Software Architecture Document

Version 1.0

for

SOEN-6461-Team 11

Prepared by

Anusha Keralapura Thandavamurthy	40102962	kt.anusha21@gmail.com
Arvind Korchibettu Adiga	40105178	adiga1993@gmail.com
Basant Gera	40082433	basantgera29@gmail.com
Koteswara Rao Panchumarthy	40084998	kotichowdary18@gmail.com
Sai Charan Duduka	40103928	charan140494@gmail.com
Sourabh Rajeev Badagandi	40098471	sourabh.rajeev@gmail.com

Instructor: Dr. C. Constantinides

Course: SOEN-6461

Date: 03-12-2019

Document history

Date	Version	Description	Author
28-September-2019	1.0	Writing the software Architecture used throughout project	Basant Gera
21- october -2019	1.1	Updating Changes / Scenarios in iteration 2/Adding scenarios for Iteration 3	Basant Gera
15-November-2019	1.2	Adding and Finalizing Iteration 4 with combined updating of SAD documents with persistence and concurrency issues	Basant Gera
17-November-2019	1.3	Updated Architecture Diagram, added process view and data model diagrams. Prepared final draft.	Sourabh Rajeev Badagandi

Table of contents

List of figures

Definitions, acronyms, and abbreviations

VRMS	Vehicle Renting management system

1. Introduction

The document will provide you an overview of the entire S***oftware Architecture*** for VRMS (vehicle Renting management system) which help in renting vehicle according to user needs.

Purpose

The document provides you the architectural overview of VRMS (vehicle renting management system).The sole purpose of this management system is Rent vehicle according to user needs so that he can travel on the date he booked his/her reservation or booked his rental of vehicle and return back the same on the date he /she specified.

Since in this project we are trying to achieve 2 things :

- ***Rental booking for clients*** : Vehicle which is Booked then and there.
- ***Reservation Booking for clients*** : Vehicle which is Booked for future date.

The document also capture and convey the significant architectural design which have been made in developing and designing the system.The documents tries to convey a system architect should involve in this project for better understanding of the problem which is represented in the system.

Scope

The scope of the document is to highlight the architecture of the VRMS which meets the desired requirements. Adding more to it basically focuses on how to Sale and distribute your vehicles to client via help of clerk and managing the work efficiently and effectively.

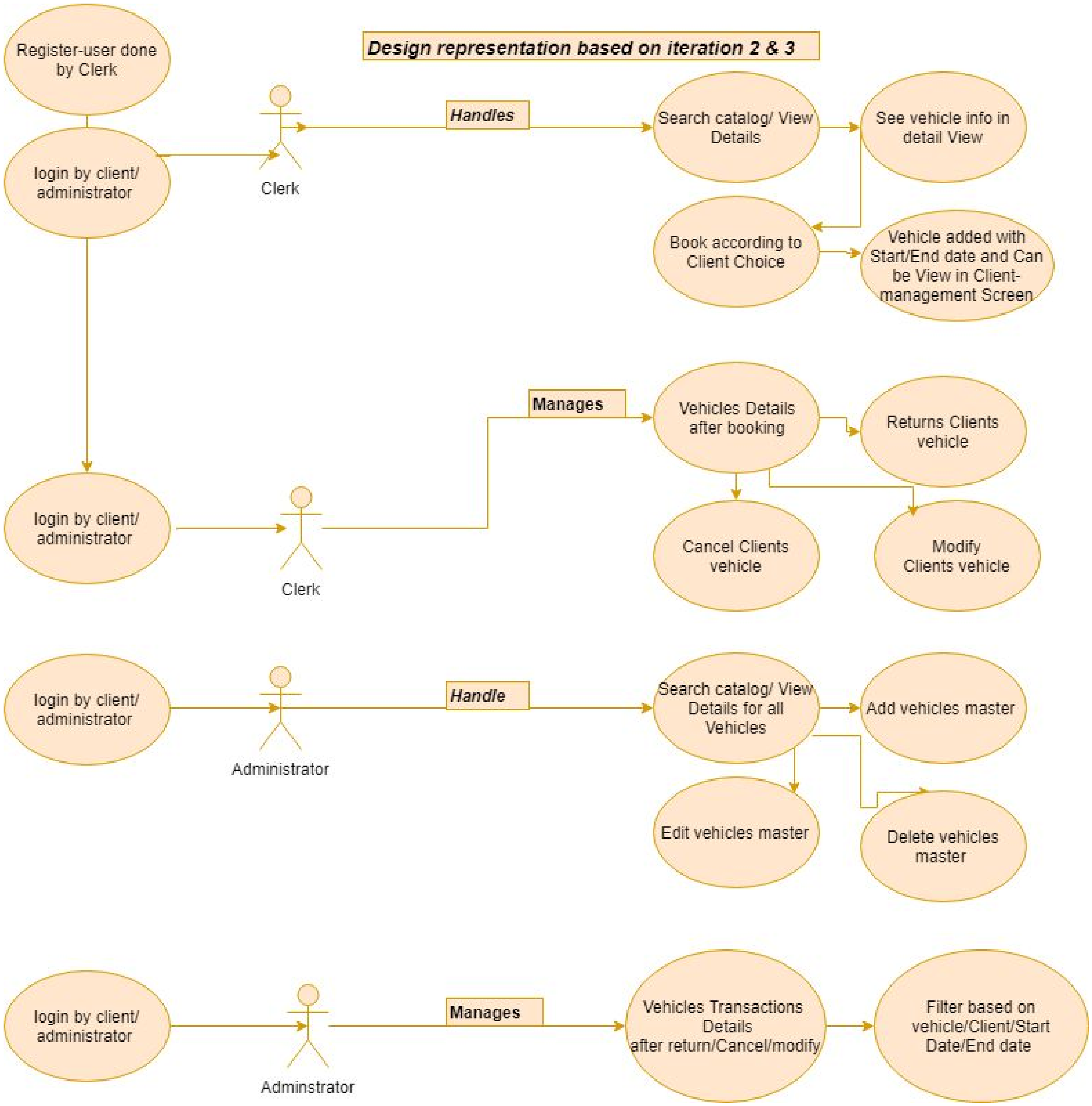
The sad document is entitled to tell you the overall architecture for the VRMS project (Rent It)and how we are able to achieve the functionalities via the following architecture which given in figure No. 2.

2. Architectural representation

Architectural representation can be explained by carrying following objective in mind which are as follows :

1. **Registration** of user on behalf of clerk/ Administrator [Register as administrator].
2. **Login** of clerk to book a vehicle according to client's availability.
3. **Checking availability of** vehicle based on various parmenter listed on Vehicle catalog page.
4. Moving to **detailed view** vehicle info and next button functionality.
5. Booking a vehicle for a client with his/her available dates in the **booking form**.
6. Checking whether vehicle is available and if yes then what kind of reservation client is looking for Such as **Reservation for future** date for vehicle or looking for **Rental reservation** in which client is looking for vehicle ASAP.
7. **Manage user request** to see which vehicle are available and which are not.
8. Clerk can edit or modify the records for the clients but not for vehicle.Only administrator has rights to add/ edit/ delete the vehicle records.
9. Clerk can add/edit/delete/ Cancel the record for booking which is done by clerk for client.
10. Administrator can add/edit/delete a vehicle from **vehicle master form**.
11. Administrator can view all the transactions/Updated record done by clerks on **transaction screen** like Modify/Cancel/Returned by Client at the same time updated record will be shown in the transactions.The records can also be filtered out based on following criteria such as: Client First Name , Client Last Name , Vehicle Model , Vehicle Make , Due Date and Start Date.

Figure 1 illustrated below shows the overall functionality / Design representation as per iteration 2.



Design representation based on iteration 2 & 3
Register-user done by Clerk
login by client/ administrator
Clerk
Handles
Search catalog/ View Details
See vehicle info in detail View
Book according to Client Choice
Vehicle added with Start/End date and Can be View in Client-management Screen
login by client/ administrator
Clerk
Manages
Vehicles Details after booking
Returns Clients vehicle
Cancel Clients vehicle
Modify Clients vehicle
login by client/ administrator
Administrator
Handle
Search catalog/ View Details for all Vehicles
Add vehicles master
Edit vehicles master
Delete vehicles master
login by client/ administrator
Adminstrator
Manages
Vehicles Transactions Details after return/Cancel/modify
Filter based on vehicle/Client/Start Date/End date

Figure 1

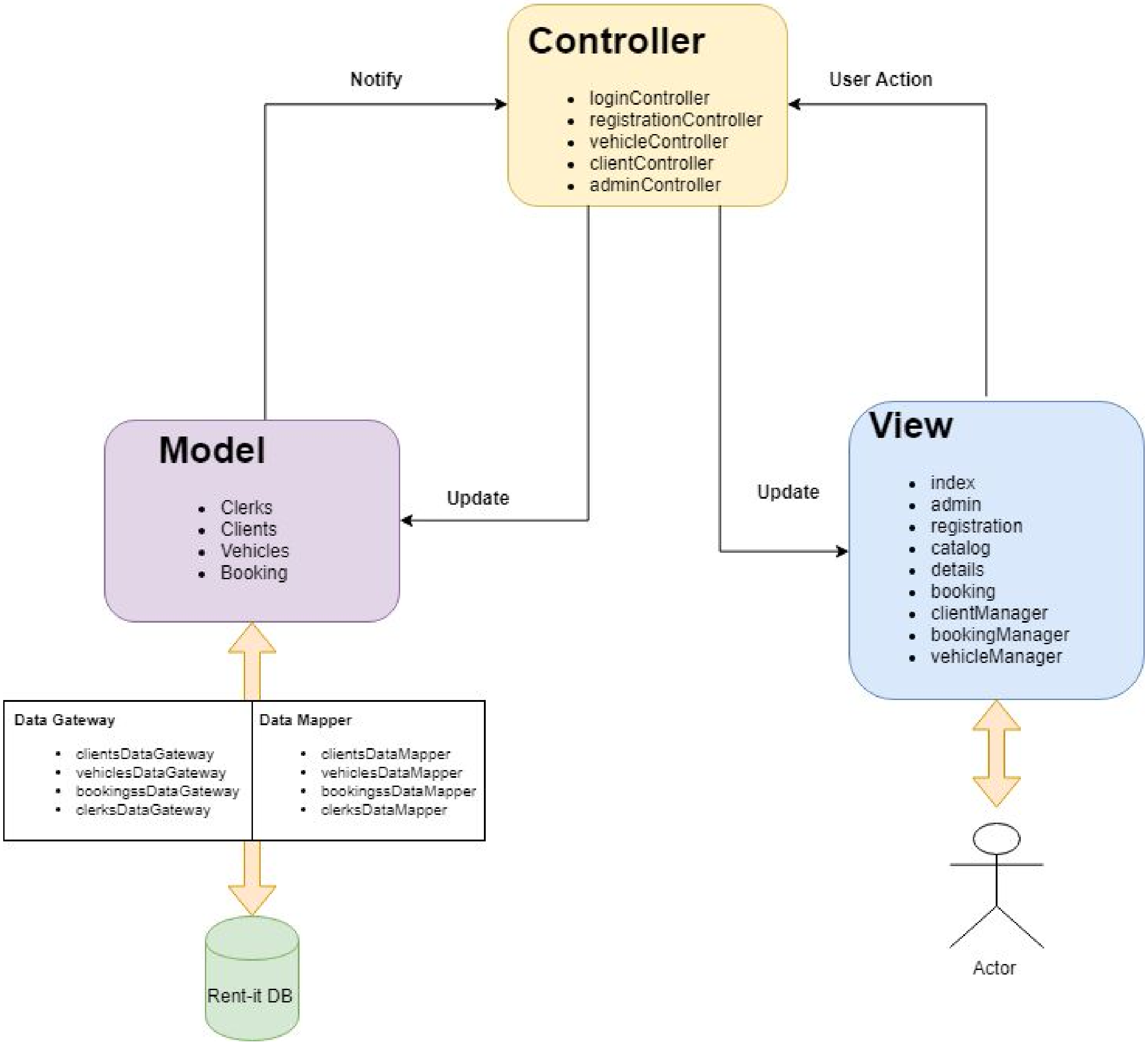


Figure 2: Overall Architecture Diagram

The application follows a MVC architecture which includes the following:

1. **Model :** Responsible to hold data of clerks/admin, vehicles, clients and booking information. The model interacts with the MySQL DB using an ORM framework.
2. **View:** Responsible for providing a rich and user friendly interface for users to interact and retrieve information.Consists of different views for viewing/editing/modifying system data.

3. **Controller:** Responsible for handling user requests and notifying any changes in the model to update different views.
4. **ORM:** Object Relational Mapping framework that has been implemented from scratch to map model elements with MySQL Database.

ORM Architecture:

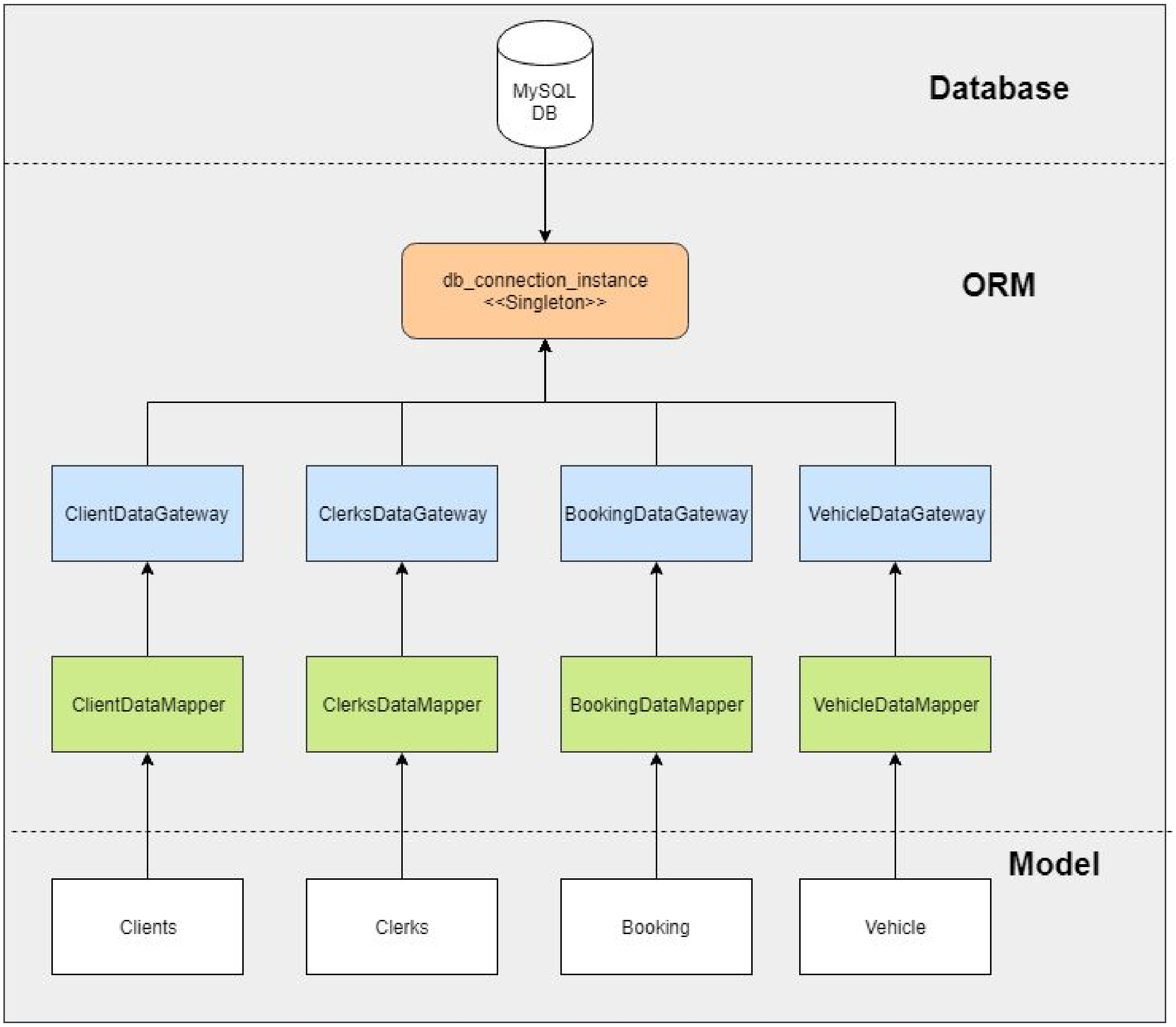


Figure:2.1 ORM Architecture

The above figure highlights the three later ORM framework:

1. **DB Connection:** It is a singleton instance which handles connection to MySQL DB.

2. **Data Gateway:** Provides interfaces to update and retrieve information from the MySQL DB via the connection instance.
3. **Data Mapper:** Provides Model Object and Relational Table mapping for the system.

Concurrency:

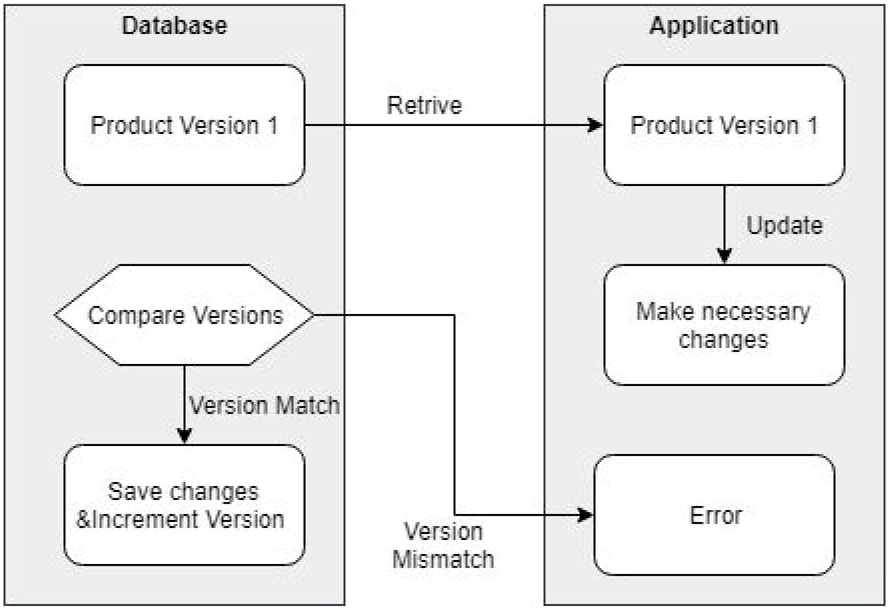


Figure:2.2

The above figure highlights the basic working of the Optimistic Concurrency Control approach used to handle concurrent user requests. Version column in the database is used to check if the current request is allowed to make changes or not. This helps avoid request anomalies such as:

1. **Dirty Reads:** reading data that is being changed by some other user.
2. **Non-repeatable reads:** reads which may result in different data if repeated in a short interval of time.
3. **Phantom Reads:** occur when other transaction sets data pertaining to current read, affecting the original data.

Architecture Patterns Used:

Pattern	Rationale
MVC(Model View Controller)	Thes system is based on MVC architecture to

	ensure that the model data is separated from the view. Also it supports different views for the same model data. Business logic is not mixed with the views, thus ensures a modular design which is easy to maintain and refactor.
Singleton	The ORM framework is built from scratch and uses a singleton class for database connection. This ensures no multiple connections to the database can be initiated in the same user session. This helps to avoid data corruption and dirty reads from the database.
OCC(Optimistic Concurrency Control) Pattern	Optimistic locking mechanism has been used to handle concurrent requests. This approach was found to the desired choice as it ensures good reliability and performance for web based applications.

System Functionalities:

1. **Clerk/Admin Registration :** Interface to register new users.
2. **Clerk/Admin Login:** Interface to validate users.
3. **Vehicles Catalog view** : interface to view vehicles added into the system, sort and search vehicles.
4. **Vehicle detailed view** : Interface to view vehicle details, view next vehicle, go back to catalog, Rent or reserve the vehicle if available.
5. **Booking a vehicle** for the client which consists of Rental /Reservation on click of book now.
6. **Managing client records** that include: handling return, cancellation and modification of order Deleting the client records if they have returned it.
7. **Administrator Add/Delete/Update/Search Catalog screen** for vehicles records
8. **Viewing the current / updated transactions** for a particular clerk and filter/ search various parameters.

The following section provides different views of the architecture and are related according to the following figure:

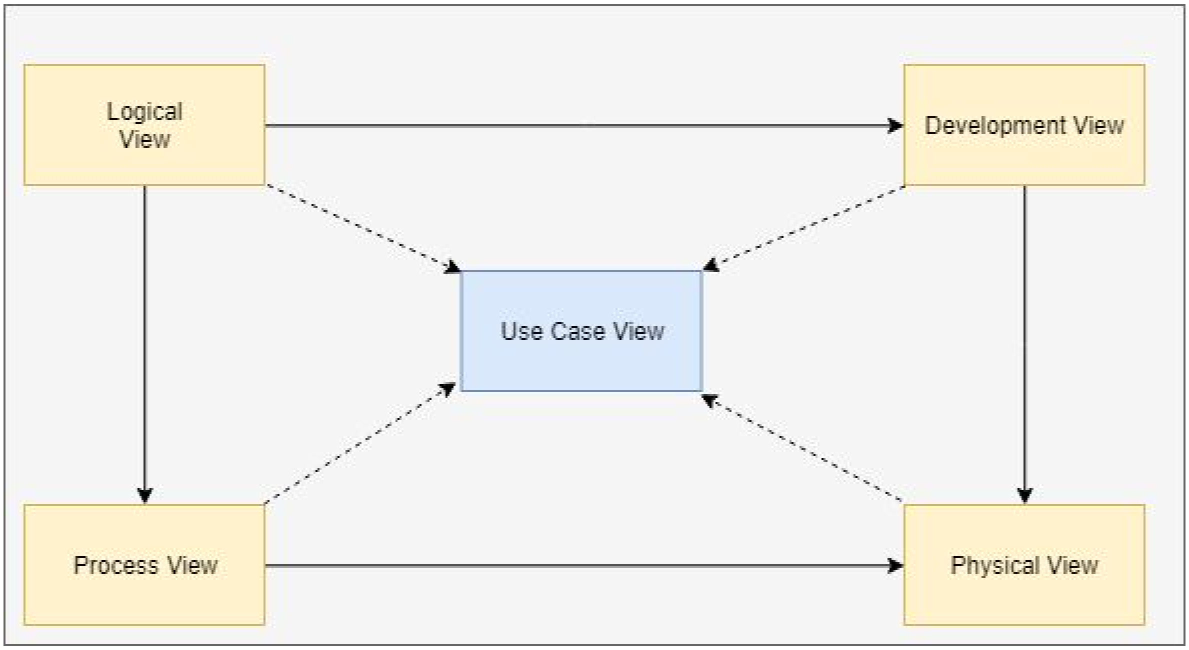


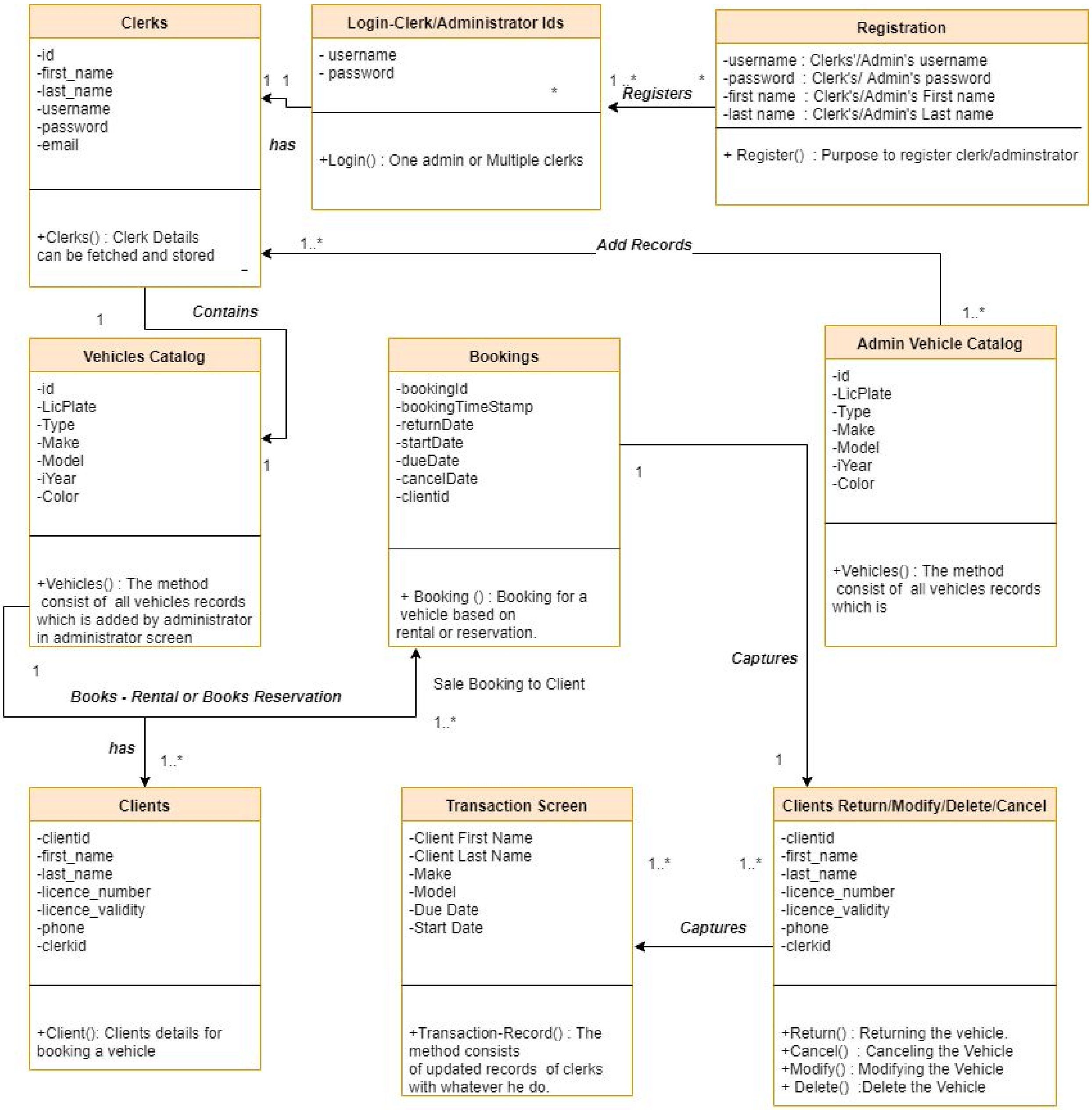
Figure:2.3 Architecture View

Logical view : Designers

Audience : Designer's

Area Concerned : How from a designer's perspective he/ she sees the overall functionality of the project.

Class Diagram : Logical View from Designer perspective.



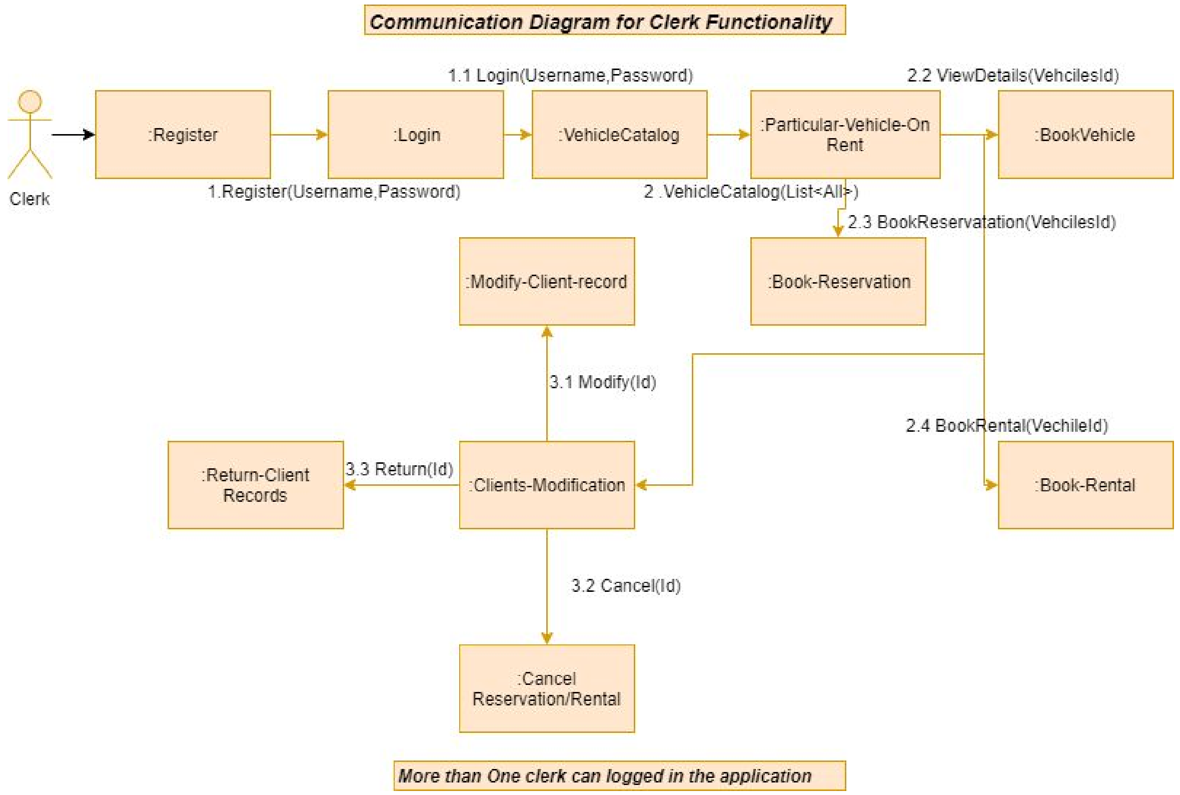
Clerks
-id
-first_name
-last_name
-username
-password
-email
+Clerks() : Clerk Details can be fetched and stored
Login-Clerk/Administrator Ids
- username
- password
+Login() : One admin or Multiple clerks
Registration
-username : Clerks'/Admin's username
-password : Clerk's/ Admin's password
-first name : Clerk's/Admin's First name
-last name : Clerk's/Admin's Last name
+ Register() : Purpose to register clerk/adminstrator
1 1
has
1..* Registers *
*
1..*
Add Records
1..*
1
Contains
Vehicles Catalog
-id
-LicPlate
-Type
-Make
-Model
-iYear
-Color
+Vehicles() : The method consist of all vehicles records which is added by administrator in administrator screen
1
Bookings
-bookingId
-bookingTimeStamp
-returnDate
-startDate
-dueDate
-cancelDate
-clientid
+ Booking () : Booking for a vehicle based on rental or reservation.
Admin Vehicle Catalog
-id
-LicPlate
-Type
-Make
-Model
-iYear
-Color
+Vehicles() : The method consist of all vehicles records which is
1
Captures
1
Books - Rental or Books Reservation
Sale Booking to Client
1..*
has
1..*
1
Clients
-clientid
-first_name
-last_name
-licence_number
-licence_validity
-phone
-clerkid
+Client(): Clients details for booking a vehicle
Transaction Screen
-Client First Name
-Client Last Name
-Make
-Model
-Due Date
-Start Date
+Transaction-Record() : The method consists of updated records of clerks with whatever he do.
1..*
1..*
Captures
Clients Return/Modify/Delete/Cancel
-clientid
-first_name
-last_name
-licence_number
-licence_validity
-phone
-clerkid
+Return() : Returning the vehicle.
+Cancel() : Canceling the Vehicle
+Modify() : Modifying the Vehicle
+ Delete() :Delete the Vehicle

Figure:2.4 Class Diagram

Interaction Diagram (Communication Diagram or Sequence Diagram):

The parts for Communication diagram is divided into 2 phases:

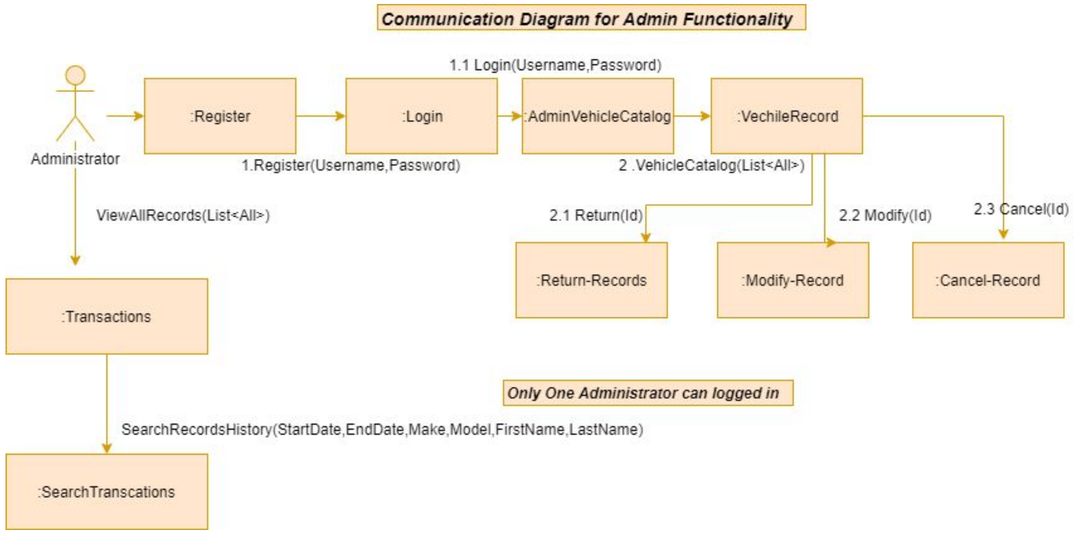
Clerk functionality : Book a vehicle for client based on Reservation or rental.



Communication Diagram for Clerk Functionality
Clerk
:Register
:Login
:VehicleCatalog
:Particular-Vehicle-On Rent
:BookVehicle
1.1 Login(Username,Password)
2.2 ViewDetails(VehcilesId)
1.Register(Username,Password)
2 .VehicleCatalog(List<All>)
2.3 BookReservatation(VehcilesId)
:Modify-Client-record
:Book-Reservation
3.1 Modify(Id)
2.4 BookRental(VechileId)
:Return-Client Records
3.3 Return(Id)
:Clients-Modification
:Book-Rental
3.2 Cancel(Id)
:Cancel Reservation/Rental
More than One clerk can logged in the application

Figure:2.5

Administrator Functionality : Adding vehicle from backend so that clerk can enroll it for client.



Communication Diagram for Admin Functionality
1.1 Login(Username,Password)
Administrator
:Register
:Login
:AdminVehicleCatalog
:VechileRecord
1.Register(Username,Password)
2 .VehicleCatalog(List<All>)
ViewAllRecords(List<All>)
2.1 Return(Id)
2.2 Modify(Id)
2.3 Cancel(Id)
:Return-Records
:Modify-Record
:Cancel-Record
:Transactions
Only One Administrator can logged in
SearchRecordsHistory(StartDate,EndDate,Make,Model,FirstName,LastName)
:SearchTranscations

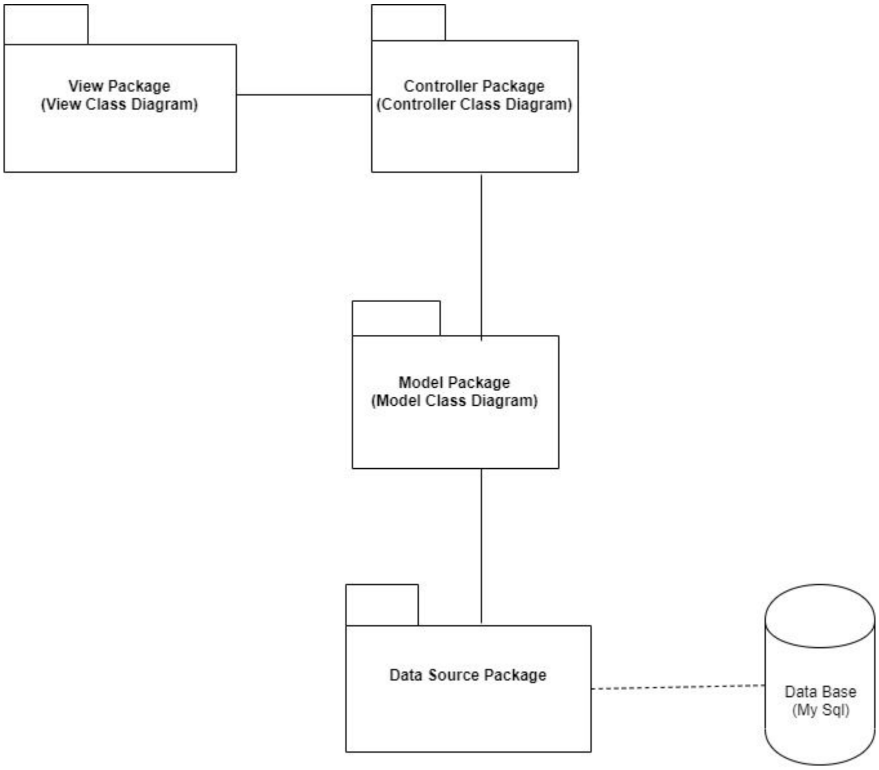
Figure:2.6

Development view : / For iteration 3

Audience : Programmers

Area Concerned : Implementation View

Audience: Programmers. The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the **Package diagram**.



Package Diagram

Figure:2.7 Package Diagram

Process view :

Audience : Integrators

Area Concerned :

Audience: Integrators. The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the **Activity diagram**.

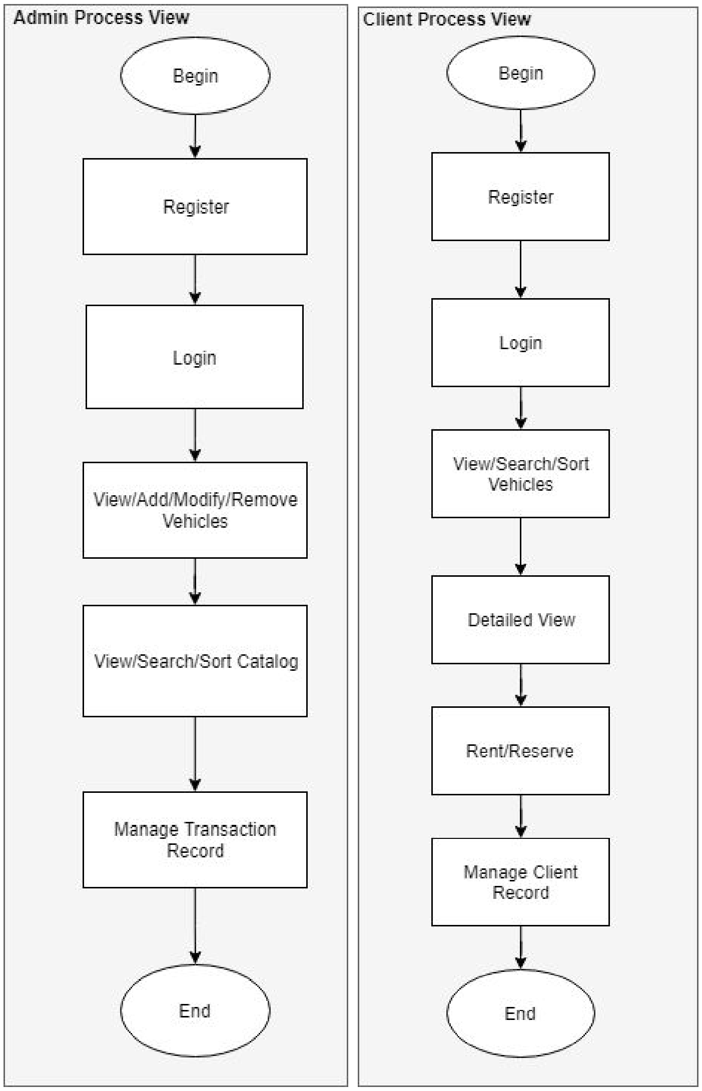


Figure :2.8 Activity Diagram

Physical view

Audience : Deployment managers

Area Concerned :

(also known as deployment view) : Audience: Deployment managers. The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. UML Diagrams used to represent physical view include the **Deployment diagram**.

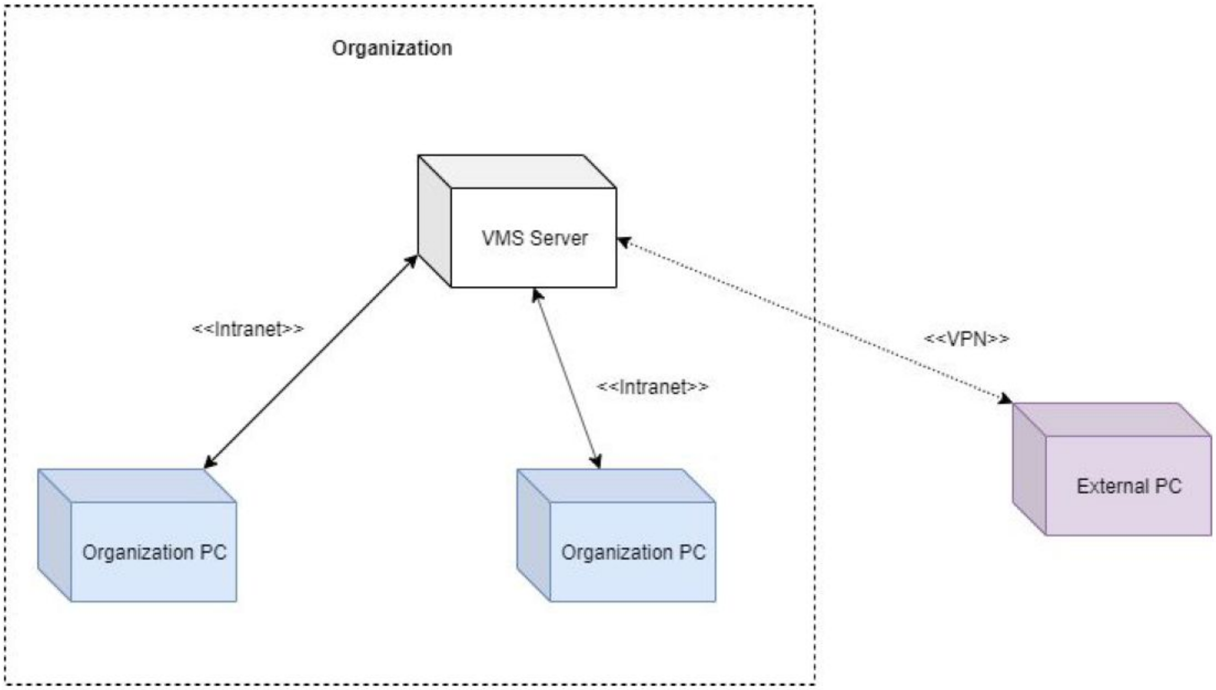


Figure: 2.9 Deployment Diagram

Use case view

Audience : All the stakeholders of the system

Area Concerned :

(also known as Scenarios) : Audience: all the stakeholders of the system, including the end-users. The description of the architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. Related Artifacts : **Use-Case Model**.

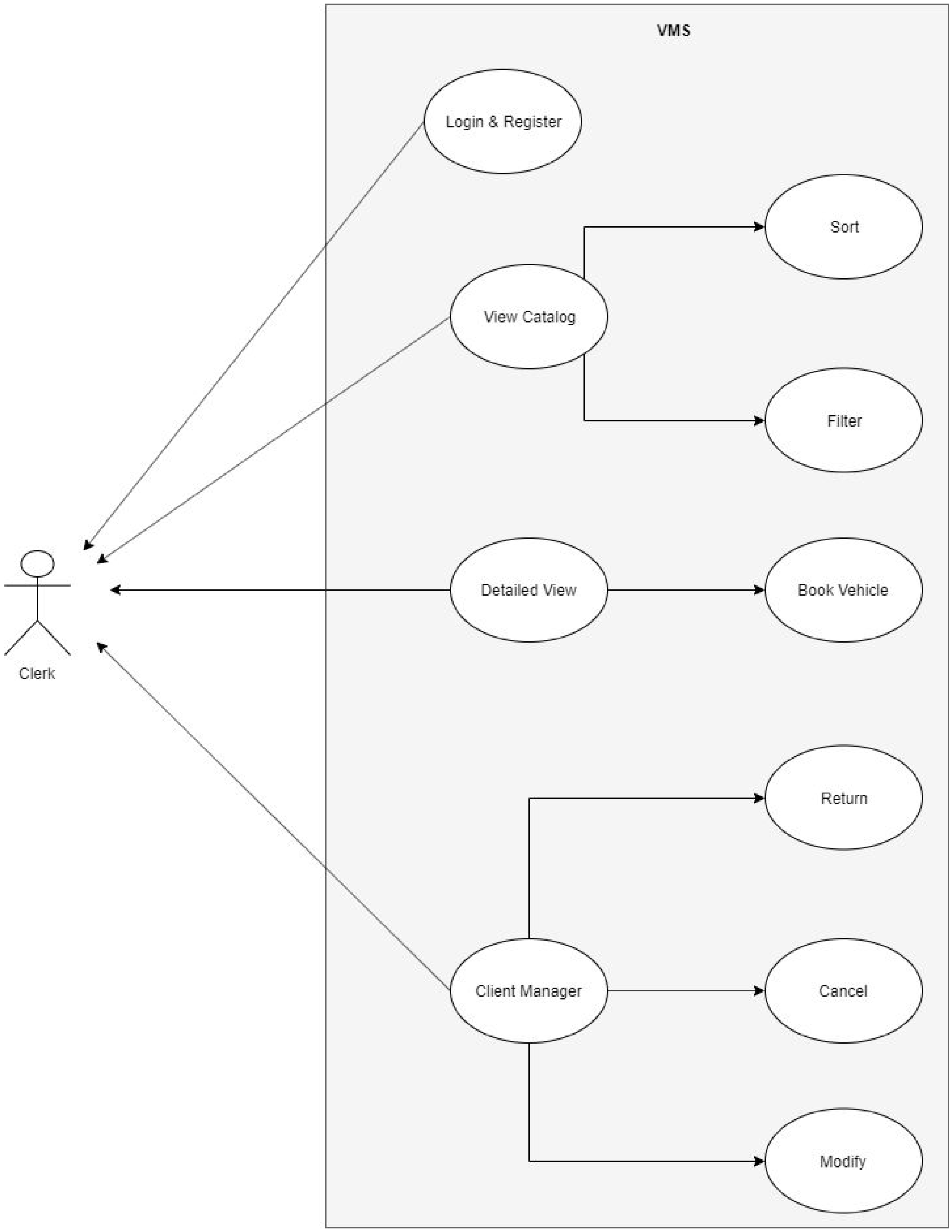


Figure:3.0 Use-Case Model

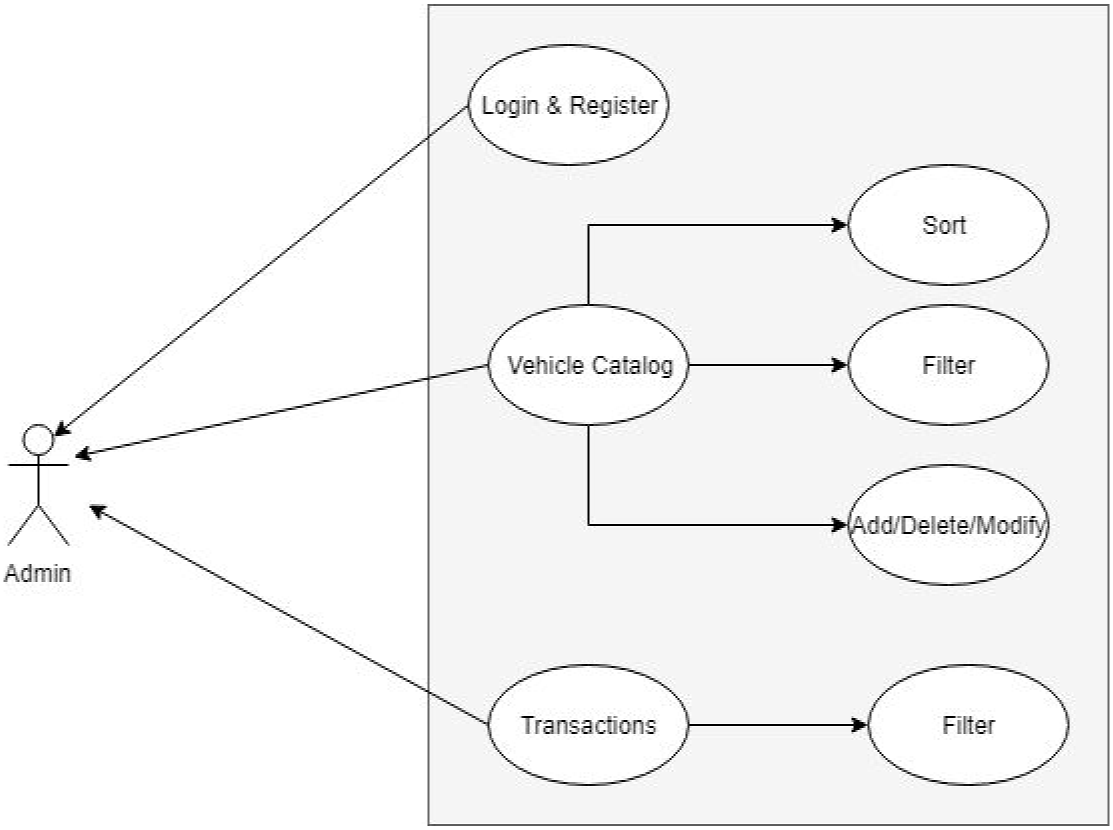


Figure 3.1 **USE CASE VIEW OF VRMS**

1. **Login and Registration:** This use case describes how a user can register and login to the Vehicle Management System. Username and Password is used for authorization to gain access into the system.Encryption was maintained for the password.
2. **View Catalog:** This use case describes how clerks can view the vehicles added to the system. It also supports filtering and sorting of the vehicle list to enhance user interaction.
3. **Detailed View:** This use case allows the clerk to view the complete details of the selected vehicle(Make, Model, Type, Year, Color, Licence Plate) and also check if it is available or not. It also supports a navigation button to navigate vehicles in detailed view.

4. **Book Vehicle:** This use case allows a clerk to reserve a vehicle for his client, booking is done by storing all the necessary information of client(first name, last name, licence number, licence validity, and phone number). The system also maintains a timestamp of when the booking was done.
5. **The Client Manager :** This use case describes how client records can be managed. Return of vehicle , cancellation of a booking and modification of a booking is handled in this use case.
6. **Vehicle Master** : This use case describes how administrator can add vehicles to the system. It also supports filtering vehicle list to enhance user interaction.
7. **Transactions Screens** : This use case describes how administrator can view the information which is maintained and viewed on transactions screens. It also supports filtering the list via various parameter sp that administrator can view overall current availability of the vehicle.

Audience: Data specialists, Database administrators. Describes the architecturally significant persistent element s in the data model . Related Artifacts: **Data model**.

Architectural requirements: goals and constraints

Requirements are already described in SRS. In this section describe *key* requirements and constraints that have a significant impact on the architecture. **For iteration 3**

18.Functional requirements (Use case view)

The overview below refers to architecturally relevant Use Cases from the Use Case Model (see references).

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case(s) or scenario(s).	**Name of case(s) or scenario(s).**	**Description on why this use case or scenario is relevant to the architecture.**	**Section number where this use case or scenario is addressed in this document.**
User Login	User able to login via username and password than : 1.**Success** : Logged In successfully. 2.**Failure** : Not able to login since username and password didn't not matched and redirected back to the login page.	We selected Login page because we need to store who log in and maintain the user name for each user so that we can maintain the history and know what a user in doing after logging and booking a vehicle on behalf of someone.	**SRS Document Figure:*4.2***
User Registration	User can register his information based on which general information is asked by the software.And than can try for login. 1. **Success** : Able to provide all the details	Since user can logged in it should have a master when he/she can give his/her basic details and than can perform log In.	**SRS Document Figure: Use case:4.2**

	to register basic information. 2.**Failure** :If not saved properly can not able to login.		
Vehicle Searching page	User can search based on the following selections : Make,Type,Year and Model.Following things will happen : 1.You can select search button and filter based on the following condition. 2.Apart from the dataset which comes based on filtering can be sorted based on pressing the button respectively.Like ascending/Descending based on [a-z],[z-a] [alphabets] and [Lowest-Highest] ,[highest-lowest] [numbers]	A user after logging will be landed on the vehicle search page.Where he can check what kind of vehicle he/she would be looking for according to his choice or can perform sorting based on alphabets on make model type and year and can be checked and can be opened or select via view details button.Since a user looking for a vehicle can look for book and he/she can book on clicking view details button.	**SRS Document Figure:*UseCase 4.2, Figure 4.8***

Detailed View	Based on view detail button user can user can check his/her vehicle detail and can also go back to the screen he came from or confirm the booking bu going on the booking view page.	In Detailed view you can edit/add the vehicle details and check the availability also.since checking the availability is an important aspect and you can proceed to click on book now where you can give start and end date.	**SRS Document Figure:*UseCase 4.1***
Booking View	user can put the start and return date and can book the vehicle.	In booking view you need to give user general info and the most important things the start and return date of the vehicle when you are renting for a particular period.	**SRS Document Figure:*UseCase 4.10***
Client Page	All the reservation of the following vehicles would be shown on the clients page with their dates and user can add/edit-modify/Return vehicles.	Client pages manages booking done by user/clerk. You can add/edit/return/cancel the booking which you have done.This page shows data in tabular table.	**SRS Document Figure: 4.8**

Clients Modify	Clients can modify/add and return the vehicles on this page.	You can add/edit/return and cancel the vehicle you rented in edit mode.	**SRS Document Figure: 4.8**
Vehicle Master	Administrator can add by clicking on the add button or filter the records based on any search criteria he wants like type,year, Make,Model.At the same time Admin can delete the vehicle if he wants.	Since the main actor for this scenario is administrator and admin has rights to add/view details to update the vehicle master.	**SRS Document Figure: 5.1**
Vehicle Add/Edit	Admin can add and add screen will open in the next window which ask him to add the attributes for the new vehicles or admin can edit the details too.	Administrator is the main actor for add/edit screen and can save a new vehicle or update a new vehicle.	**SRS Document Figure: 5.1**
Transaction Screen	Administrator can Filter and view history based on client/Vehicle info or by particular start/end date.	Based on the transaction done by clerk the admin can view the current record for the clerk(updated record).At the same	**SRS Document Figure: 4.9**

		time admin can filter the same based on many criteria like per client or per vehicle or start date or end date.	

19.Non-functional requirements

Persistence Design:

We have followed the object oriented mapping and created separate gateways for each table.Below are the gateway designs of tables used in the application.

19.1:Clerks: Below is the description how clerks achieve respective functionality as given in SRS document.To attain persistence clerks model class is mapped with the Databases.Whenever a clerk tries to do some action or any DML command (Such as Select,Insert,Delete,Update) it maps with Clerks data mapper class and ask for respective sql command which is written in Clerks Data gateway to return the result set to achieve the respective functionality.

ClerksModel -->ClerksDataMapper → Clerks Data Gateway → Database [Execution of command via Batch files and return the result set for a query fired by user]

Database →Clerk Data Gateway →ClerksDataMapper →ClerksModel →Functionality achieved in controller where Business logic is written.

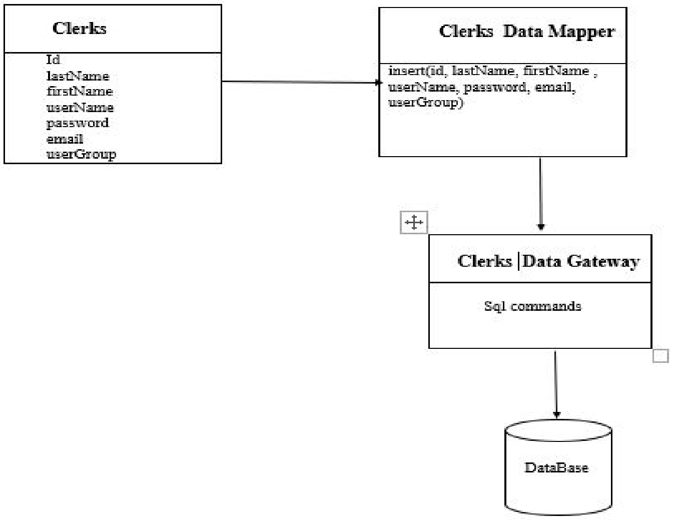


Figure :3.2 Clerks ORM

19.2:Clients: Below is the description how clients achieve respective functionality as given in SRS document.To attain persistence clerks model class is mapped with the Databases.Whenever a client tries to do some action or any DML command (Such as Select,Insert,Delete,Update) it maps with clients data mapper class and ask for respective sql command which is written in clients Data gateway to return the result set to achieve the respective functionality.

ClientsModel -->ClientsDataMapper → Clients Data Gateway → Database [Execution of command via Batch files and return the result set for a query fired by user]

Database →Clients Data Gateway →ClientsDataMapper →ClientsModel →Functionality achieved in controller where Business logic is written.

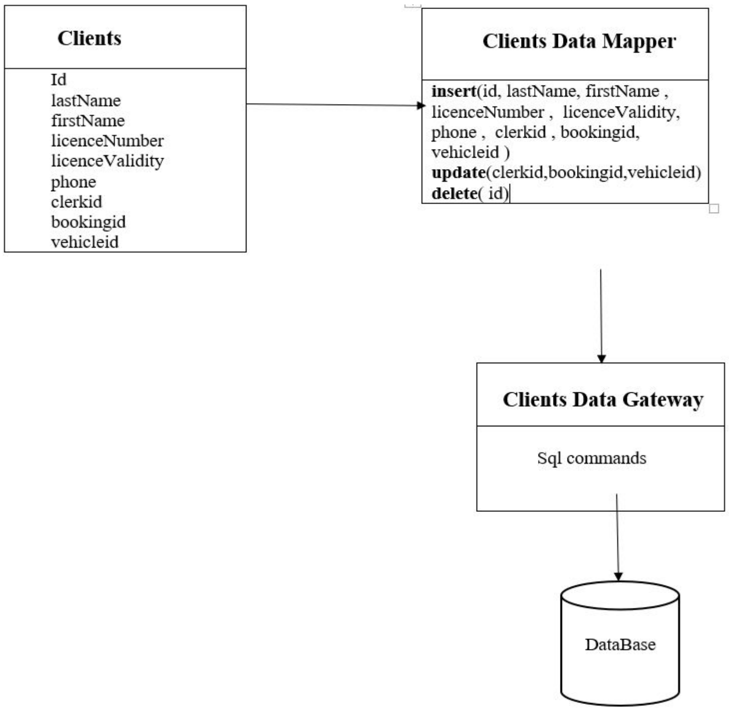


Figure:3.3 Clients ORM

19.3:Bookings: Booking is the most important part in VMS.In which Multiple clerks try to book vehicles for clients to book vehicle based on rental or reservation for future date.Since Booking consists of many insertions and Updating in database it hard to maintain at the same time when multiple clerks try to book the same vehicle. To achieve the same many things need to be kept in mind.Since we are maintaining versioning to handle concurrency.
Booking Data Mapper consists of Various commands which will get executed in data gateway.As shown in the picture below.

BookingsModel -->BookingDataMapper → Booking Data Gateway → Database [Execution of command via Batch files and return the result set for a query fired by user]
Database →Booking Data Gateway →BookingDataMapper →ClientsModel →Functionality achieved in controller where Business logic is written.

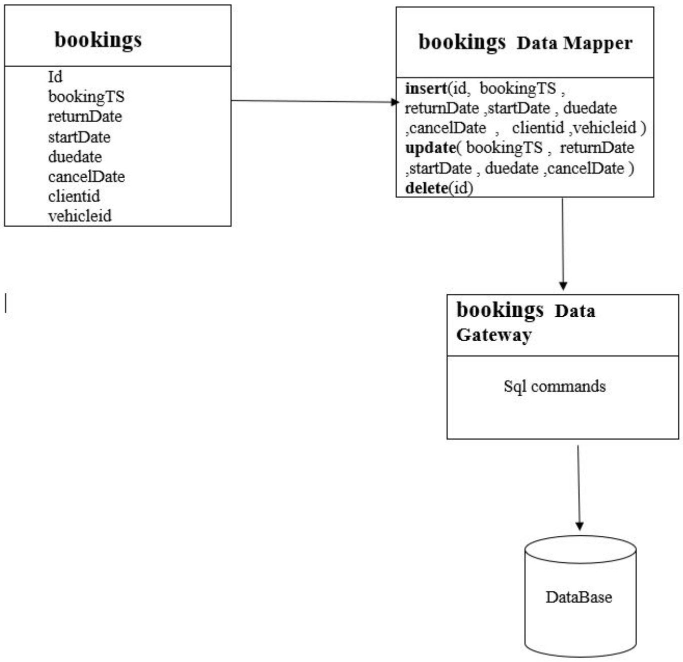

bookings
Id
bookingTS
returnDate
startDate
duedate
cancelDate
clientid
vehicleid
bookings Data Mapper
insert(id, bookingTS , returnDate ,startDate , duedate ,cancelDate , clientid ,vehicleid)
update(bookingTS , returnDate ,startDate , duedate ,cancelDate)
delete(id)
bookings Data Gateway
Sql commands
DataBase

Figure:3.4 Bookings ORM

19.4:Vehicles: Since the project is about to rent the Vehicles for clients.So the importance of Vehicles for insert/update/Delete is important to all clerks who are using the application.Below is the description how Vehicles achieve respective functionality as given in SRS document.To attain persistence Vehicles model class is mapped with the Databases.Whenever a client tries to do some action or any DML command (Such as Select,Insert,Delete,Update) it maps with Vehicles data mapper class and ask for respective sql command which is written in Vehicles Data gateway to return the result set to achieve the respective functionality.

VehiclesModel -->VehiclesDataMapper → Vehicles Data Gateway → Database [Execution of command via Batch files and return the result set for a query fired by user]

Database →Vehicles Data Gateway →VehiclesDataMapper →VehiclesModel →Functionality achieved in controller where Business logic is written.

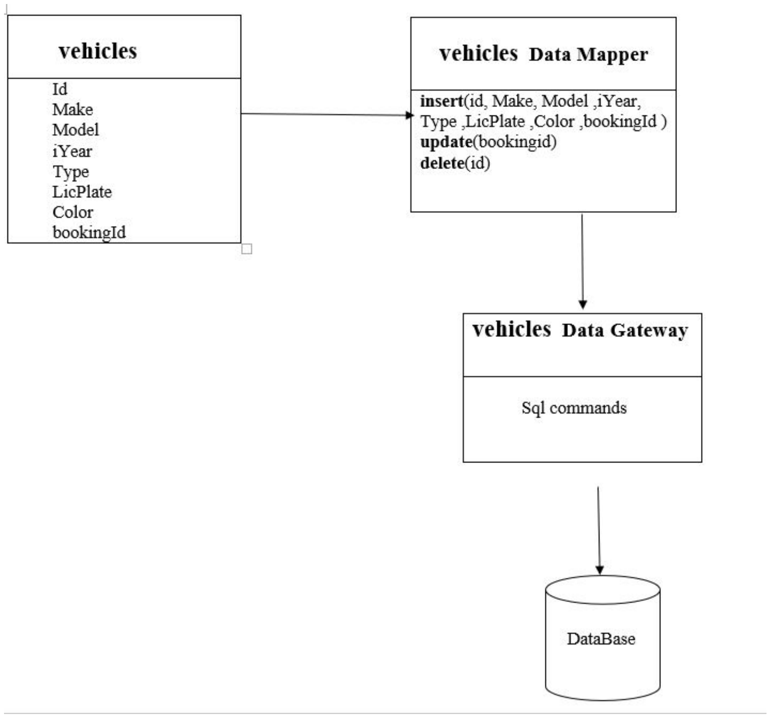


Figure:3.5 Vehicles ORM

Describe the architecturally relevant non-functional requirements, i.e. those which are important for developing the software architecture. Think of security, privacy, third-party products, system dependencies, distribution and reuse. Also environmental factors such as context, design, implementation strategy, team composition, development tools, time to market, use of legacy code may be addressed.

Usually, the non-functional requirements are already in place and can be referenced here. This document is not meant to be the source of non-functional requirements, but to address them. Provide a reference per requirement, and where the requirement is addressed.

Source	**Name**	**Architectural relevance**	**Addressed in:**
e.g. Vision, SRS.	Name of requirement.	Description on why this requirement is relevant to the software architecture.	Section number where this requirement is addressed in this document.

20.Use case view (Scenarios)

The scenarios (or functional view) represent the behavior of the system as seen by its actors. Use case scenarios describe sequences of interactions between actors and the system (seen as a black box) as well as between the system and external systems .The *UML use case diagram* is used to capture this view.

Use Case No.	**Use Case**	**Type**
UC-1	Register	Non-critical
UC-2	Login	Non-critical
UC-3	View Catalog	Non-critical
UC-4	Search Catalog	Critical
UC-5	View Vehicle Details	Non-critical
UC-6	Book Vehicle	Critical
UC-7	Manage Client Records	Critical
UC-8	View Bookings	Non-critical

UC-9	Search Bookings	Critical
UC-10	Manage Vehicle Record	Critical

Logical view

The logical view captures the functionality provided by the system; it illustrates the collaborations between system components in order to realize the system's use cases. Describe the architecturally significant logical structure of the system. Think of decomposition in tiers and subsystem. Also describe the way in which, in view of the decomposition, Use Cases are technically translated into Use Case Realizations.

Layers, tiers etc.

Layered architecture style is the most common architecture style. Horizontal layers are grouped into modules or components with different functionalities, so each layer plays a specific role within the framework. A presentation layer would be responsible for managing all user interface and browser communication logic, while a domain layer would be responsible for enforcing specific business rules pertaining to a request. The presentation layer doesn’t need to know or worry about how to get customer data; it only needs to display that information on a screen in particular format. Similarly, the domain layer doesn’t need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the data source layer, perform business logic against the data and pass that information up to the presentation layer.

Here is a simple Rentit application using 3 tier architecture. If a clerk wants to book a vehicle , he will first need to interact with the presentation layer(Booking form), which is the top-most layer in this diagram, the main function of this layer is to translate tasks and results to something the clerk can understand. And then the clerk request is handled by middle layer which is called domain layer(bookingController). It is used to coordinate to application layer and makes logical decision by clerks request. Domain layer will validates the data received from booking form and then it goes to data source layer, which has a control to the actual database, to perform the database writing.

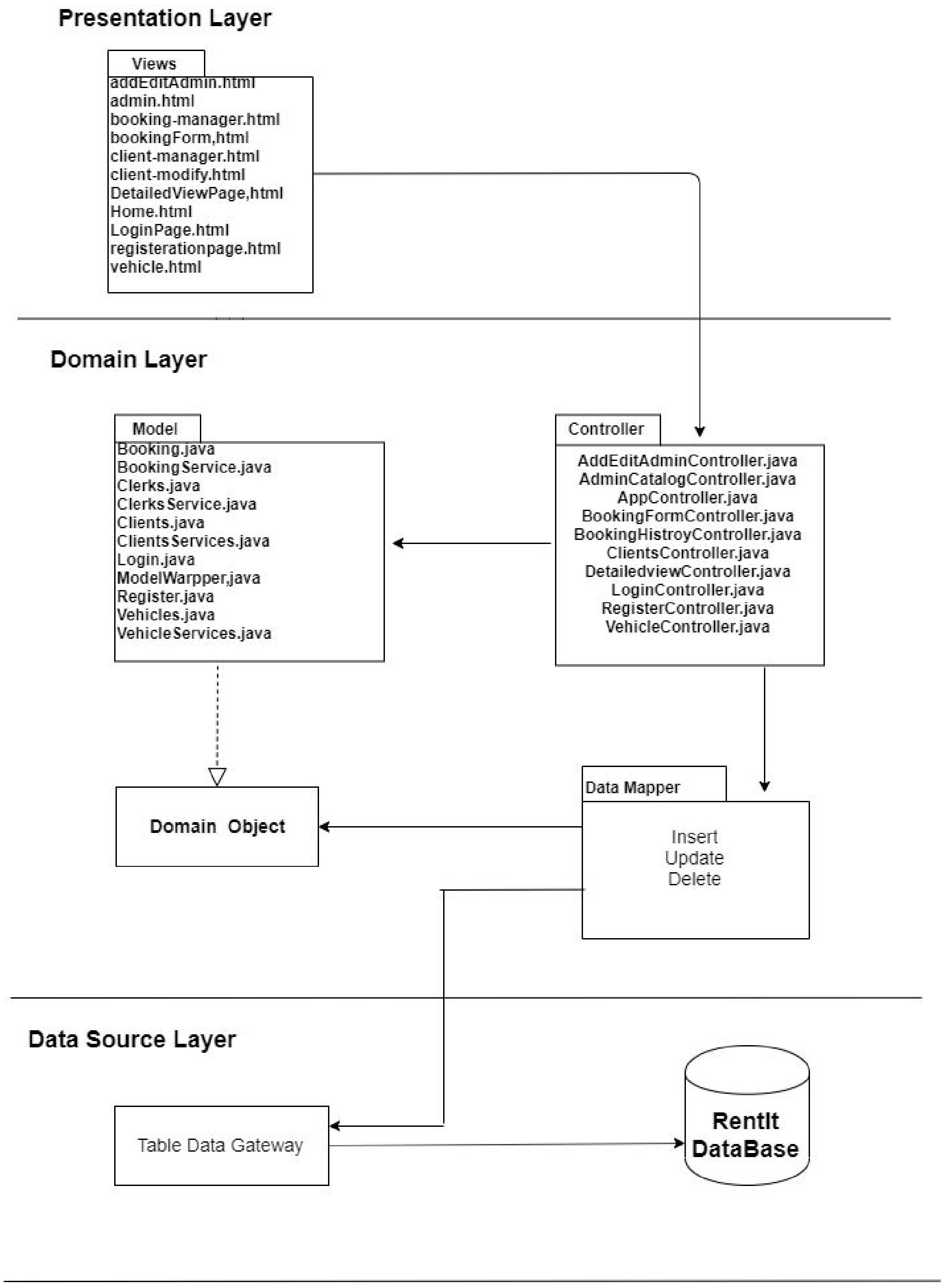


Figure:3.6 3 tier Architecture

Subsystems

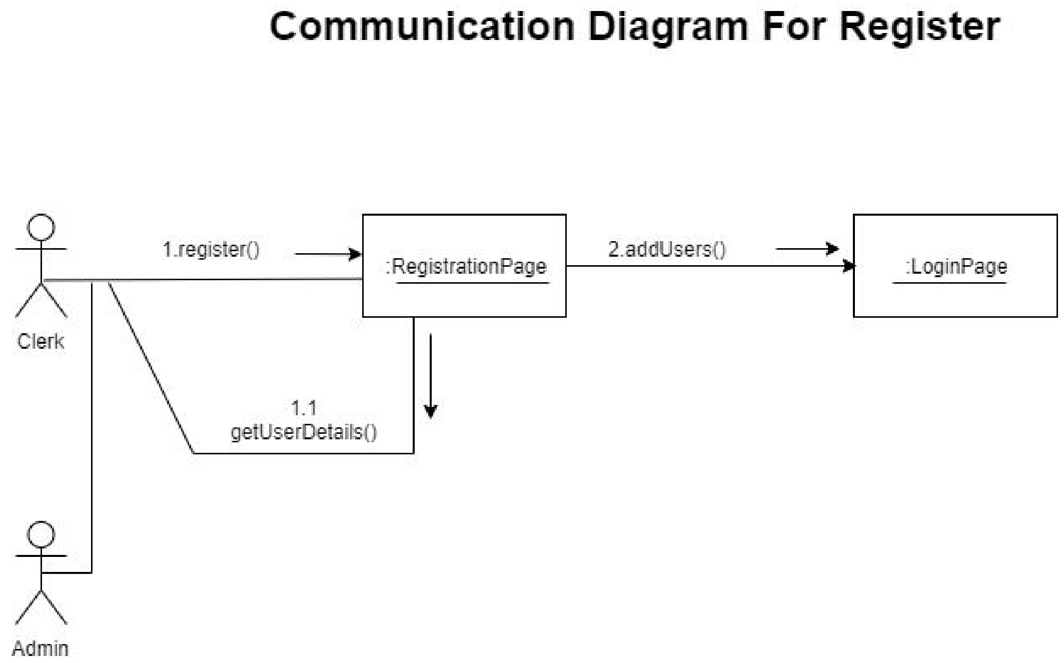
Describe the decomposition of the system in subsystems and show their relation.

Architecturally significant design packages

Describe packages of individual subsystems that are architecturally significant. Each package includes a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

Use case realizations

In this section you have to illustrate how use cases are translated into *UML interaction diagrams*. Give examples of the way in which the Use Case Specifications are technically translated into Use Case Realizations, for example, by providing a sequence-diagram. Explain how the tiers communicate and clarify how the components or objects used realize the functionality.



Communication Diagram For Register

Figure 3.7: CD-1

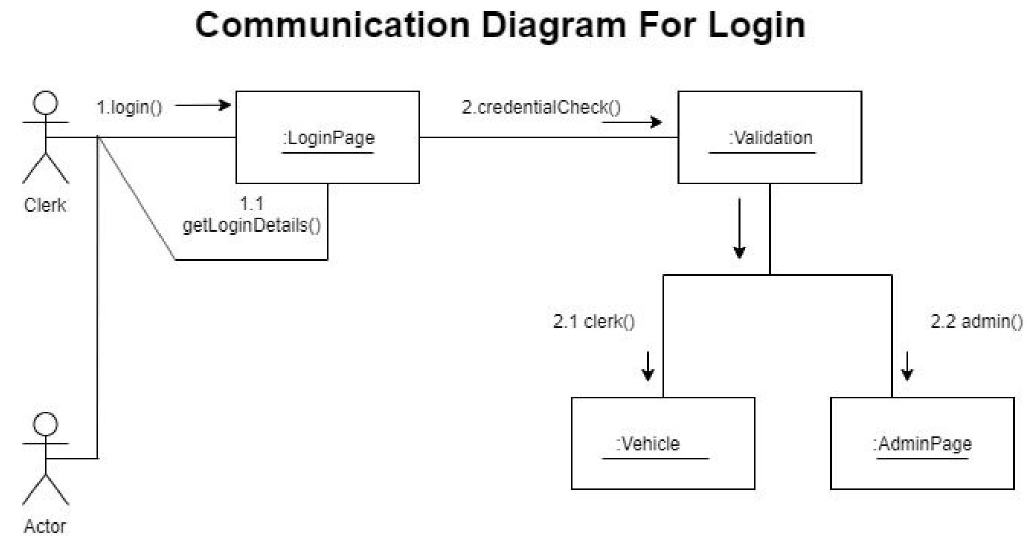


Figure 3.8: CD-2

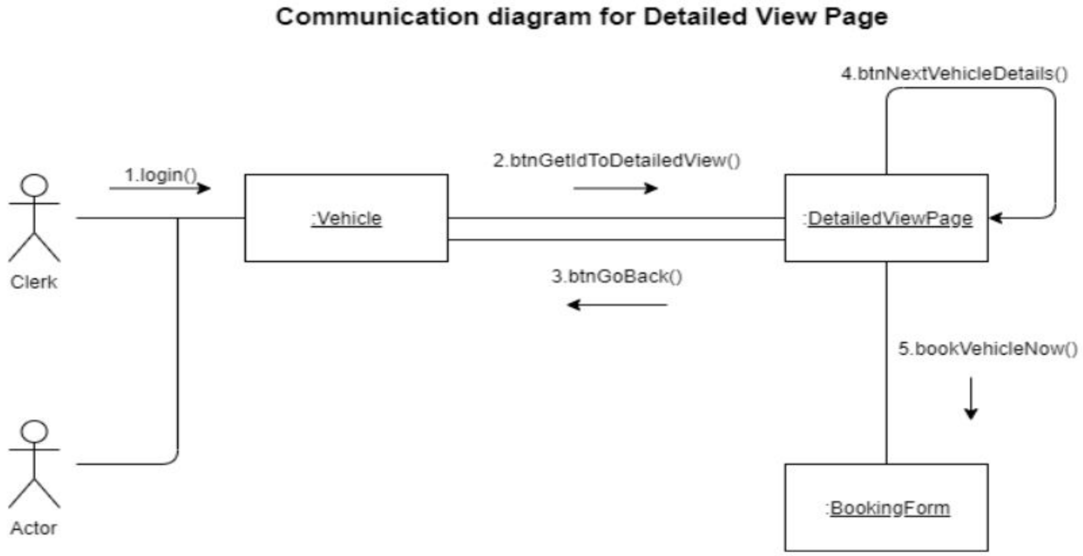

Communication diagram for Detailed View Page
4.btnNextVehicleDetails()
2.btnGetIdToDetailedView()
1.login()
:Vehicle
:DetailedViewPage
Clerk
3.btnGoBack()
5.bookVehicleNow()
Actor
:BookingForm

Figure 3.9:CD-3

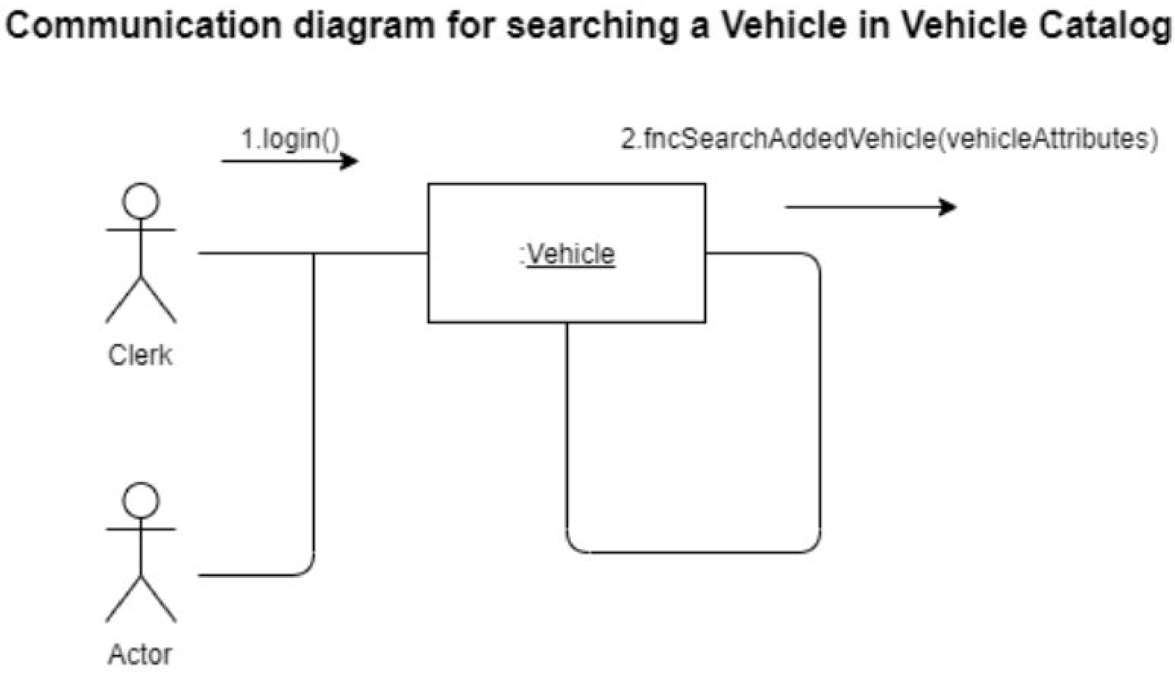

Communication diagram for searching a Vehicle in Vehicle Catalog
1.login()
2.fncSearchAddedVehicle(vehicleAttributes)
:Vehicle
Clerk
Actor

Figure 4.0: CD-4

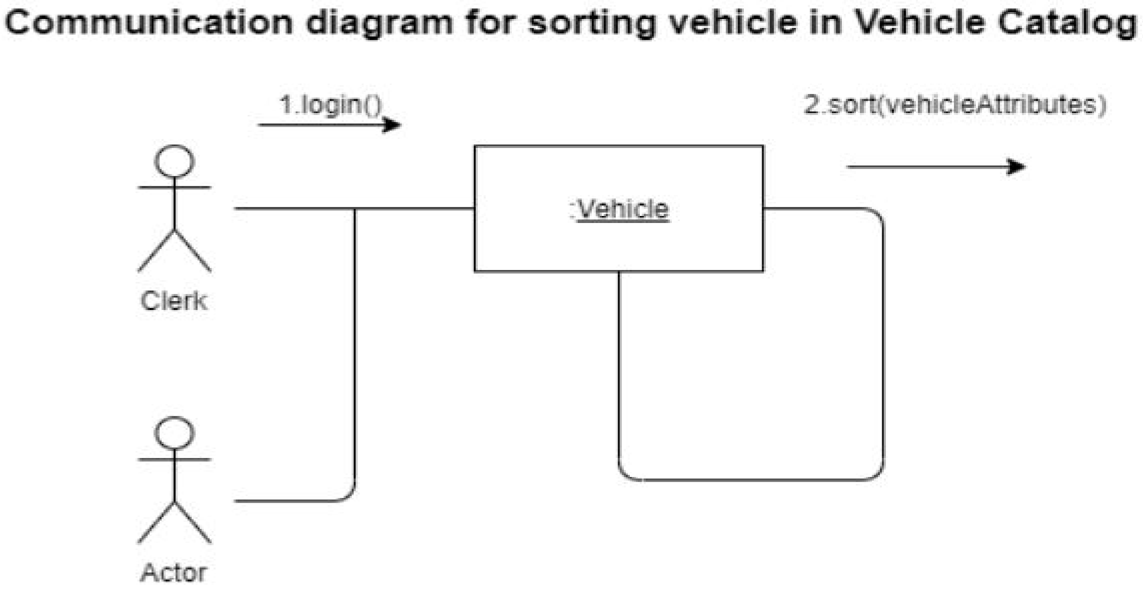

Communication diagram for sorting vehicle in Vehicle Catalog
1.login()
2.sort(vehicleAttributes)
:Vehicle
Clerk
Actor

Figure 4.1: CD-5

Communication diagram for adding a new vehicle as an admin

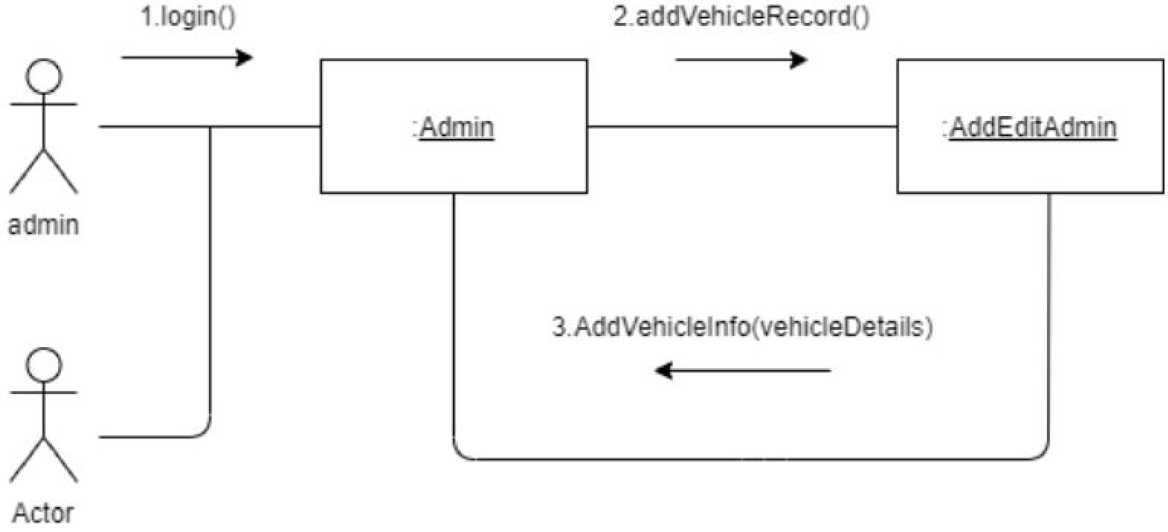


Figure 4.2: CD-6

Communication diagram for updating a vehicle record in admin page

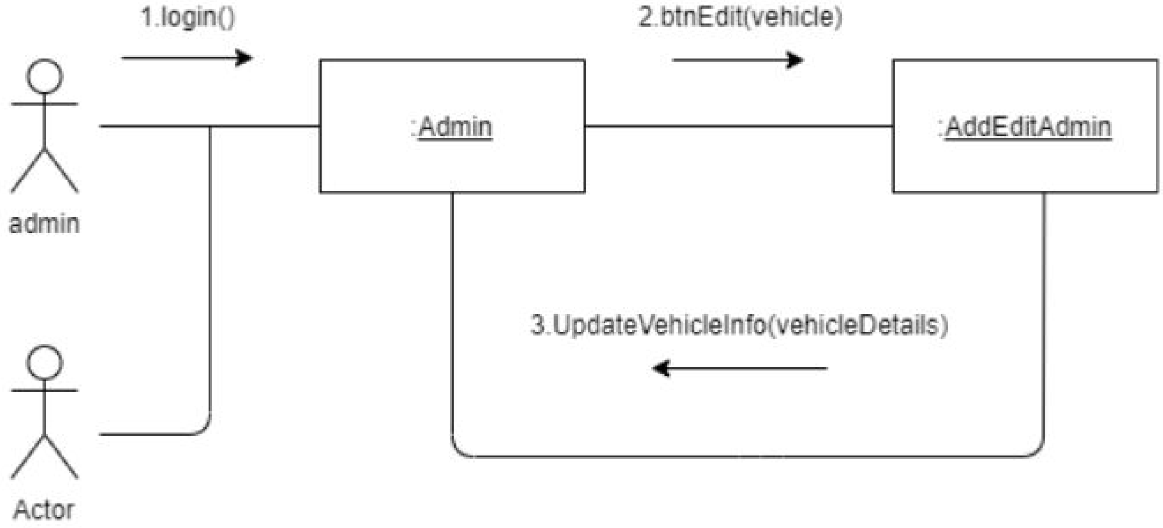


Figure 4.3: CD-7

Communication diagram for deleting a vehicle record in admin page

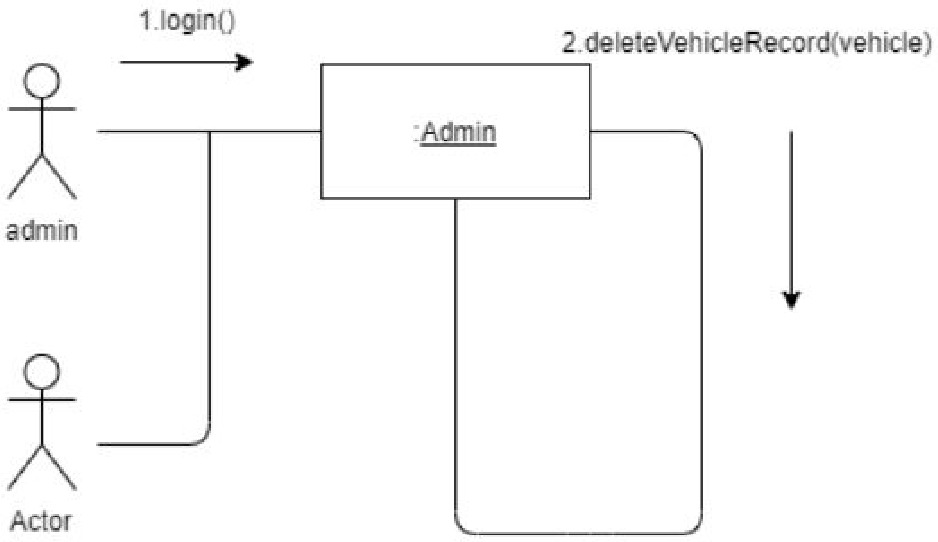


Figure 4.4: CD-8

Communication Diagram For Booking Page

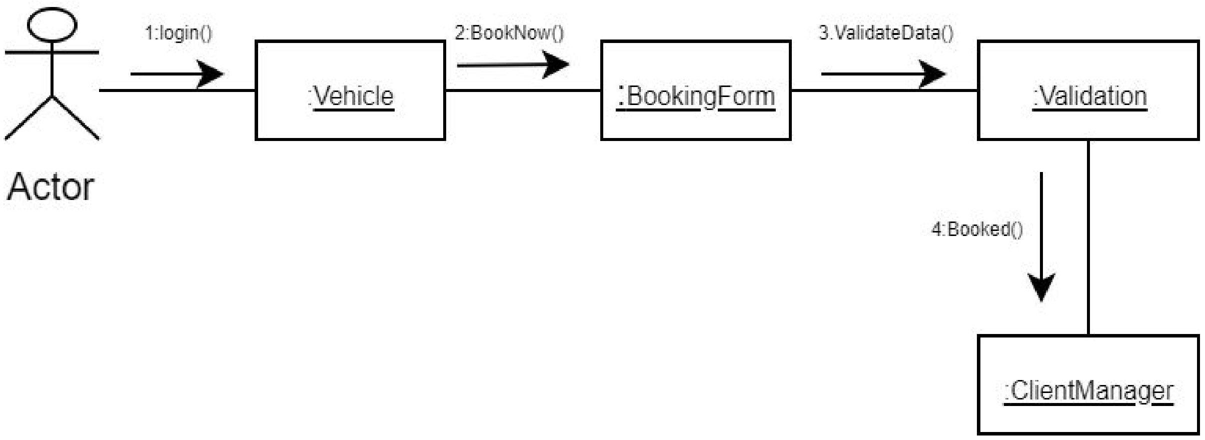


Figure 4.5: CD-9

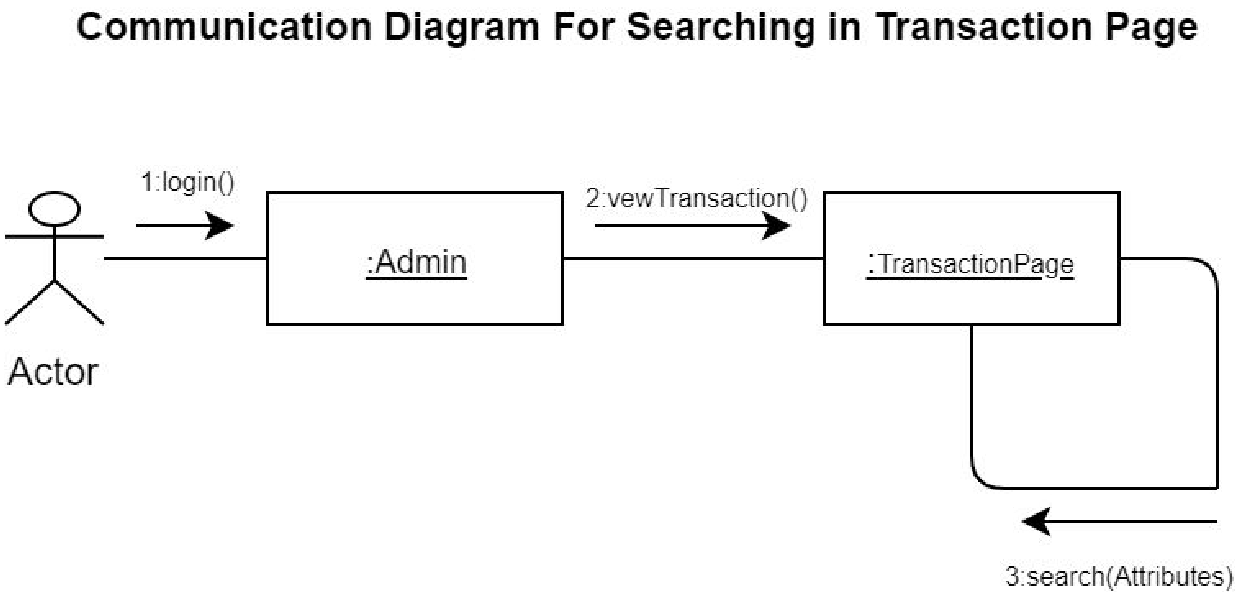


Figure 4.6: CD-10

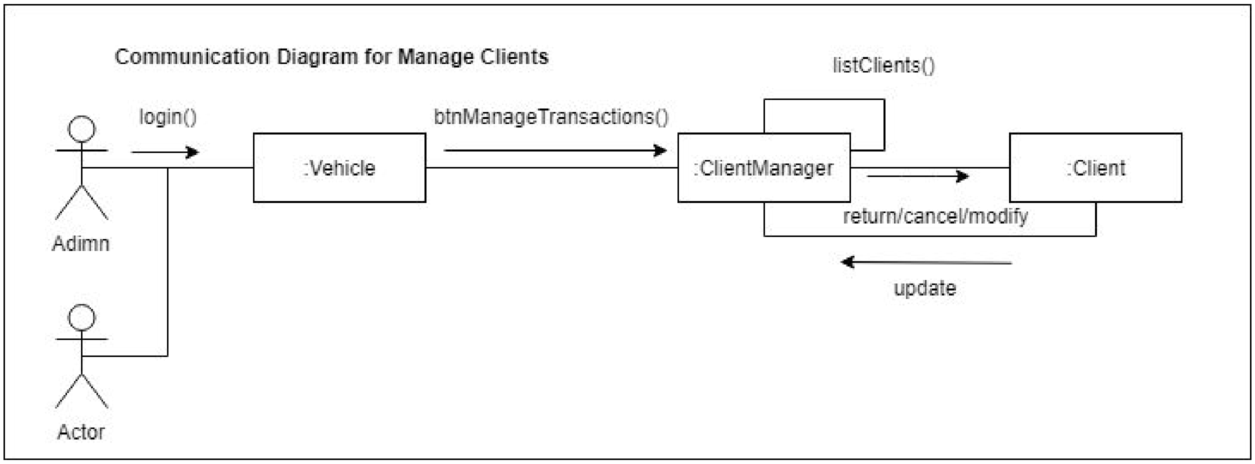


Figure 4.7: CD-11

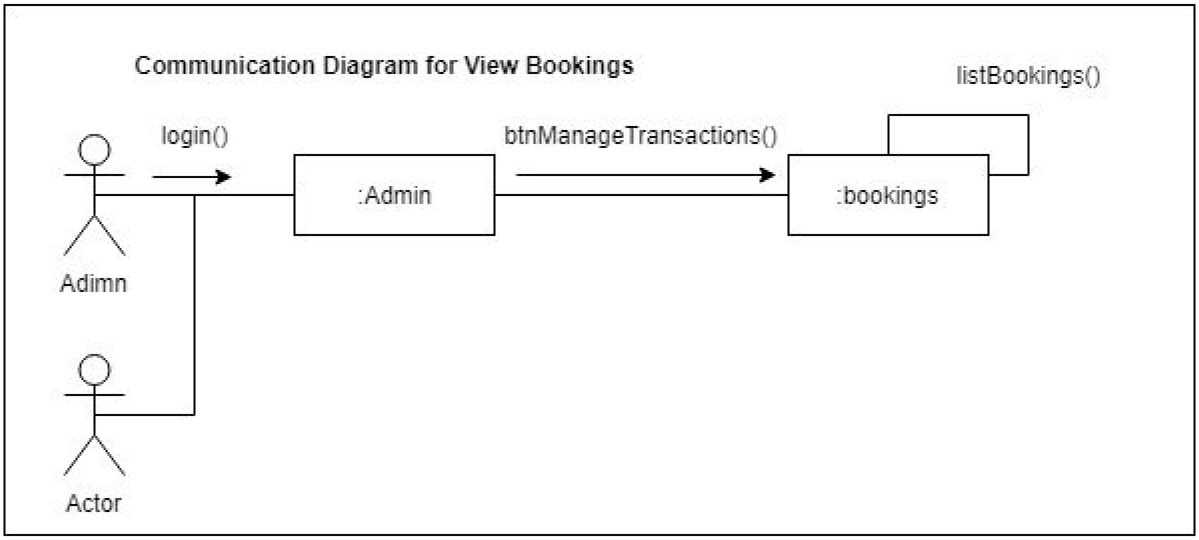
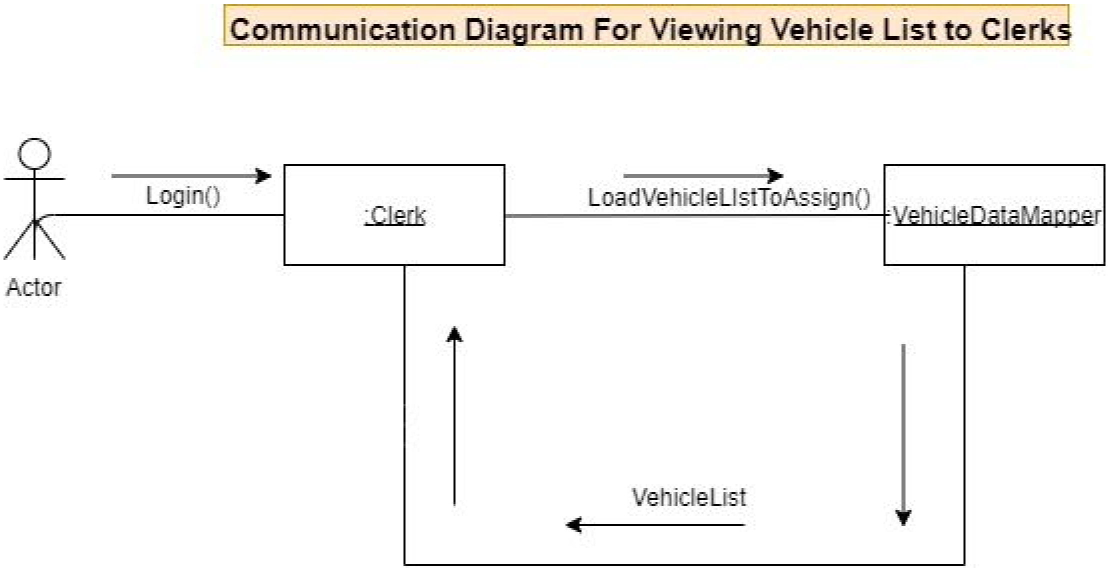


Figure 4.8: CD-12



Communication Diagram For Viewing Vehicle List to Clerks
Login()
:Clerk
LoadVehicleLIstToAssign()
:VehicleDataMapper
Actor
VehicleList

Figure 4.9 : CD:13

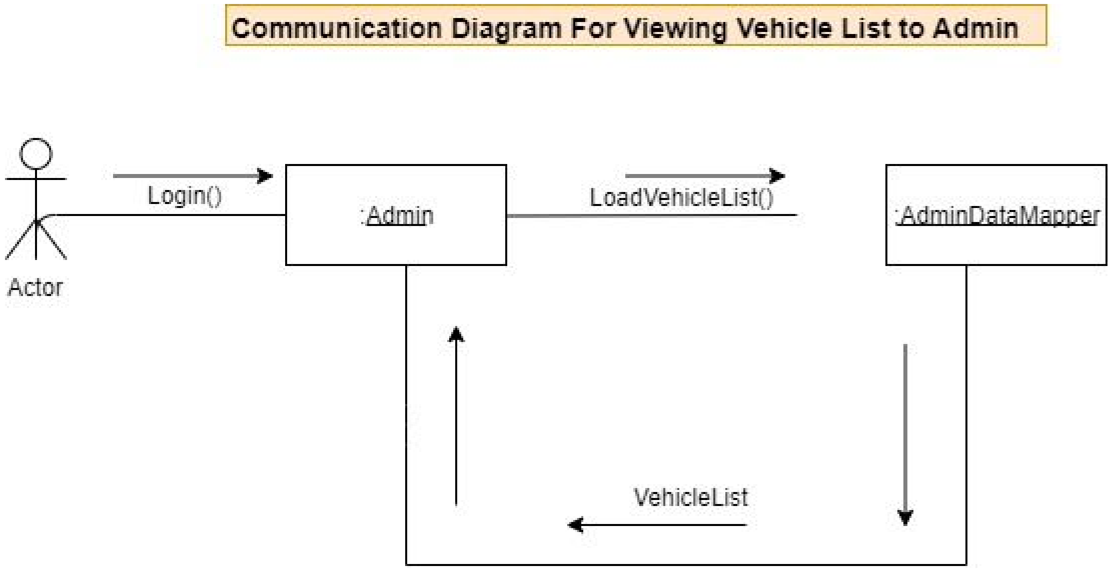


Figure 5.0 : CD:14

Sequence Diagrams for Critical Use cases

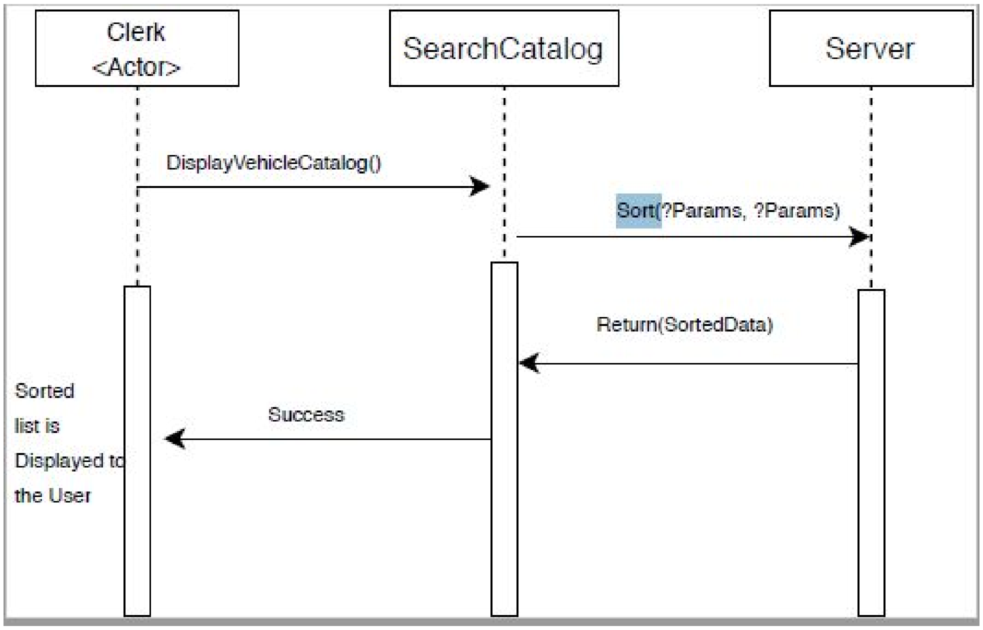


Figure 5.1:UC-4

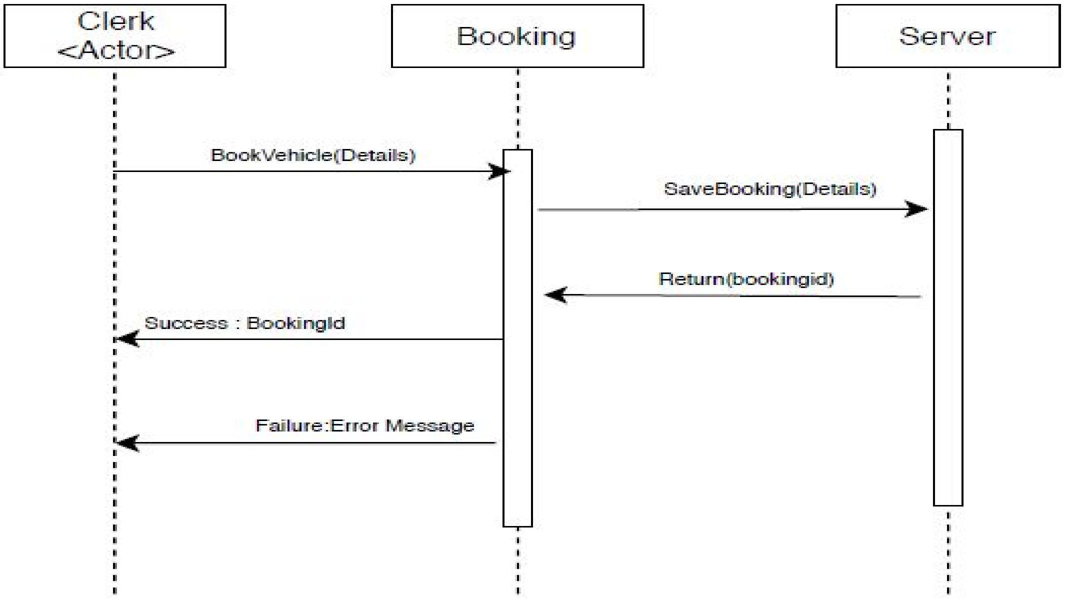


Figure 5.2:UC-6

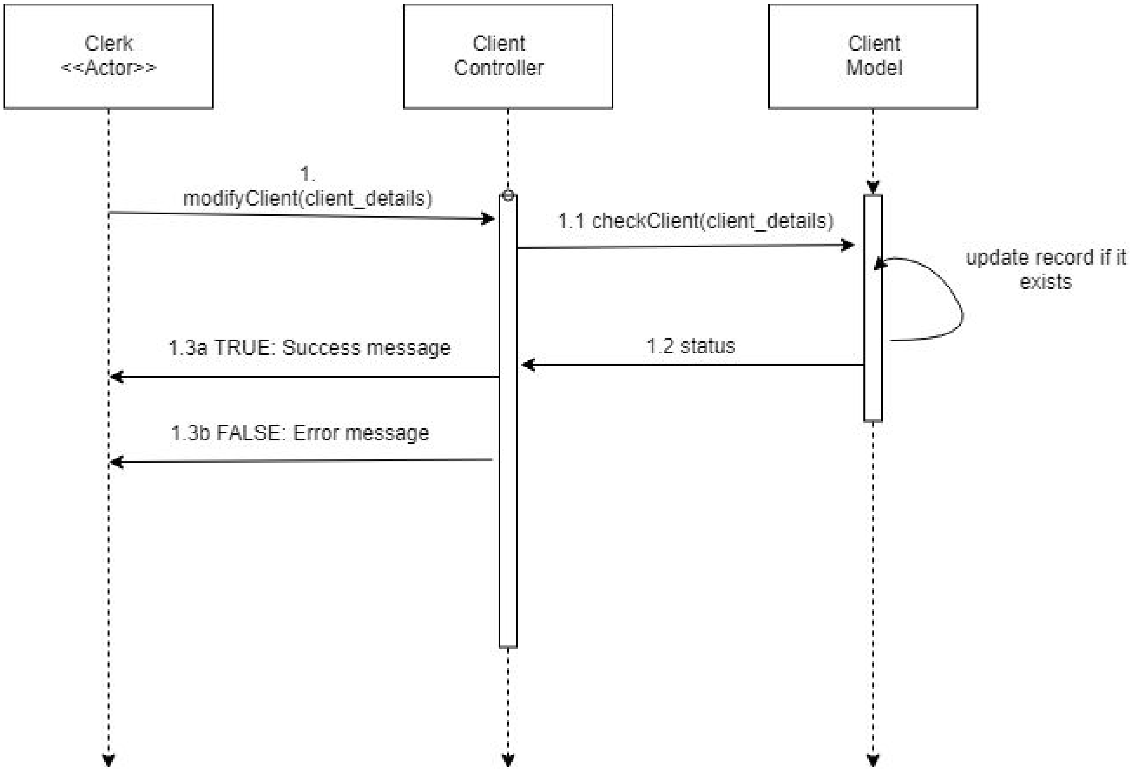

Clerk
<<Actor>>
Client
Controller
Client
Model
1.
modifyClient(client_details)
1.1 checkClient(client_details)
update record if it
exists
1.2 status
1.3a TRUE: Success message
1.3b FALSE: Error message

Figure 5.3:UC-7

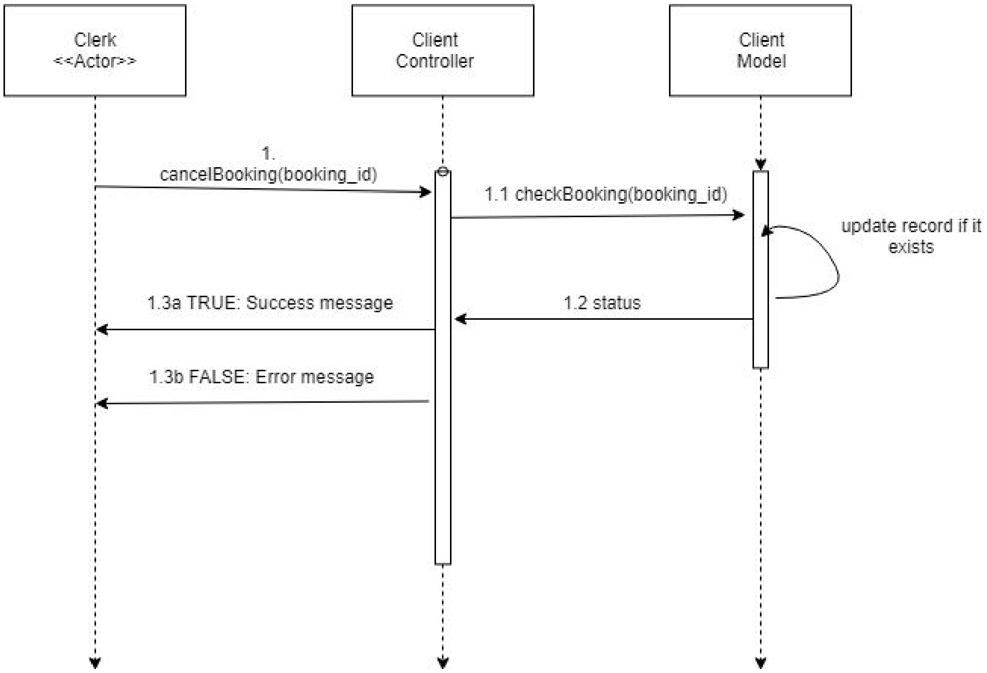

Clerk
<<Actor>>
Client
Controller
Client
Model
1.
cancelBooking(booking_id)
1.1 checkBooking(booking_id)
update record if it
exists
1.3a TRUE: Success message
1.2 status
1.3b FALSE: Error message

Figure 5.4:UC-7

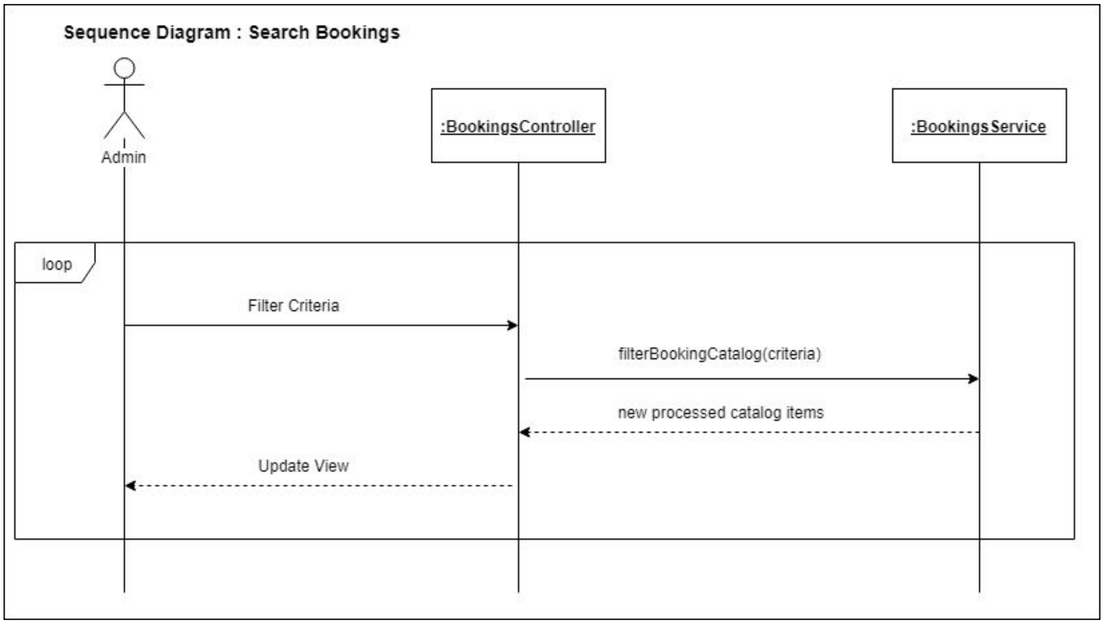
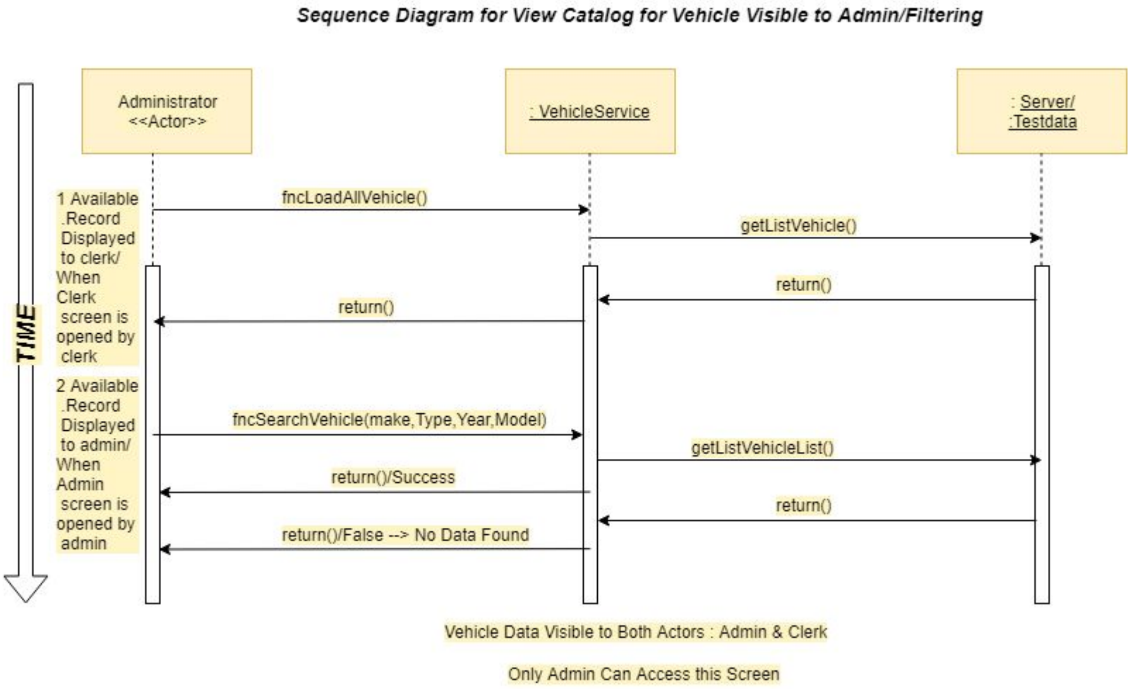
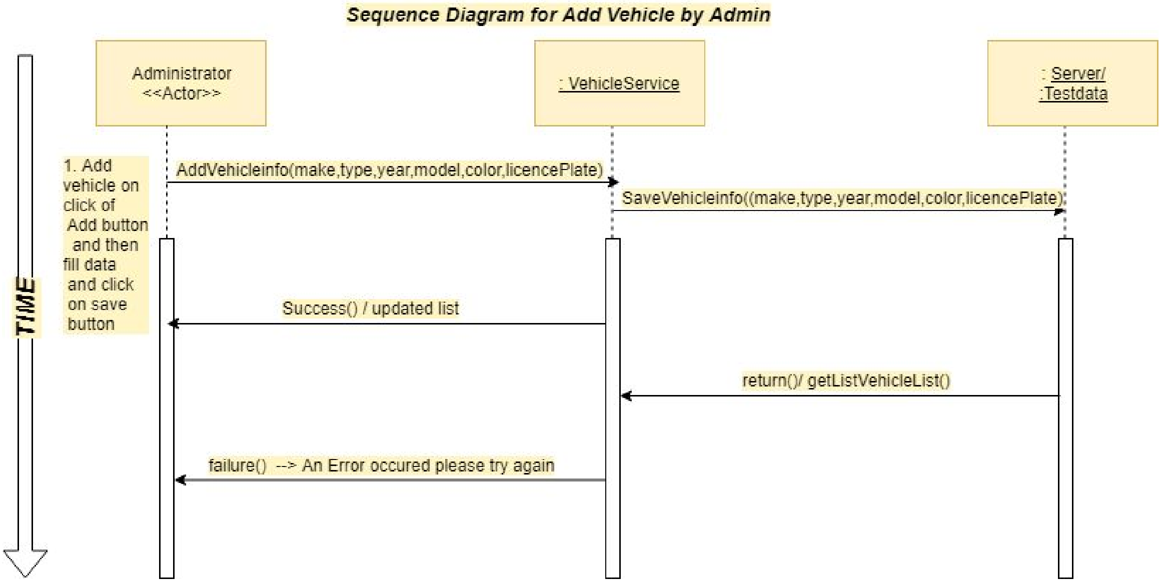

Sequence Diagram : Search Bookings
Admin
:BookingsController
:BookingsService
loop
Filter Criteria
filterBookingCatalog(criteria)
new processed catalog items
Update View

Figure 5.5:UC-9



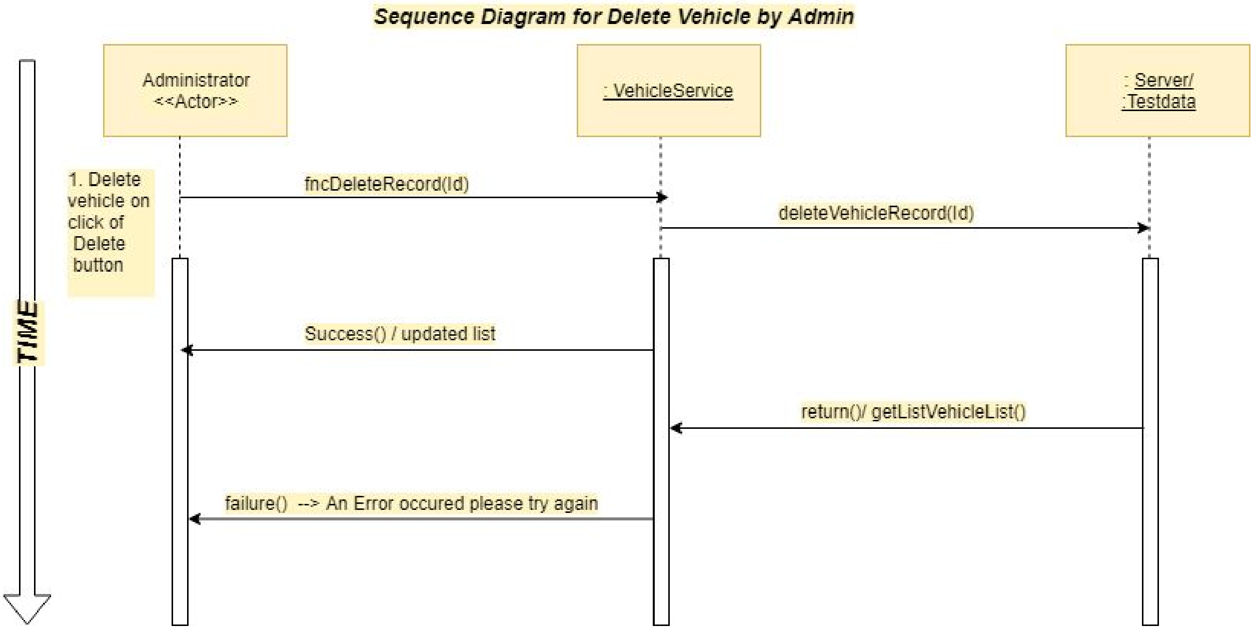
Sequence Diagram for View Catalog for Vehicle Visible to Admin/Filtering
Administrator <<Actor>>
: VehicleService
: Server/ :Testdata
TIME
1 Available .Record Displayed to clerk/ When Clerk screen is opened by clerk
2 Available .Record Displayed to admin/ When Admin screen is opened by admin
fncLoadAllVehicle()
getListVehicle()
return()
return()
fncSearchVehicle(make,Type,Year,Model)
getListVehicleList()
return()/Success
return()
return()/False --> No Data Found
Vehicle Data Visible to Both Actors : Admin & Clerk
Only Admin Can Access this Screen

Figure 5.6:UC-10



Sequence Diagram for Add Vehicle by Admin
Administrator <<Actor>>
: VehicleService
: Server/ :Testdata
TIME
1. Add vehicle on click of Add button and then fill data and click on save button
AddVehicleinfo(make,type,year,model,color,licencePlate)
SaveVehicleinfo((make,type,year,model,color,licencePlate)
Success() / updated list
return()/ getListVehicleList()
failure() --> An Error occured please try again

Figure 5.7:UC-10

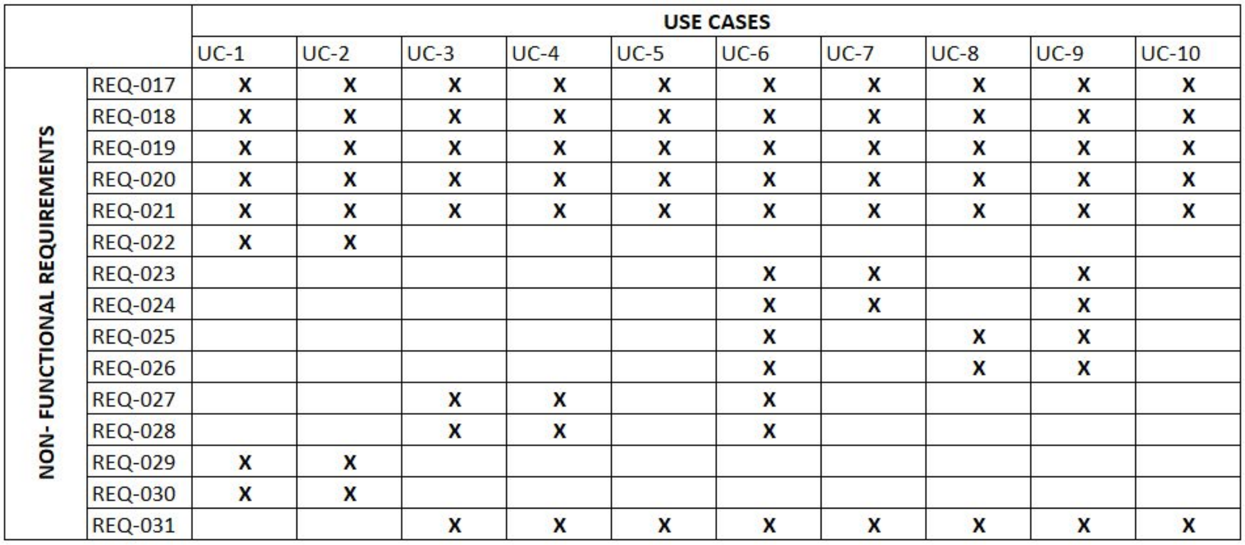


Sequence Diagram for Delete Vehicle by Admin
Administrator
<<Actor>>
: VehicleService
: Server/
:Testdata
TIME
1. Delete vehicle on click of Delete button
fncDeleteRecord(Id)
deleteVehicleRecord(Id)
Success() / updated list
return()/ getListVehicleList()
failure() --> An Error occured please try again

Figure 5.8:UC-10

Functional Requirements and Use case Mappings:

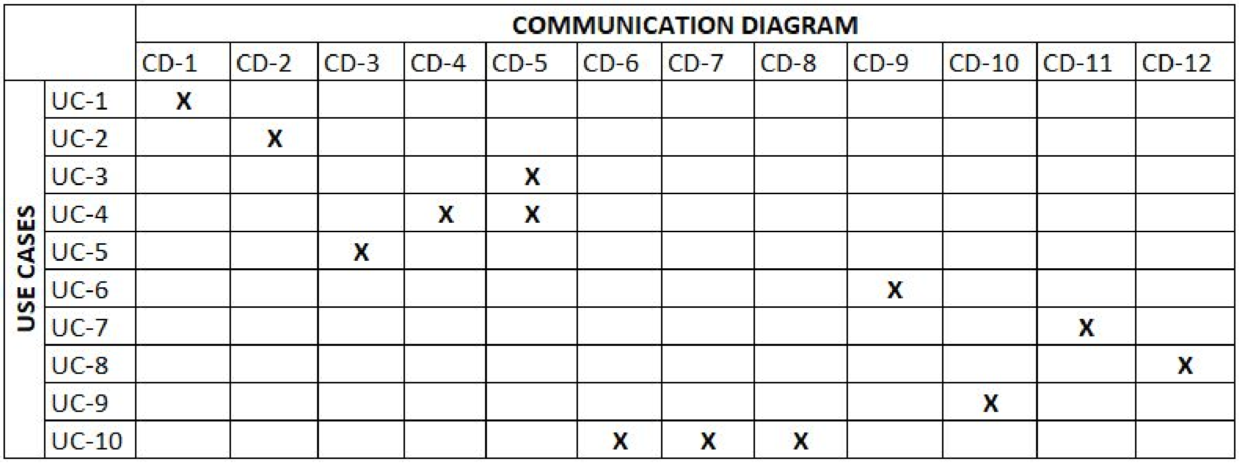
		USE CASES									
		UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10
FUNCTIONAL REQUIREMENTS	REQ-001	X									
	REQ-002		X								
	REQ-003			X							
	REQ-004				X						
	REQ-005				X						
	REQ-006					X					
	REQ-007					X					
	REQ-008							X			
	REQ-009							X			
	REQ-010							X			
	REQ-011							X			
	REQ-012						X				
	REQ-013						X				
	REQ-014										X
	REQ-015									X	
	REQ-016									X	



		USE CASES									
		UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10
NON- FUNCTIONAL REQUIREMENTS	REQ-017	X	X	X	X	X	X	X	X	X	X
	REQ-018	X	X	X	X	X	X	X	X	X	X
	REQ-019	X	X	X	X	X	X	X	X	X	X
	REQ-020	X	X	X	X	X	X	X	X	X	X
	REQ-021	X	X	X	X	X	X	X	X	X	X
	REQ-022	X	X								
	REQ-023						X	X		X	
	REQ-024						X	X		X	
	REQ-025						X		X	X	
	REQ-026						X		X	X	
	REQ-027			X	X		X				
	REQ-028			X	X		X				
	REQ-029	X	X								
	REQ-030	X	X								
	REQ-031			X	X	X	X	X	X	X	X

Figure 18: Requirements and Use Case mapping

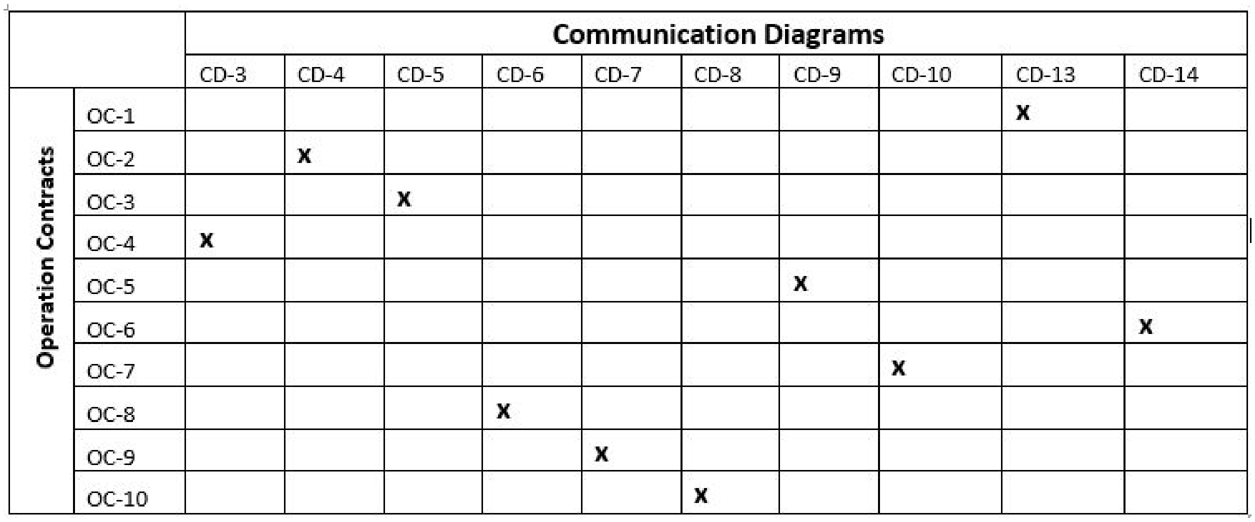
Use case and Communication diagram mapping:



		COMMUNICATION DIAGRAM											
		CD-1	CD-2	CD-3	CD-4	CD-5	CD-6	CD-7	CD-8	CD-9	CD-10	CD-11	CD-12
USE CASES	UC-1	X											
	UC-2		X										
	UC-3					X							
	UC-4				X	X							
	UC-5			X									
	UC-6									X			
	UC-7											X	
	UC-8												X
	UC-9										X		
	UC-10						X	X	X				

Figure 19: Use Case and Communication diagram mapping

Operation Contracts and Communication diagram mapping:



		Communication Diagrams									
		CD-3	CD-4	CD-5	CD-6	CD-7	CD-8	CD-9	CD-10	CD-13	CD-14
Operation Contracts	OC-1									X	
	OC-2		X								
	OC-3			X							
	OC-4	X									
	OC-5							X			
	OC-6										X
	OC-7								X		
	OC-8				X						
	OC-9					X					
	OC-10						X				

Figure 20: Operation Contracts and Communication diagram mapping

Development (Implementation) view

The development (or implementation) view describes the components used to assemble the system. Use a *UML component diagram* to capture this view. The Development of project is done in Model View Controller Design pattern which consists of how each of them talk to each other with whole functionality which are as follows :

1. Model

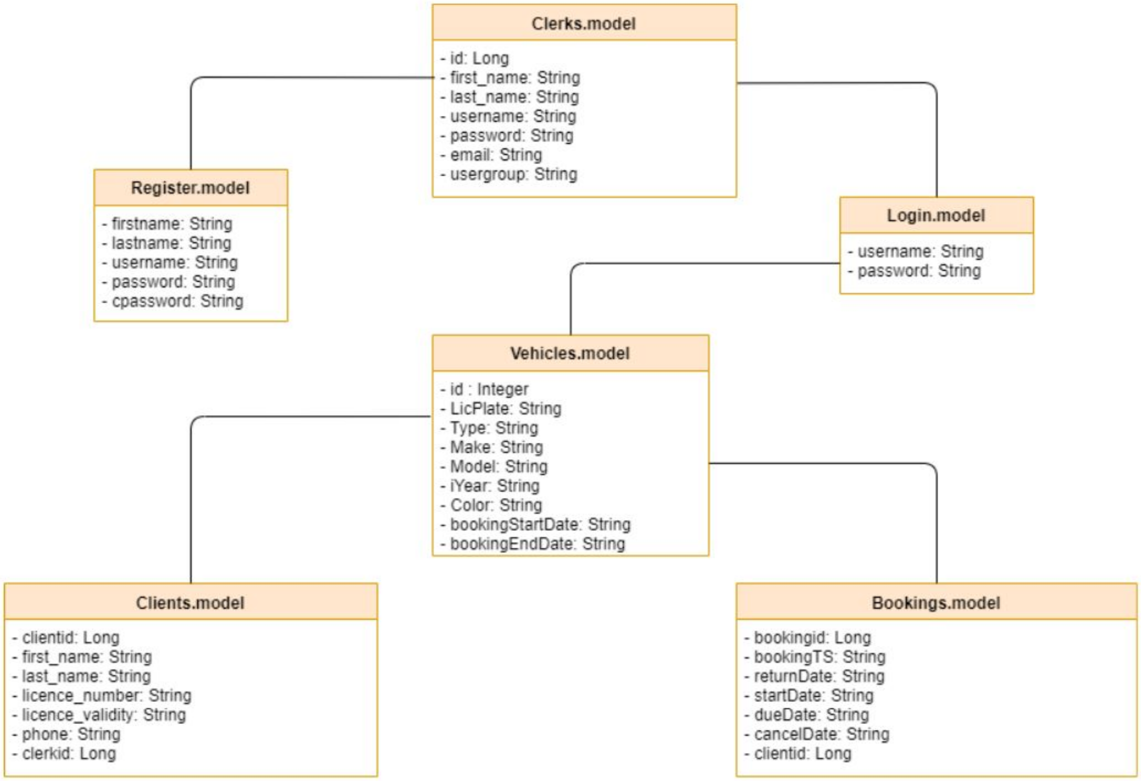


Figure:5.9

2. View : How all views are connected to each other are shown below in the snapshot in VRMS.

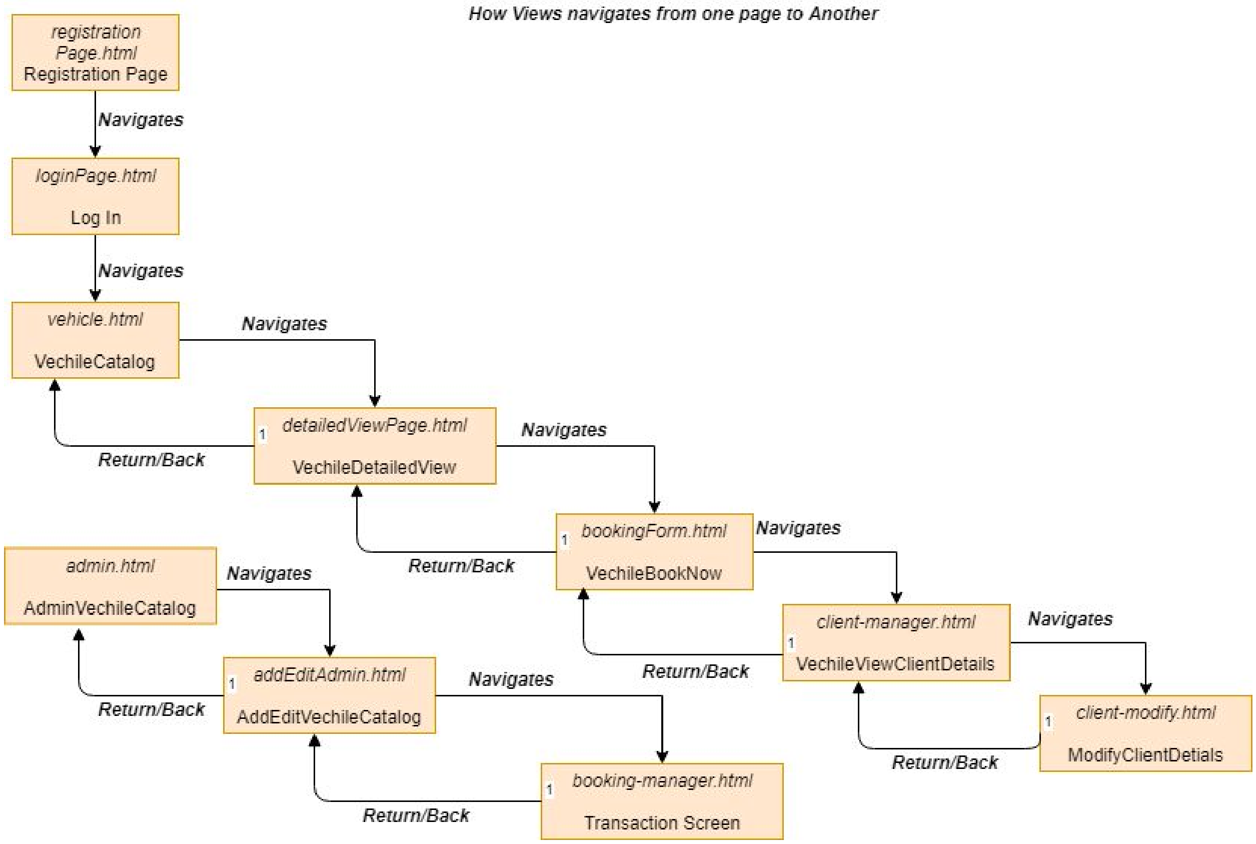
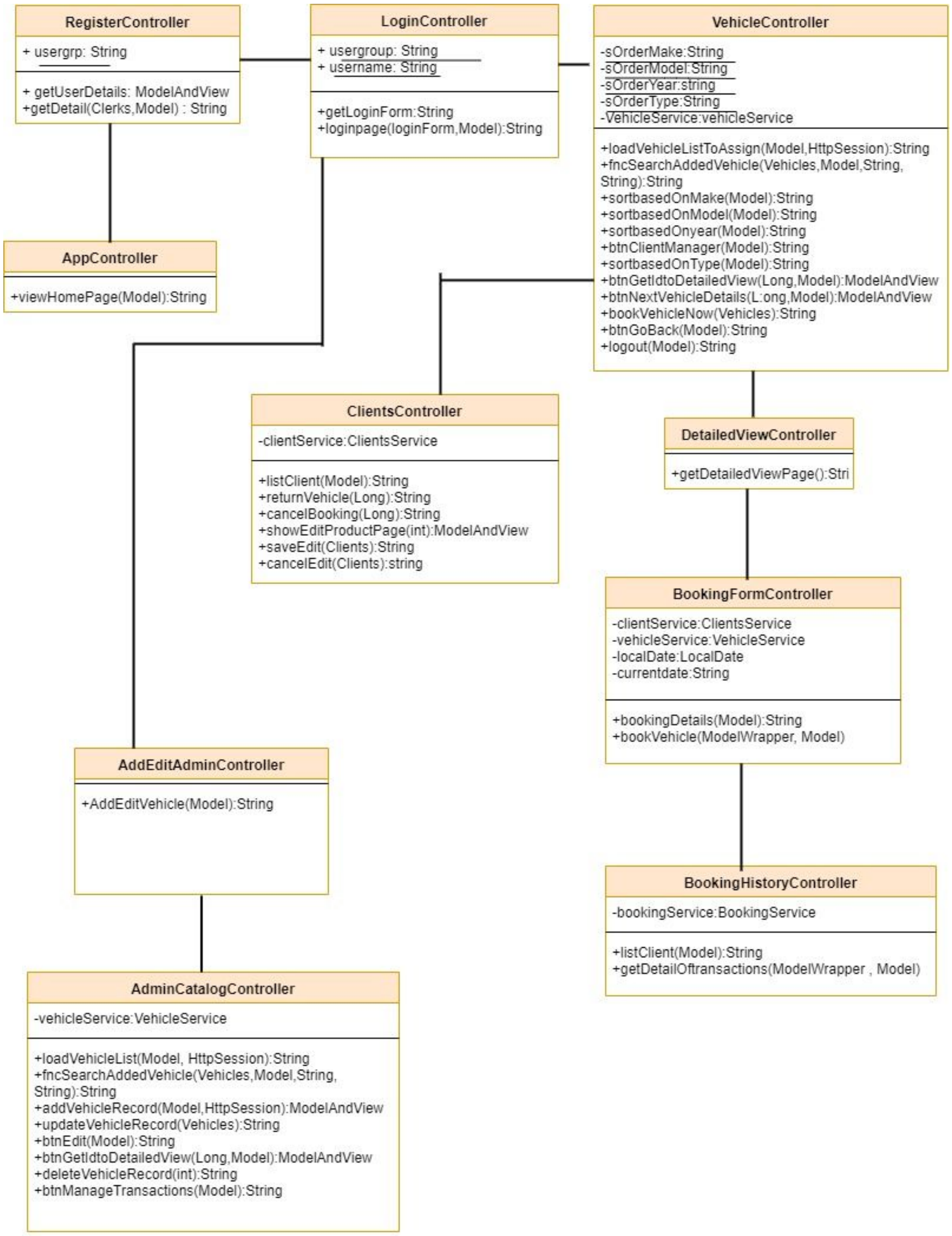


Figure:6.0

3. Controller



RegisterController
+ usergrp: String
+ getUserDetails: ModelAndView
+getDetail(Clerks,Model) : String
LoginController
+ usergroup: String
+ username: String
+getLoginForm:String
+loginpage(loginForm,Model):String
VehicleController
-sOrderMake:String
-sOrderModel:String
-sOrderYear:string
-sOrderType:String
-VehicleService:vehicleService
+loadVehicleListToAssign(Model,HttpSession):String
+fncSearchAddedVehicle(Vehicles,Model,String,
String):String
+sortbasedOnMake(Model):String
+sortbasedOnModel(Model):String
+sortbasedOnyear(Model):String
+btnClientManager(Model):String
+sortbasedOnType(Model):String
+btnGetIdtoDetailedView(Long,Model):ModelAndView
+btnNextVehicleDetails(L:ong,Model):ModelAndView
+bookVehicleNow(Vehicles):String
+btnGoBack(Model):String
+logout(Model):String
AppController
+viewHomePage(Model):String
ClientsController
-clientService:ClientsService
+listClient(Model):String
+returnVehicle(Long):String
+cancelBooking(Long):String
+showEditProductPage(int):ModelAndView
+saveEdit(Clients):String
+cancelEdit(Clients):string
DetailedViewController
+getDetailedViewPage():Stri
BookingFormController
-clientService:ClientsService
-vehicleService:VehicleService
-localDate:LocalDate
-currentdate:String
+bookingDetails(Model):String
+bookVehicle(ModelWrapper, Model)
AddEditAdminController
+AddEditVehicle(Model):String
BookingHistoryController
-bookingService:BookingService
+listClient(Model):String
+getDetailOftransactions(ModelWrapper , Model)
AdminCatalogController
-vehicleService:VehicleService
+loadVehicleList(Model, HttpSession):String
+fncSearchAddedVehicle(Vehicles,Model,String,
String):String
+addVehicleRecord(Model,HttpSession):ModelAndView
+updateVehicleRecord(Vehicles):String
+btnEdit(Model):String
+btnGetIdtoDetailedView(Long,Model):ModelAndView
+deleteVehicleRecord(int):String
+btnManageTransactions(Model):String

Controllers Class Diagram

Figure:6.1

4.Package Level Diagram:

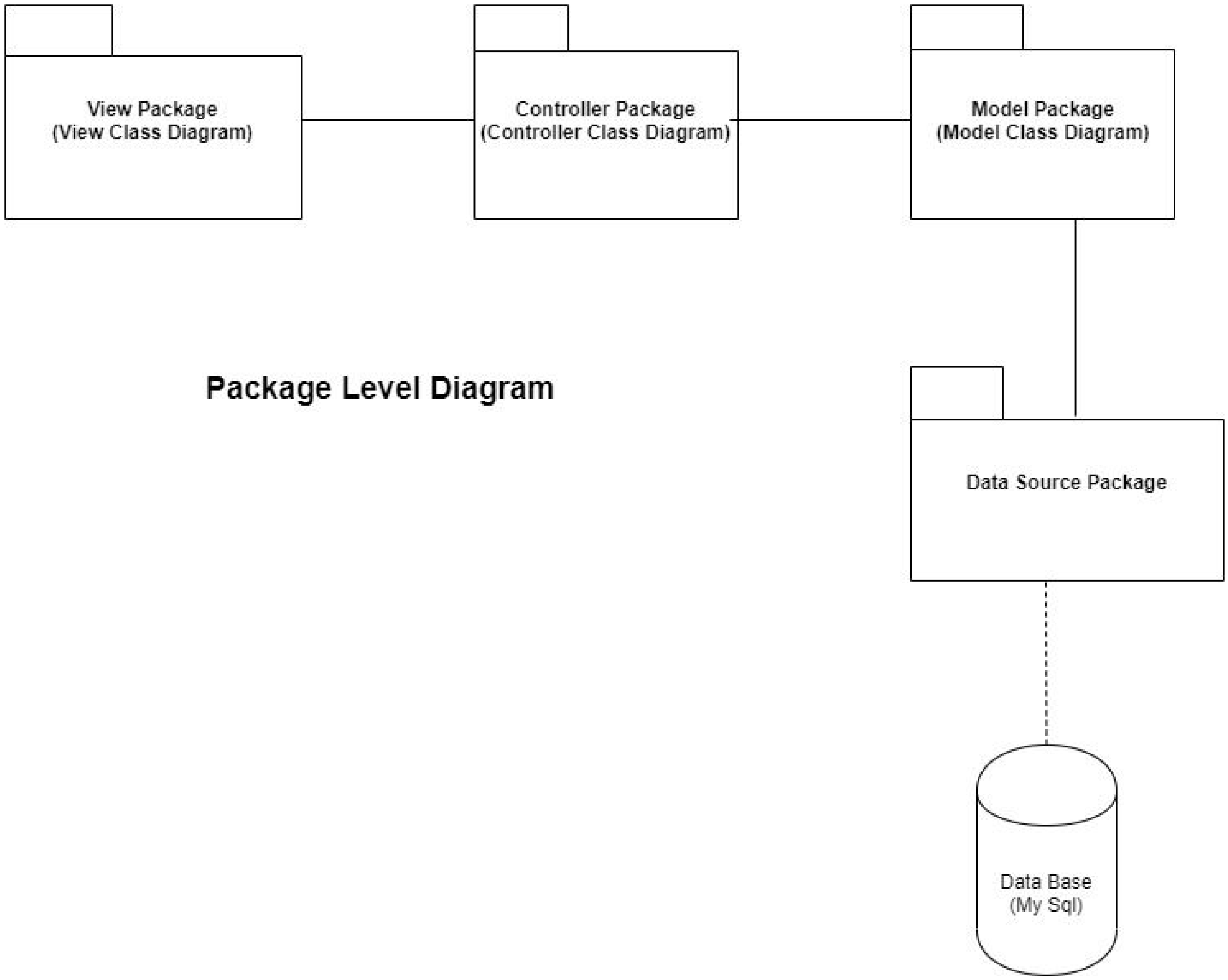


Figure:6.2

Reuse of components and frameworks

Describe any third-party or home-made components and frameworks that will be reused.

Data view (optional)

An enterprise software system would additionally require a data view. The data view describes the data entities and their relationships. Deploy an *Entity-Relationship* (ER) *Model* to represent this view. Note that the ER model is not part of the UML specification. Additionally you can deploy a UML class diagram to represent the data view where classes would correspond to data entities.

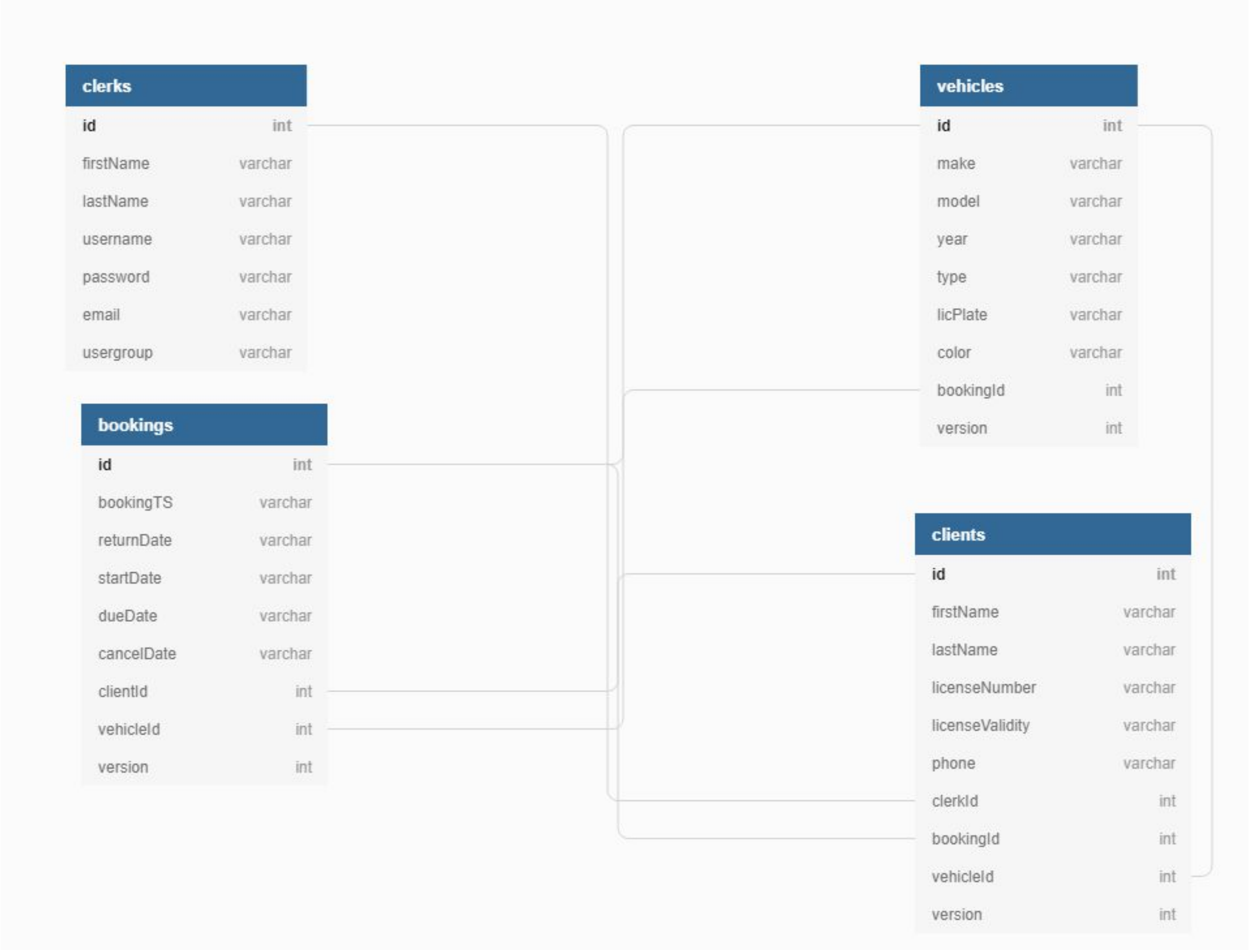


Figure:6.3

Quality

A description of how the software architecture contributes to the quality attributes of the system as described in the ISO-9126 (I) standard. **For example**: The following quality goals have been identified:

Scalability:

- Description : System's reaction when user demands increase
- Solution : J2EE application servers support several workload management techniques

Reliability, Availability:

- Description : Transparent failover mechanism, mean-time-between-failure
- Solution : : J2EE application server supports load balancing through clusters

Portability:

- Description : Ability to be reused in another environment
- Solution : The system me be fully J2EE compliant and thus can be deployed onto any J2EE application server

Security:

- Description : Authentication and authorization mechanisms
- Solution : J2EE native security mechanisms will be reused.