Use case**

- **UseCase 3: View Transaction History:** The admin is the actor/ stakeholder here. He is able to view every transaction that has been carried out in the car rental system and its corresponding timestamp. The relevant transactions are rentals or reservations. The admin is also able to sort through or filter these records at his own discretion either by due date, client, car model, etc.

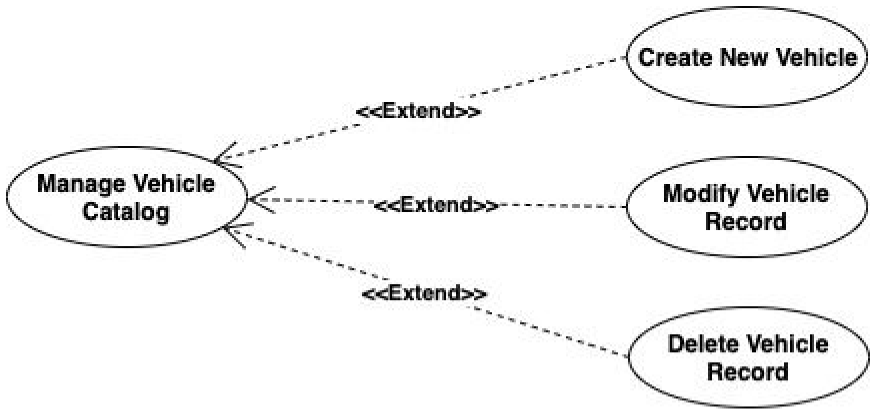


**Figure 3.4 Manage Vehicle Record**

- **UseCase 4- Manage Vehicle Records:** The admin is the actor/stakeholder here. he is responsible for managing the car rental systems vehicle catalog. He can create/add a vehicle record, modify a vehicle record or delete the vehicle record. This is an essential part of the car rental system as it if from the vehicle catalog a clerk is able to get the required information to make a rental or reservation for a client.

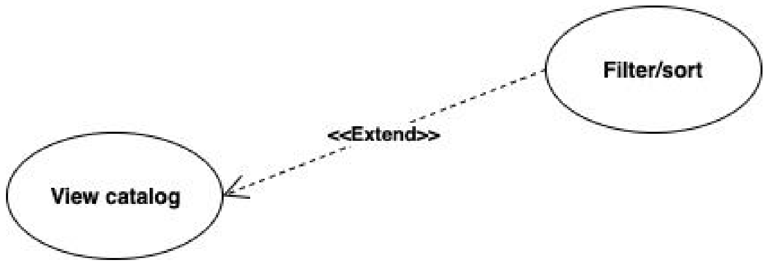


**Figure 3.5 View Vehicle Catalog Use Case**

- **UseCase 5: View Catalog:** All users/actors of the system are able to view the vehicle Catalog. This gives an overview of all vehicles in the Car rental system and the details of the vehicle: color, make, model, type, license plate,year and status(available, rented or reserved)

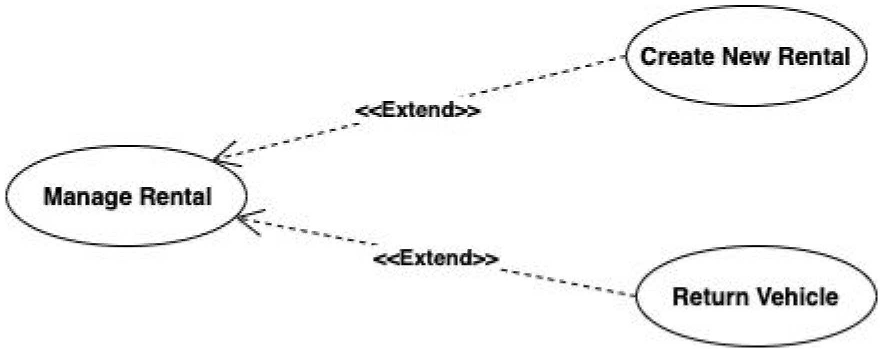


**Figure 3.6 Manage Rental Use case**

- **UseCase 6-Manage rental:** The clerk is the actor/stakeholder here. He is able to create a rental, or return a vehicle for a client. He is required to get all the clients information(name(first and last), license number, license expiry duration of rental, vehicle) and submits this information the system for a rental. When a return is to be made on the due date, the clerk update the rental information so that the vehicle is now available for other clients.

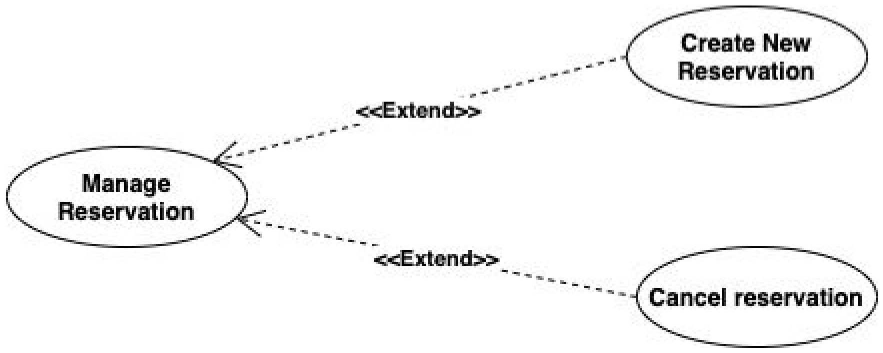


**Figure 3.7 Manage Reservation Use case**

- **UseCase 7-Manage Reservation:** The clerk is the actor/stakeholder here. He is able to create a reservation, or cancel a reservation for a client. He is required to get all the clients information(name(first and last), license number, license expiry duration of rental, vehicle) and submits this information the system for a reservation of a vehicle. If a client cancels his reservation, the clerk update the reservation  information so that the vehicle can be reserved for some other client.

### 3.3 Non-functional requirements

#### *Performance efficiency*

This refers to the responsiveness of this solution. As an interactive solution, there should be no delays to accomplish any task performed by the system users i.e. Admin or Clerk.

#### *Compatibility*

As a web application software, this solution will be compatible with every internet browser namely: Safar, Chrome, Firefox etc. The system can run on any operating system supporting a web browser namely: MAC OS or WINDOWS OS

#### *Usability*

They system will have a user friendly interface that will require little to no training for the Admin or Clerk.

#### *Reliability*

Since the requirements for this solution are limited and specific, the car rental system should work correctly after being tested. Also error handling and ability to recover after any failure will be addressed.

#### *Security*

The system does not allow unauthorised  access by just anyone. Only users that are created on the system who provide correct credentials(username and password) will access the system.

#### *Maintainability*

The system architecture used must allow for ease of modification and upgrades after project development and initial release.

### *Portability*

The project requires minimum set up and is accessible and executable on any browser irrespective of version.

### *Design constraints*

The system is built using Java with Spring Framework  and a persistent database that supports concurrency.

### *(On-line) user documentation and help*

Description.

### *Purchased components*

Not Applicable.

### *Licensing requirements*

Not Applicable.

### *Legal, copyright and other notices*

Not Applicable.

## 4. Analysis Models

The list of all analysis models used in developing specific requirements previously given in this SRS. Furthermore, each model can be traced in the SRS's requirements.

- Use case Diagram (Section 3).
- Domain Model

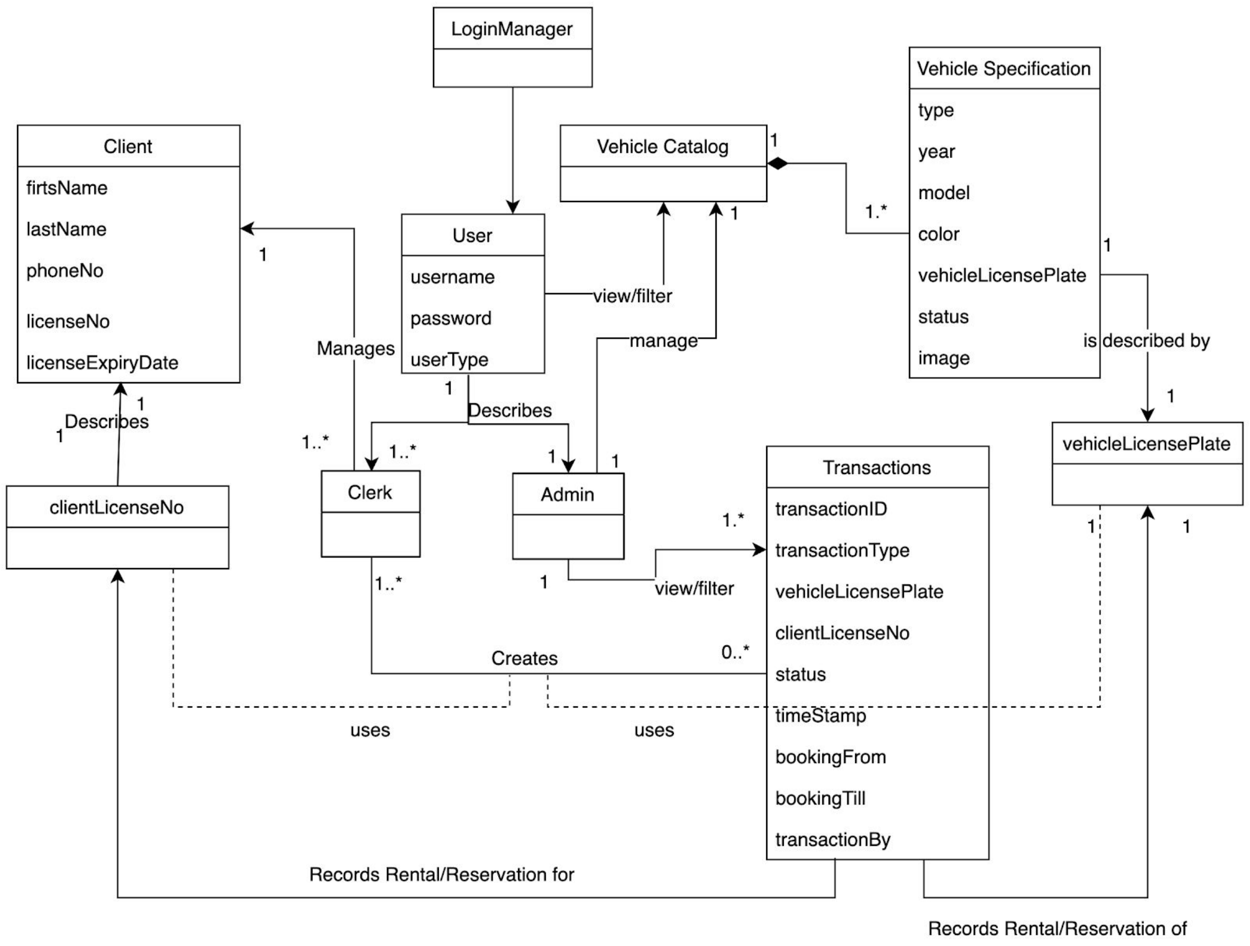


**Figure 4.1: Domain Model**

We have a large domain model. To better organise and make it easy to understand we have divided it into packages.

Below are the **package diagrams**:

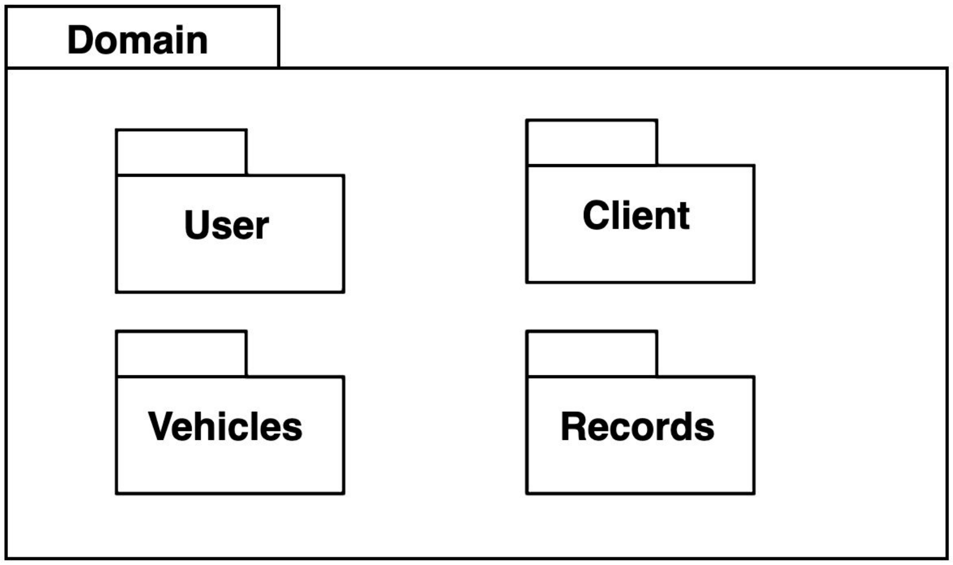


Figure 4.2 Domain Model Package diagram

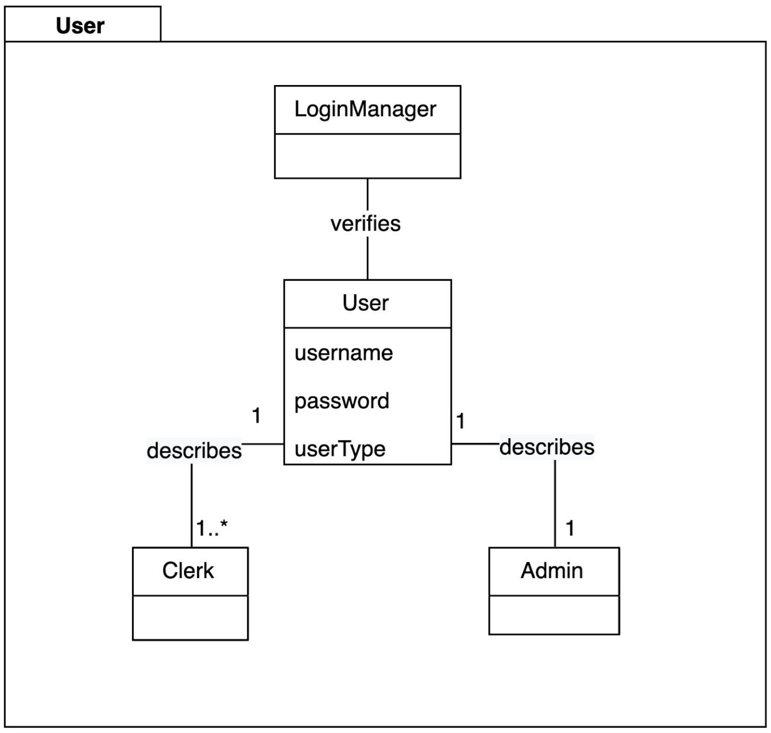


**Figure 4.3 User Package Diagram**

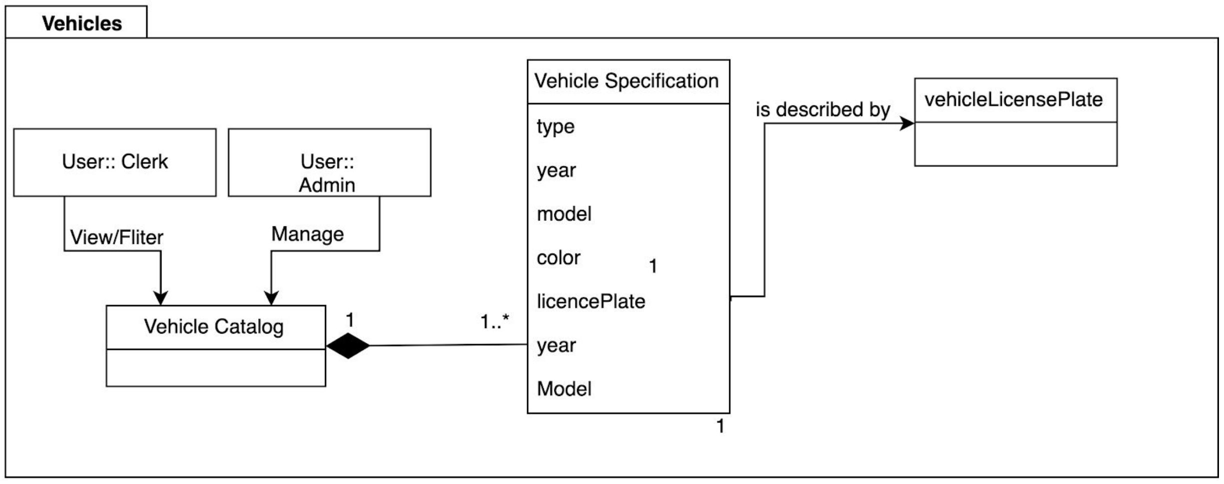


**Figure 4.4 Vehicle Package Diagram**

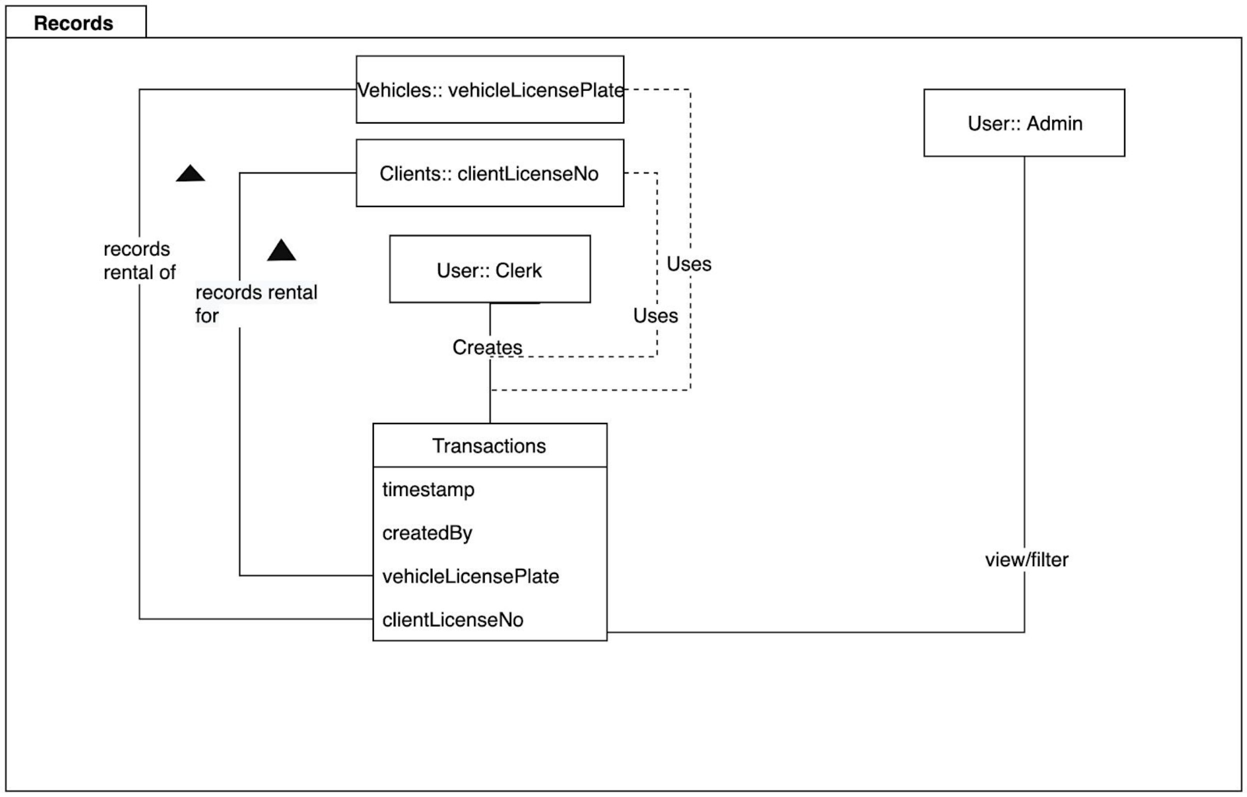


**Figure 4.5 Records Package Diagram**

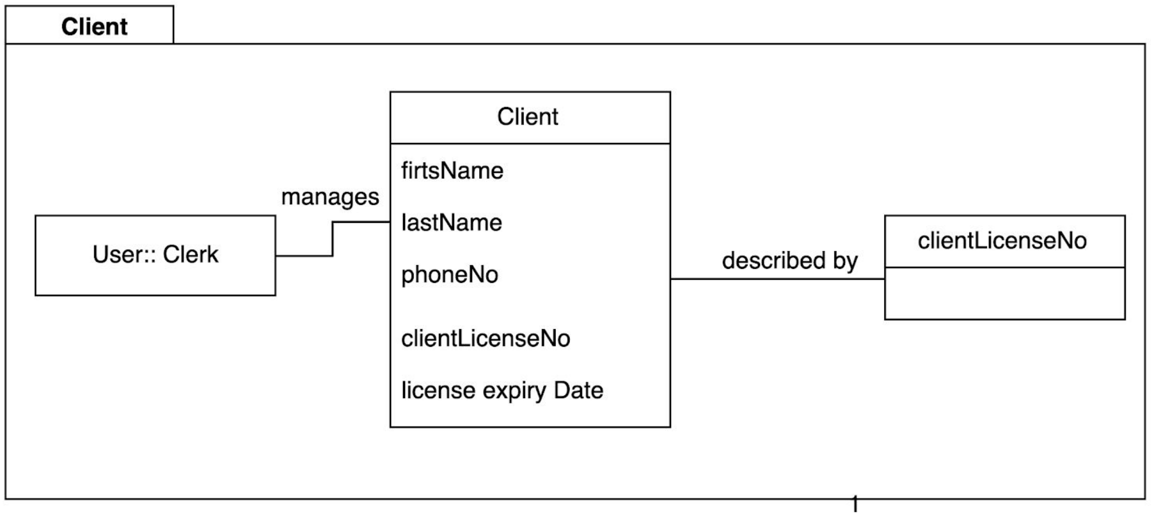
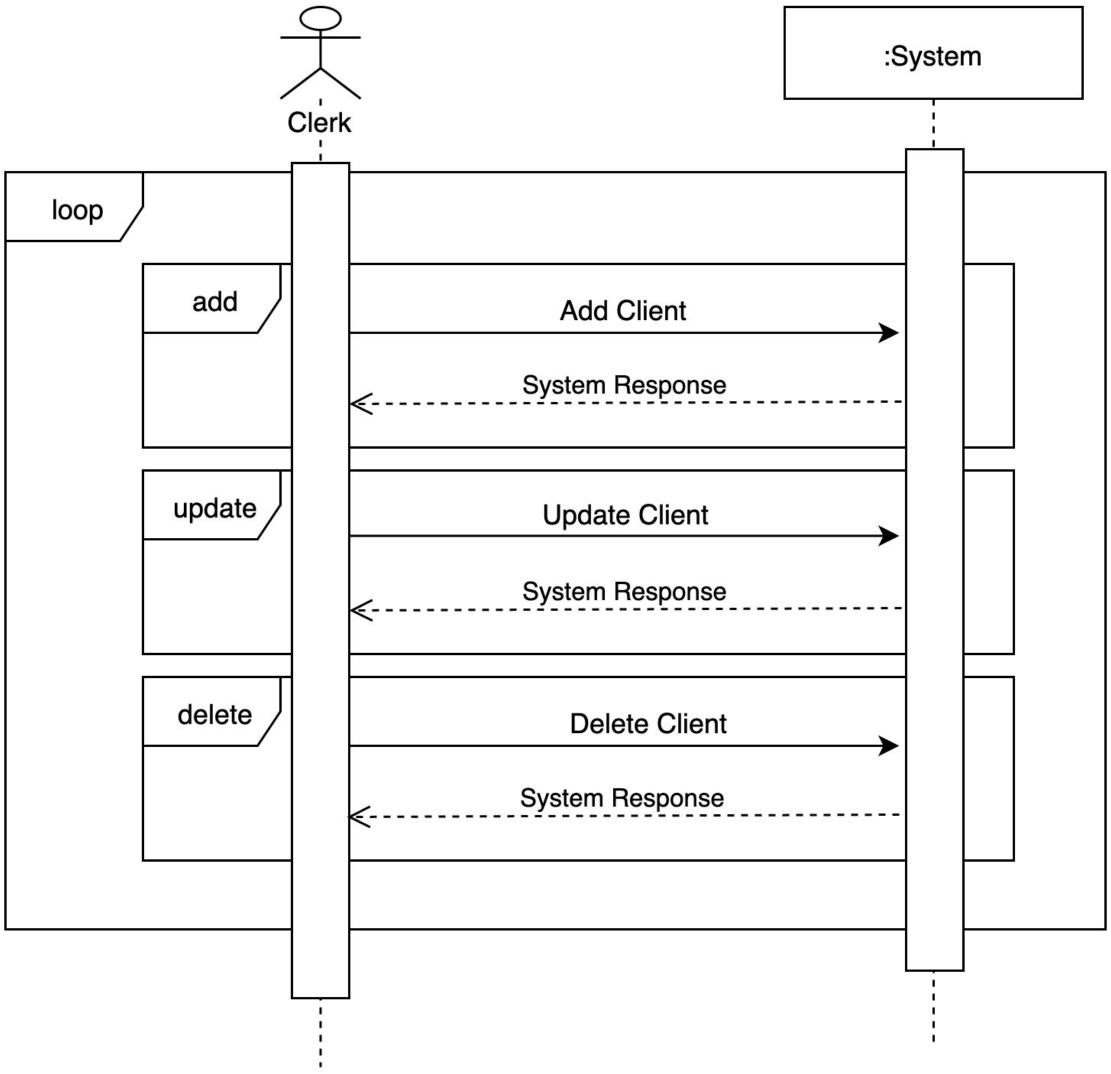


**Figure 4.6 Client Package Diagram**

**Based on domain model we have generated system operations and SSD’s:**



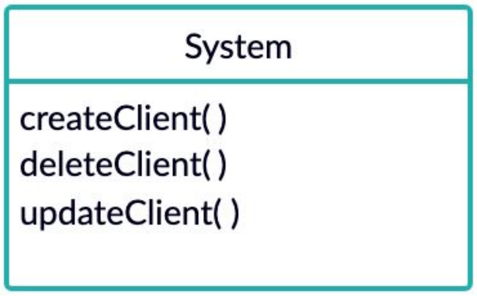


**Figure 4.2 Manage Client System Sequence Diagram and System Operation**

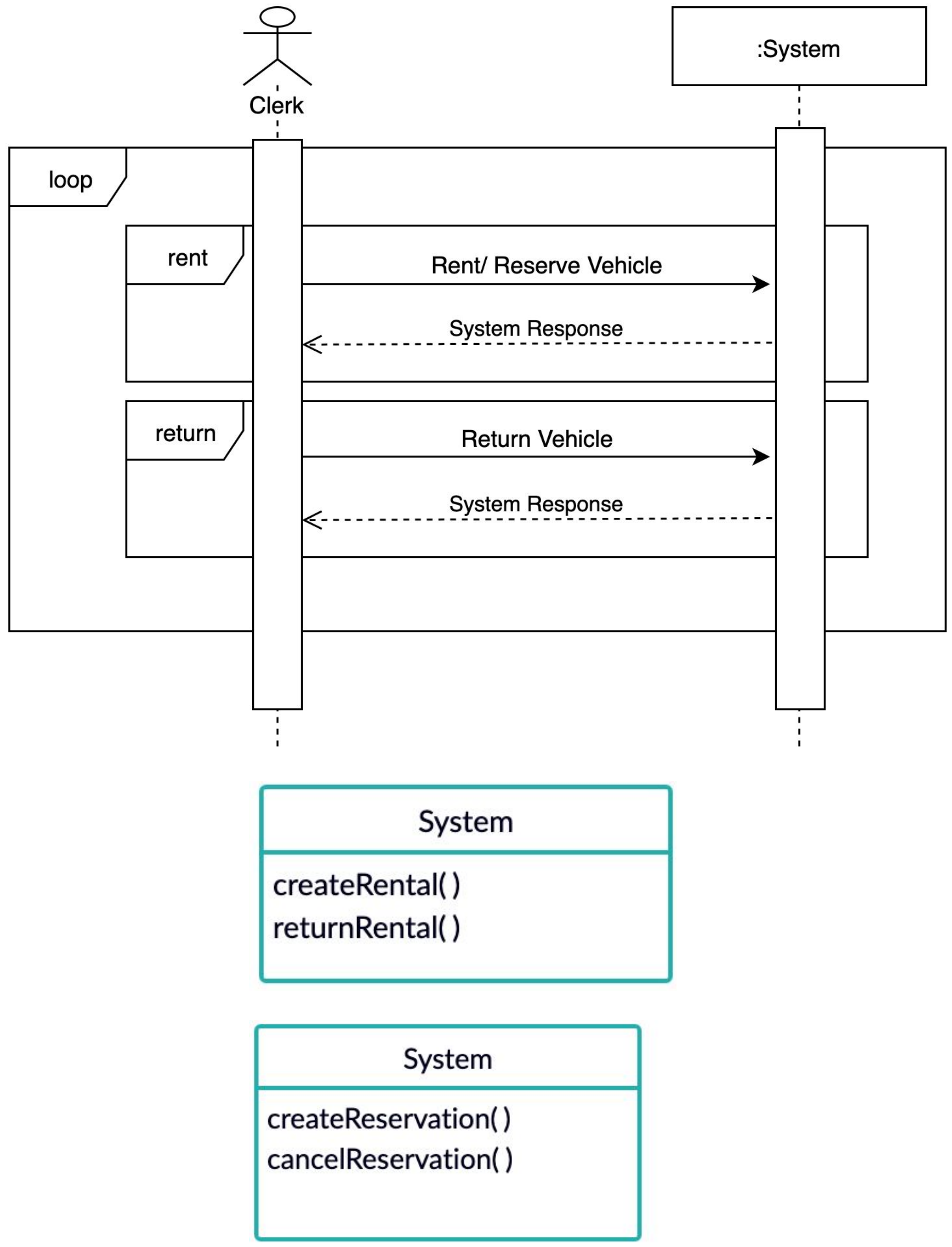
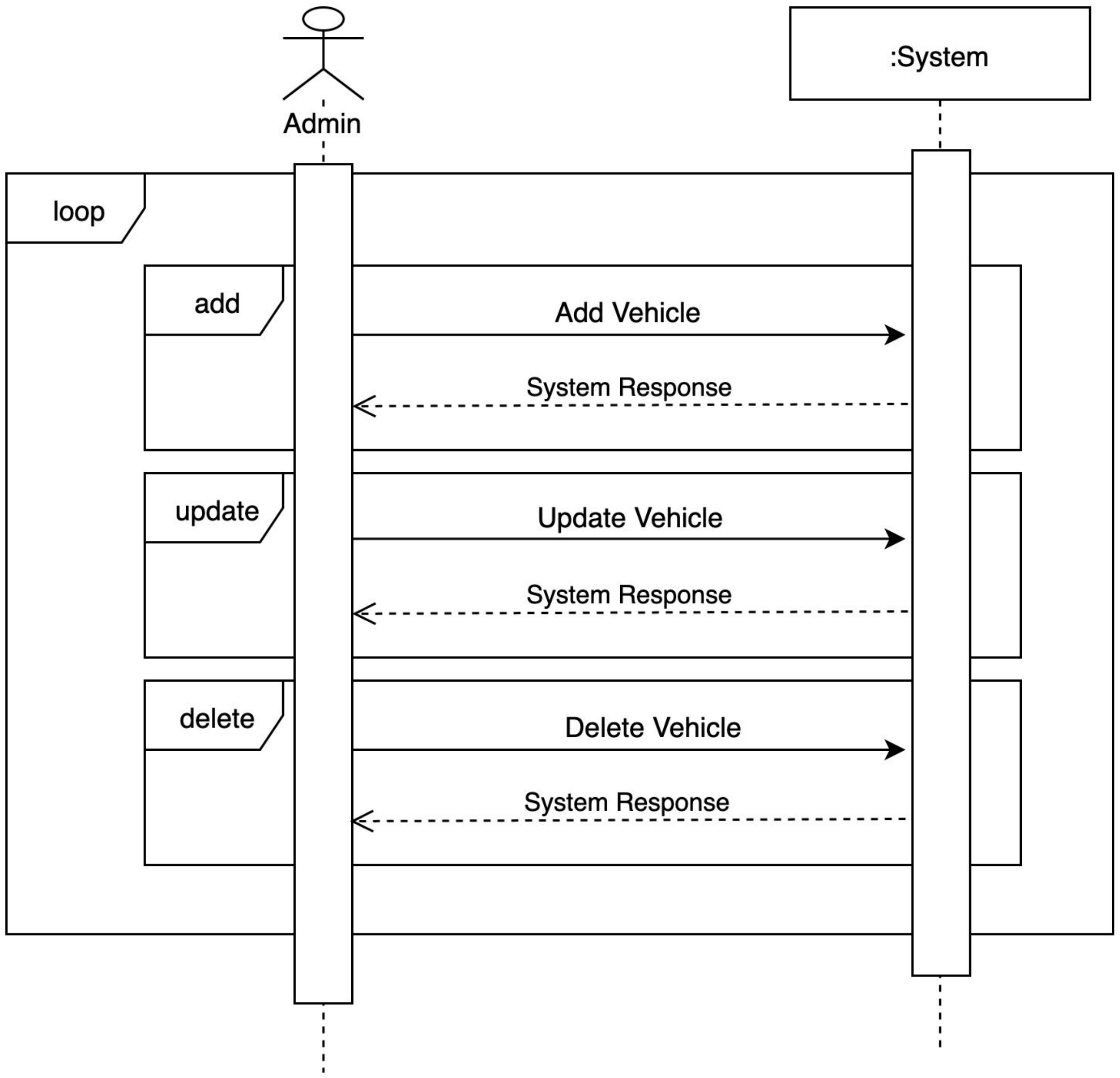


**Figure 4.3 Vehicle Rental/Reservation System Sequence Diagram & System Operation**



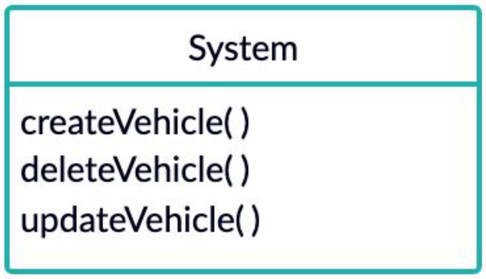
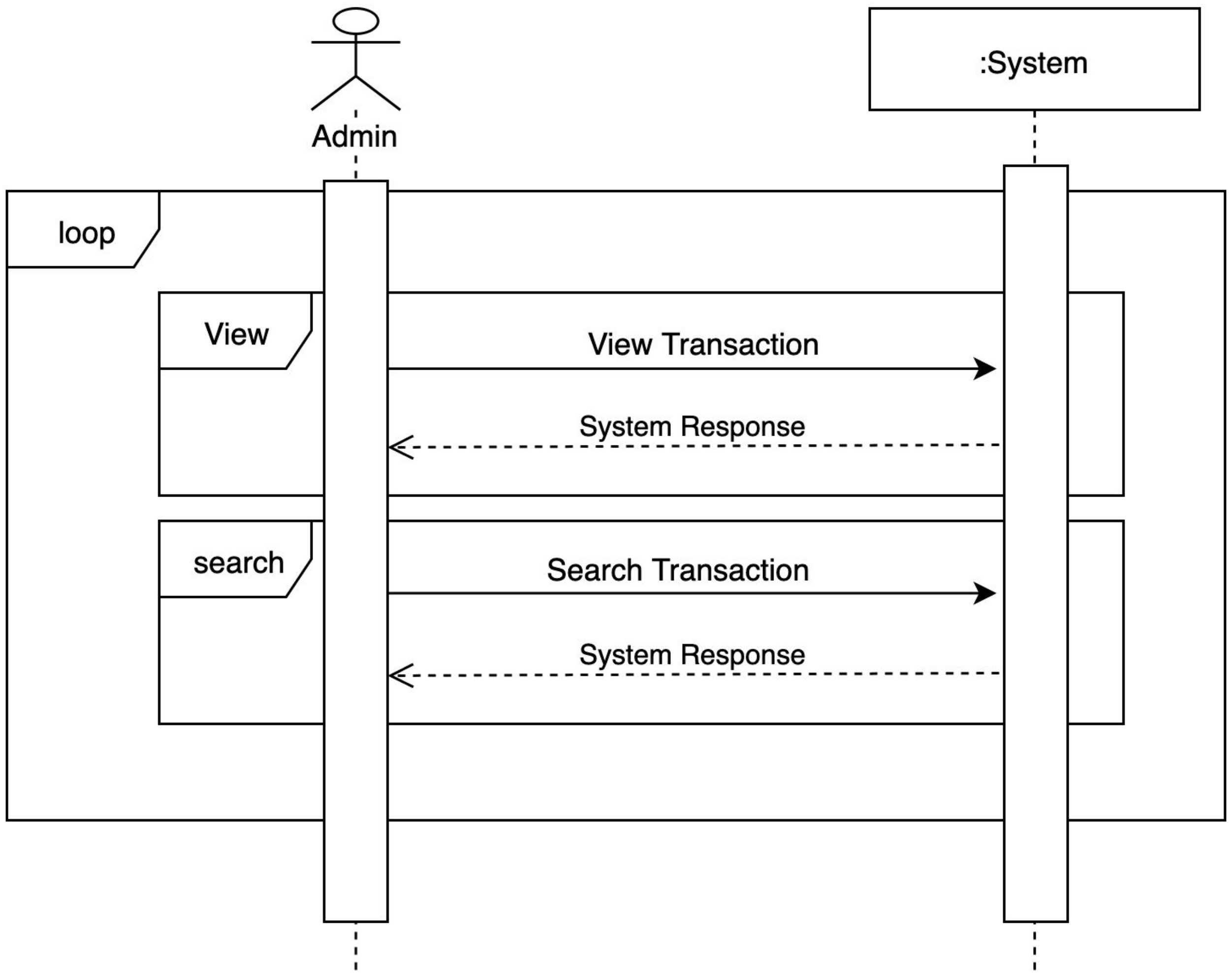


**Figure 4.5 Manage Vehicle System Sequence Diagram and System Operation**



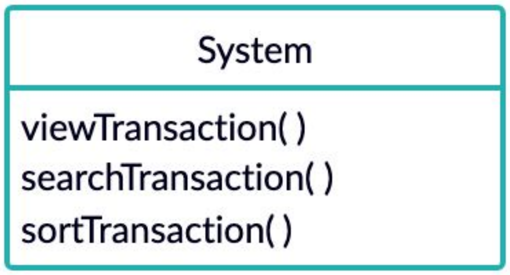


**Figure 4.6 View Transaction System Sequence Diagram and System Operation**

Based on the **System Operations** following are **Operation Contracts** for critical scenarios:

**Contract CO1:** CreateClient

**Operation:** createClient()

**CrossReferences:** **Use Case: Manage Client**

**Preconditions:**
- There is a new client being created.
- A record with same clientLicenseNumber does not exist in database.

**Postconditions:**
- A Client instance was created.
- The instance is verified and registered to clientUoW.
- A new client row successfully added to client the database.

**Contract CO2:** UpdateClient

**Operation:** updateClient()

**CrossReferences:** **Use Case: Manage Client**

**Preconditions:**
- A client with the requested record exists in the client UOW or database.
- Information modified is valid.

**Postconditions:**
- The Client instance fetched was updated successfully.
- The instance is verified and registered to client UOW.
- The respective client row successfully updated in the client database.

**Contract CO3:** DeleteClient

**Operation:** deleteClient()

**CrossReferences:** **Use Case: Manage Client**

**Preconditions:**
- A client with the requested record exists in the database.
-The client instance is not a part of clientUOW for addition or updation.

**Postconditions:**
- A Client instance fetched and added to client UOW for deletion.
- The respective client row deleted from the database.

**<u>Contract CO4:</u>** **<u>CreateRental</u>**

**Operation:** createRental()

**CrossReferences:** **Use Case: Create Rental**

**Preconditions:** - A record with clientLicenseNumber exists in database.

- A record with vehicleLicenseNumber exists in database and is available.

**Postconditions:** - A vehicle instance *newV* fetched and transaction instance *newT* created.

- *newV* is updated with Rented status.

- newT is associated with clientLicenseNumber, vehicleLicenceNumber and timeStamp.

- newV is verified and registered to vehicleUOW for update.

- The respective vehicle row updated in vehicle database.

- newR is verified and registered to transactionUOW for addition.

- A new transaction row for *newT* added to transaction database.

**<u>Contract CO5:</u>** **<u>ReturnRental</u>**

**Operation:** returnRental()

**CrossReferences:** **Use Case: Return rental**

**Preconditions:** - A Transaction with matching rental with the requested record exists in the database.

**Postconditions:** - A vehicle instance *newV* fetched and transaction instance *newT* created.

- *newV* is updated with Rented status.

- newT is associated with clientLicenseNumber, vehicleLicenceNumber and timeStamp.

- newV is verified and registered to vehicleUOW for update.

- The respective vehicle row updated in vehicle database.

- newR is verified and registered to transactionUOW for addition.

- A new transaction row for *newT* added to transaction database.

**<u>Contract CO6:</u>** **<u>CreateReservation</u>**

**Operation:** createReservation()

**CrossReferences:** **Use Case: Create Reservation**

**Preconditions:** - A record with clientLicenseNumber exists in database.

- A record with vehicleLicenseNumber exists in database and is available.

**Postconditions:** - A vehicle instance *newV* fetched and transaction instance *newT* created.

- *newV* is updated with Reserved status.

- newT is associated with clientLicenseNumber, vehicleLicenceNumber and timeStamp.

- newV is verified and registered to vehicleUOW for update.

- The respective vehicle row updated in vehicle database.

- newR is verified and registered to transactionUOW for addition.

- A new transaction row for *newT* added to transaction database.

**<u>Contract CO7:</u>** **<u>ReturnReservation</u>**

**Operation:** returnReservation()

**CrossReferences:** **Use Case: Return Reservation**

**Preconditions:** - A Transaction with matching reservation with the requested record exists in the database.

**Postconditions:** - A vehicle instance *newV* fetched and transaction instance *newT* created.

- *newV* is updated with UnReserved status.

- newT is associated with clientLicenseNumber, vehicleLicenceNumber and timeStamp.

- newV is verified and registered to vehicleUOW for update.
- The respective vehicle row updated in vehicle database.
- newR is verified and registered to transactionUOW for addition.
- A new transaction row for *newT* added to transaction database.

**<u>Contract CO8:</u>** **<u>CreateVehicle</u>**

| | |
|---|---|
| **Operation:** | createVehicle() |
| **CrossReferences:** | **Use Case: Manage Vehicle** |
| **Preconditions:** | - There is a new vehicle being created. |
| | - A record with same vehicleLicenseNumber does not exist in database. |
| **Postconditions:** | - A Vehicle instance v is created. |
| | - The new instance v, is registered with the vehicle UOW for creation. |
| | - The newly created vehicle v is inserted into the database. |

**<u>Contract CO9:</u>** **<u>UpdateVehicle</u>**

| | |
|---|---|
| **Operation:** | createVehicle() |
| **CrossReferences:** | **Use Case: Manage Vehicle** |
| **Preconditions:** | - The requested vehicle record exists in the database or vehicle UOW. |
| | - Information modified is valid. |
| **Postconditions:** | - The Vehicle instance v, to be modified is updated successfully. |
| | - The instance v, is registered with the vehicle UOWork for an update. |
| | - The instance v in respective vehicle row in the database is updated. |

**<u>Contract CO10:</u>** **<u>DeleteVehicle</u>**

| | |
|---|---|
| **Operation:** | deleteVehicle() |

**CrossReferences:** **Use Case: Manage Vehicle**

**Preconditions:** - A Vehicle with the requested record exists in the database.

**Postconditions:** - A Vehicle instance v to be deleted is  fetched.

-  The instance v is registered with the vehicle UOW for a delete.

- The instance v is  successfully removed from the database.

## Contract CO11: ViewTransactions

**Operation:** viewTransactions()

**CrossReferences:** **Use Case: View Transactions History**

**Preconditions:** -  Transaction records are available in the database.

**Postconditions:** - A list for transactions instances created.

- List is populated with all transaction rows from database.

## Contract CO12: searchTransactions

**Operation:** searchTransactions()

**CrossReferences:** **Use Case: View Transactions History**

**Preconditions:** -  Transaction records are available in the database.

**Postconditions:** - A list for transactions instances *trans* fetched from viewTransactions is used.

- A new list *searched* is generated.

- Transaction instances in *trans* instance that match the search criteria are added to *searched* list instance.

## Contract CO13: sortTransactions

**Operation:** sortTransactions()

**CrossReferences:** **Use Case: View Transactions History**

**Preconditions:** - Transaction records are available in the database.

**Postconditions:** - List of transactions instances *trans* fetched from viewTransactions used.

- A new list *sorted* is generated.

- Transaction instances in *trans* instance are sorted and are added to *sorted* list instance.

You may also use ***UML state diagrams*** to describe **critical use cases**, one state diagram per use case.

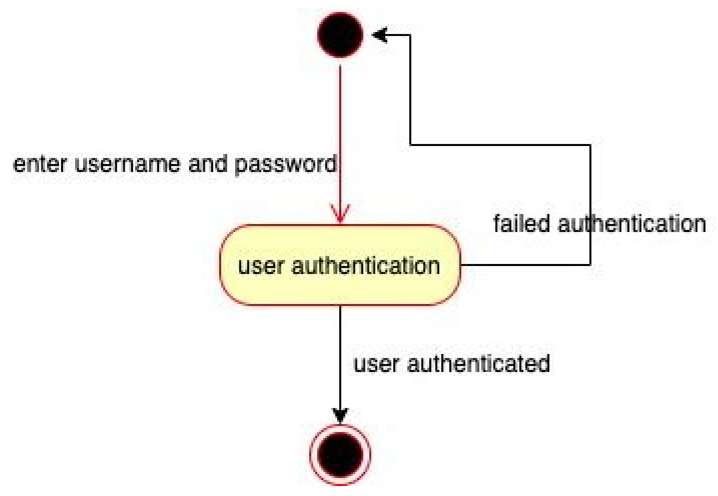


**Figure 4.6 Login state diagram**

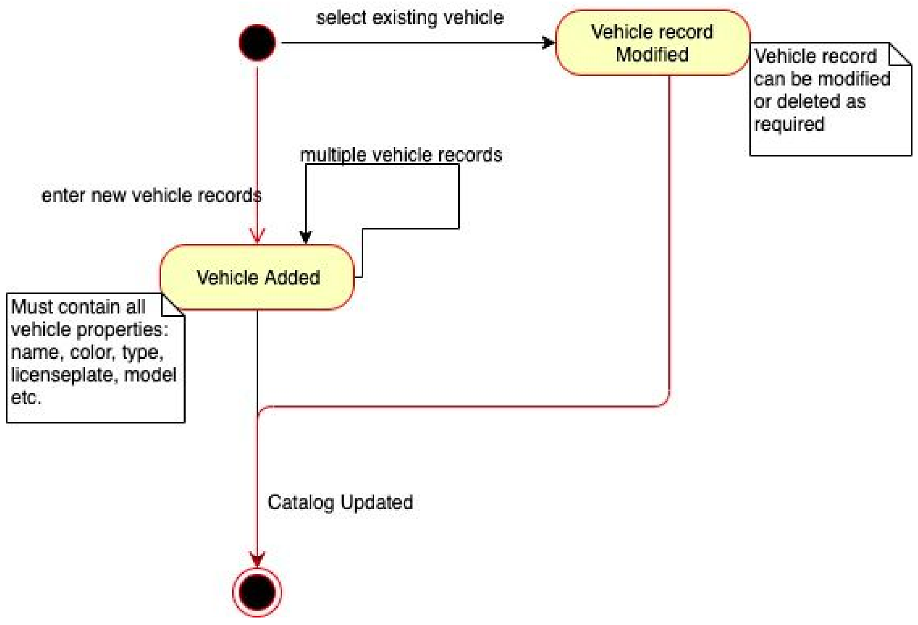

select existing vehicle
Vehicle record Modified
Vehicle record can be modified or deleted as required
multiple vehicle records
enter new vehicle records
Vehicle Added
Must contain all vehicle properties: name, color, type, licenseplate, model etc.
Catalog Updated

**Figure 4.7 Manage Vehicle Records State Diagram**

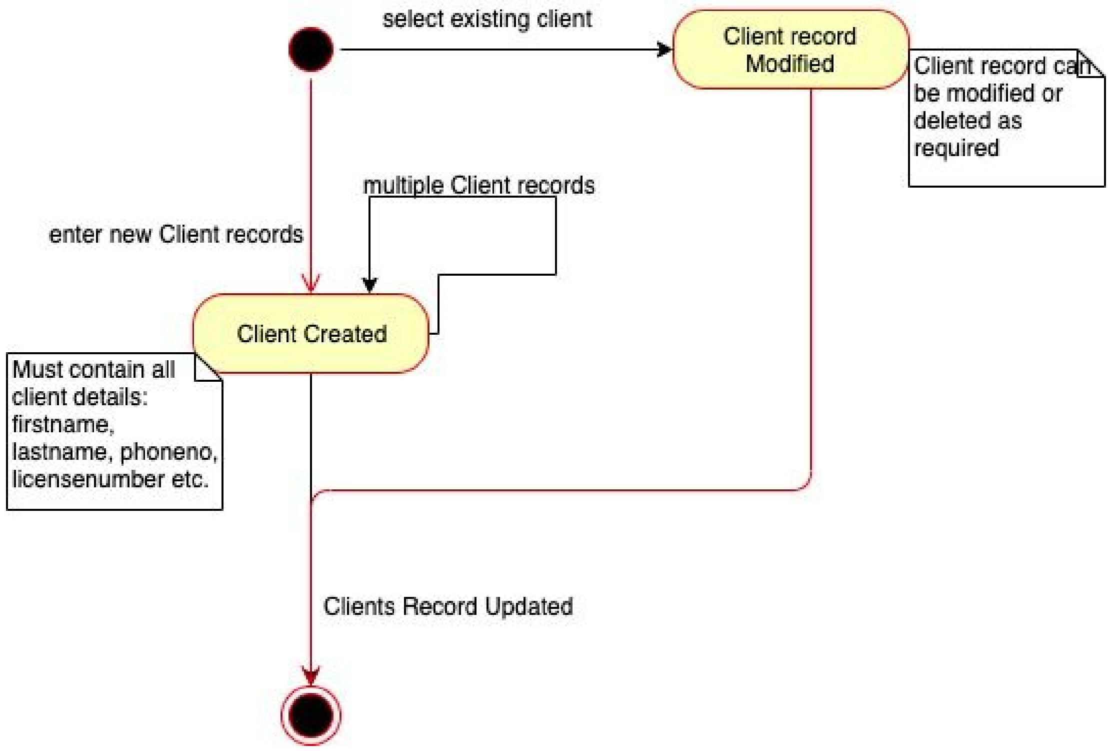

select existing client
Client record Modified
Client record can be modified or deleted as required
multiple Client records
enter new Client records
Client Created
Must contain all client details: firstname, lastname, phoneno, licensenumber etc.
Clients Record Updated

**Figure 4.8 Manage Client Record State Diagram**

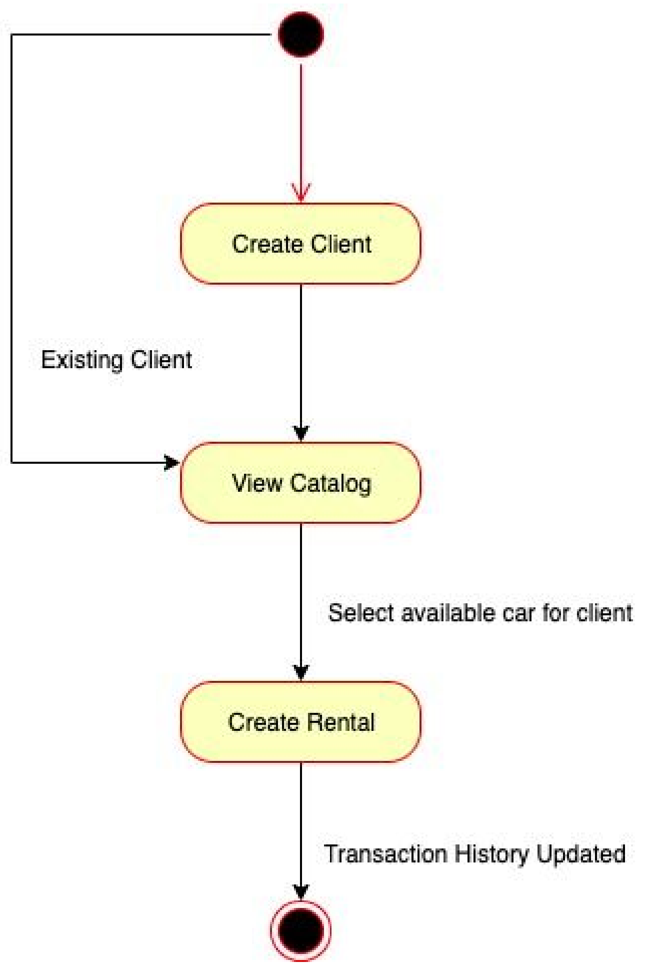


**Figure 4.9 Rent Vehicle State Diagram**