Software Architecture Document

Version 5.0

for

Vehicle Renting System

Prepared by

1.Bhavana Bomma	- 40071320 - bomma08@gmail.com
2.Divyaprabha Rajendran	- 40089282 - divyanajma@gmail.com
3.Harish Jayasankar	- 40105791 - harishjayasankar28@gmail.com
4.Ishpreet Singh	- 40093666 - ishpreetsingh923@gmail.com
5.Mahy Salama	- 40074737 - mahyhanysalama@gmail.com
6.Swetha Chenna	- 40092019 - swethachenna2018@gmail.com [Team Leader]

Instructor:	Dr. C. Constantinides
Course:	SOEN-6461
Date:	17/11/2019

Document history

Date	Version	Description	Author
17/11/2019	V.5	Updating document	all team members
17/11/2019	V.5	Updating sequence diagram	Swetha Chenna
06/10/2019	V2	missing parts and format	all team members
05/10/2019	V1	Use Case realization and sequence diagrams	all team members
10/27/2019	V3	Adding admin functionality related diagrams and reformatting class diagram	all team members
31/10/2019	V4	Updated team leader information	Swetha Chenna

Table of contents

1. Introduction 4
2. Architectural representation 5
7. Architectural requirements: goals and constraints 7
8. Use case view (Scenarios) 8
9. Logical view 8
10. Development (Implementation) view 10
11. Process view 10
12. Deployment (Physical) view 10
13. Data view (optional) 10
14. Quality 11
15. Design patterns table 32

List of figures

Figure 1: The 4+1 view model. 4

1. Introduction

Vehicle Rental System is a web application which includes reserving ,renting and returning vehicles. The Software Architecture Document is requires while developing the web application, the significant decisions are captured here in the Software Architecture Document, it is the map of the software. We use it to see and understand the software's modules and components before digging into the code.

Purpose

The Software Architecture Document provides a comprehensive architectural overview of the system, using a different architectural views to show different aspects of the system, the 4+1 view model that contains logical view for the designers to design a system that will provide the proper functionality for the end user, development view for programmers that basically deals with source code, executable files and component diagram for the implementation, process view for integrators that focus on how the system components communicate during the run time, physical view for deployment managers that considered with the physical connections between these components, and use case views the audiences are all the stakeholders of the system including the end users that illustrates the different scenarios and paths for the user to interact with the system.

Scope

The Software Architecture Document applies to all stakeholders that meant to design and implement the functionality of the clerk and administrator in our vehicle rental system where architecture of each functionality will be specified.

Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Architecture Document which will be used in complete project

project's Glossary

Abbreviation	Definition
SAD	Software Architecture Document, is a set of practices, techniques and types of representations used by software architects to record a software

	architecture. It is a modeling activity (Software architectural model). Architecture models can take various forms, including text, informal drawings, diagrams or other formalisms (UML).
UML	Unified Modeling Language, is a standardized modeling language enables developers to specify, visualize, construct and document the artifacts of a software system.
UC	Use Case, is a list of actions or event steps typically defining the interactions between a role (an actor in UML) and a system to achieve a goal. The actor can be a human, machine or other external system.

2. Architectural representation

Describe the top-level architectural style of the system and the view model you will adopt. Additionally describe what each individual view will provide. Many enterprise software systems are modeled using the 4+1 view illustrated in Figure 1.

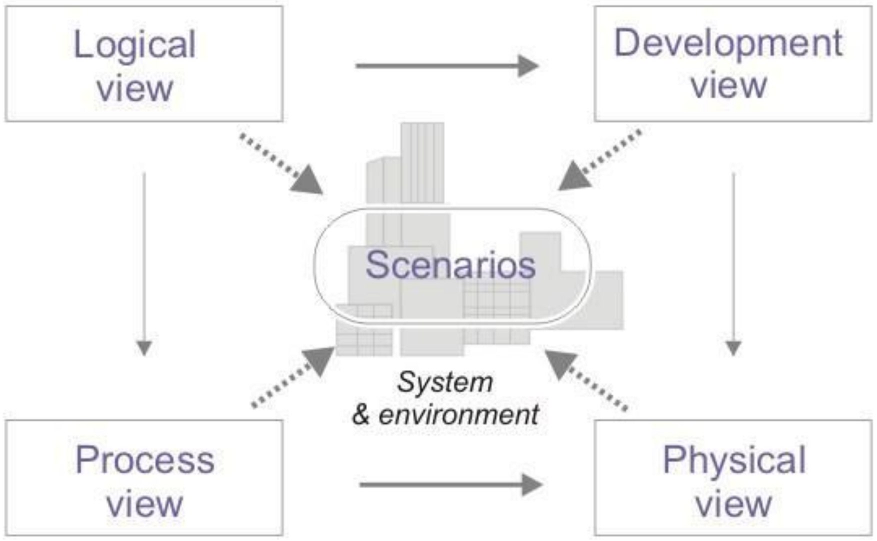


Figure 1: The 4+1 view model.

1. **Logical view** :It supports and provides clarification for the functional requirements, what services the system should provide to its end users, in this phase we try to get better knowledge of problem by talking to domain experts and other stakeholders, whatever decision taken here are independent of implementation decision. We elucidate logical view using Class diagram.

2. **Development view** :The development view focuses on organization of software modules in actual software. the software is organised as layers, each layer providing service to layer above it. we allocate effort required for development and maintenance in this view. it is represented using package diagram.
3. **Process view** : It is a dynamic view and it focuses on what will happen during run-time and also on non-functional requirements like performance. It addresses issues of concurrency and distribution, fault tolerance, process synchronization etc. It is explained using activity diagram.

4. **Physical view**:It focus on mapping software on to existing hardware.All Non functional requirements are taken care off in this view like performance, scalability etc. Its represented using deployment diagram.
5. **Use case view** (also known as Scenarios) : This view shows the interactions between actors/objects and the processes. In this view, we use a set of use cases or scenarios to depict the description of the system architecture.
6. **Data view** (optional): Audience: Data specialists, Database administrators. Describes the architecturally significant persistent elements in the data model .
   Related Artifacts: **Data model**.
7. **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.

Functional requirements (Use case view)

Refer to Use Cases or Use Case scenarios which are relevant with respect to the software architecture. The Use Cases referred to should contain central functionality, many architectural elements or specific delicate parts of the architecture.

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

<u>Use cases realization</u>

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC1 in SRS	View Catalog	Clerk: Wants to view the vehicle CatLog	9.4.1 (Use case realization) UC1

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC2 in SRS	View Vehicle Details	Clerk: Wants to view the vehicle CatLog and view a particular vehicle detail	9.4.2 (Use case realization) UC2

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC3 in SRS	Sorting/Filtering	Clerk: Wants to search the desired vehicle based on Sorting and filtering criteria	9.4.3 (Use case realization) UC3

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC4 in SRS	create Client	Clerk: wants to create a new client record.	9.4.4 (Use case realization) UC4

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC5 in SRS	Edit Client	Clerk: Wants to edit the existing client record	9.4.5 (Use case realization) UC5

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC6 in SRS	Delete Client	Clerk: Wants to delete a client	9.4.6 (Use case realization) UC6

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC7 in SRS	makeReservatioOrRental	Clerk: Requested to create a reservation or rental	9.4.7 (Use case realization) UC7

Use case UC8 in SRS	Cancel/Return	Clerk: Requested to cancel a reservation/return a rental	9.4.8 (Use case realization) UC8

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC9 in SRS	view Transaction History	Administrator: Wants to view the vehicle transaction history	9.4.9 (Use case realization) UC9

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC10 in SRS	Transaction Search	Administrator: Should be able to search the transaction history as per client ,per vehicle,per due date	9.5.0 (Use case realization) UC10

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC11 in SRS	Create new vehicle record	Administrator: Should be able to create a new vehicle record with the necessary conditions.	9.5.1 (Use case realization) UC11

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC12 in SRS	Modify vehicle record	Administrator: Should be able to modify vehicle record with the necessary conditions.	9.5.2 (Use case realization) UC12

Source	**Name**	**Architectural relevance**	**Addressed in:**
Use case UC13 in SRS	Delete Vehicle	Administrator: Should	9.5.3 (Use case

		be able to delete vehicle using license plate number.	realization) UC13

Source	Name	Architectural relevance	Addressed in:
Use case UC14 in SRS	Catalog Search	Administrator: Should be able to search catalog for particular vehicle using license plate number.	9.5.4 (Use case realization) UC14

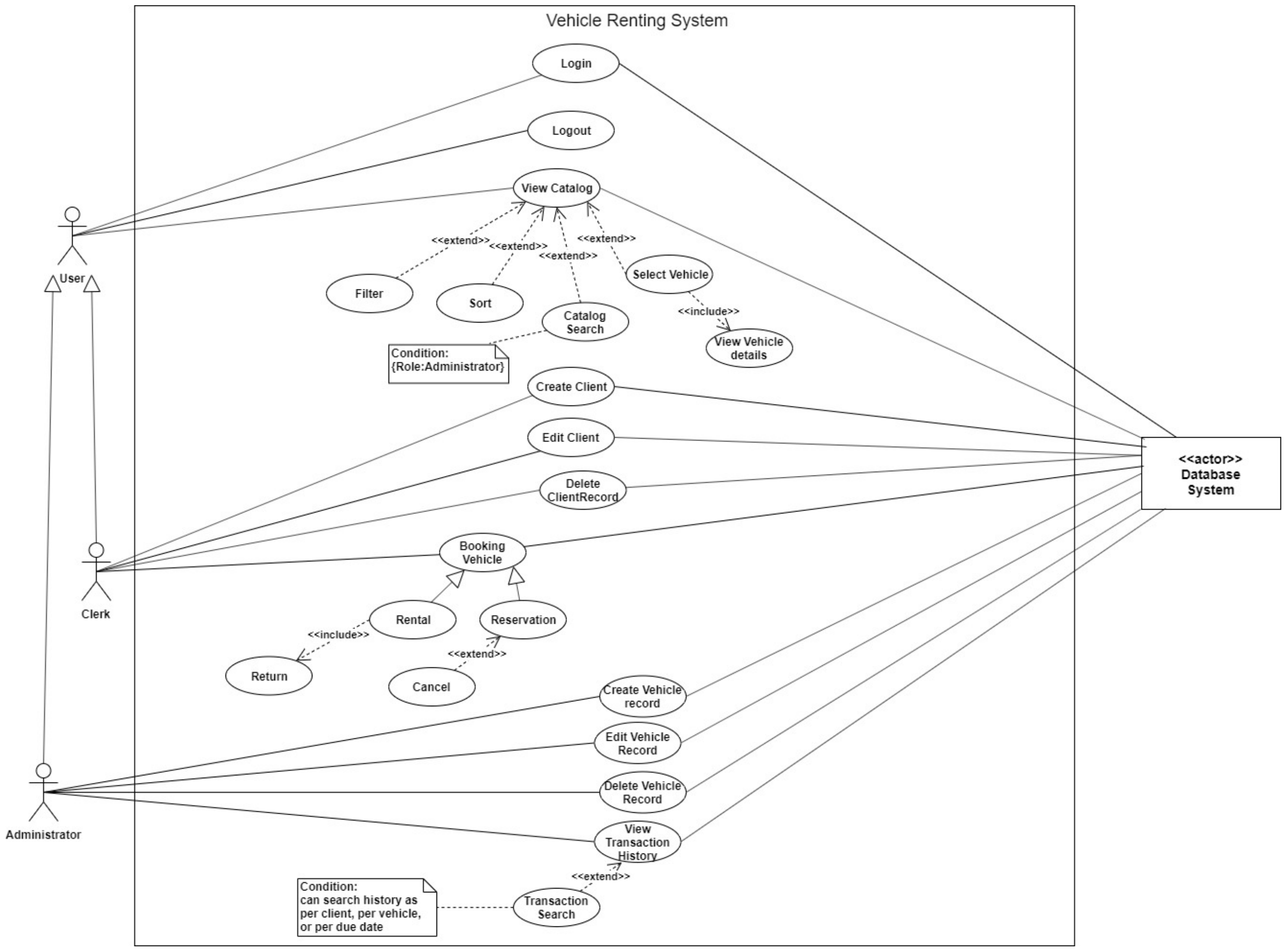
Non-functional requirements

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.

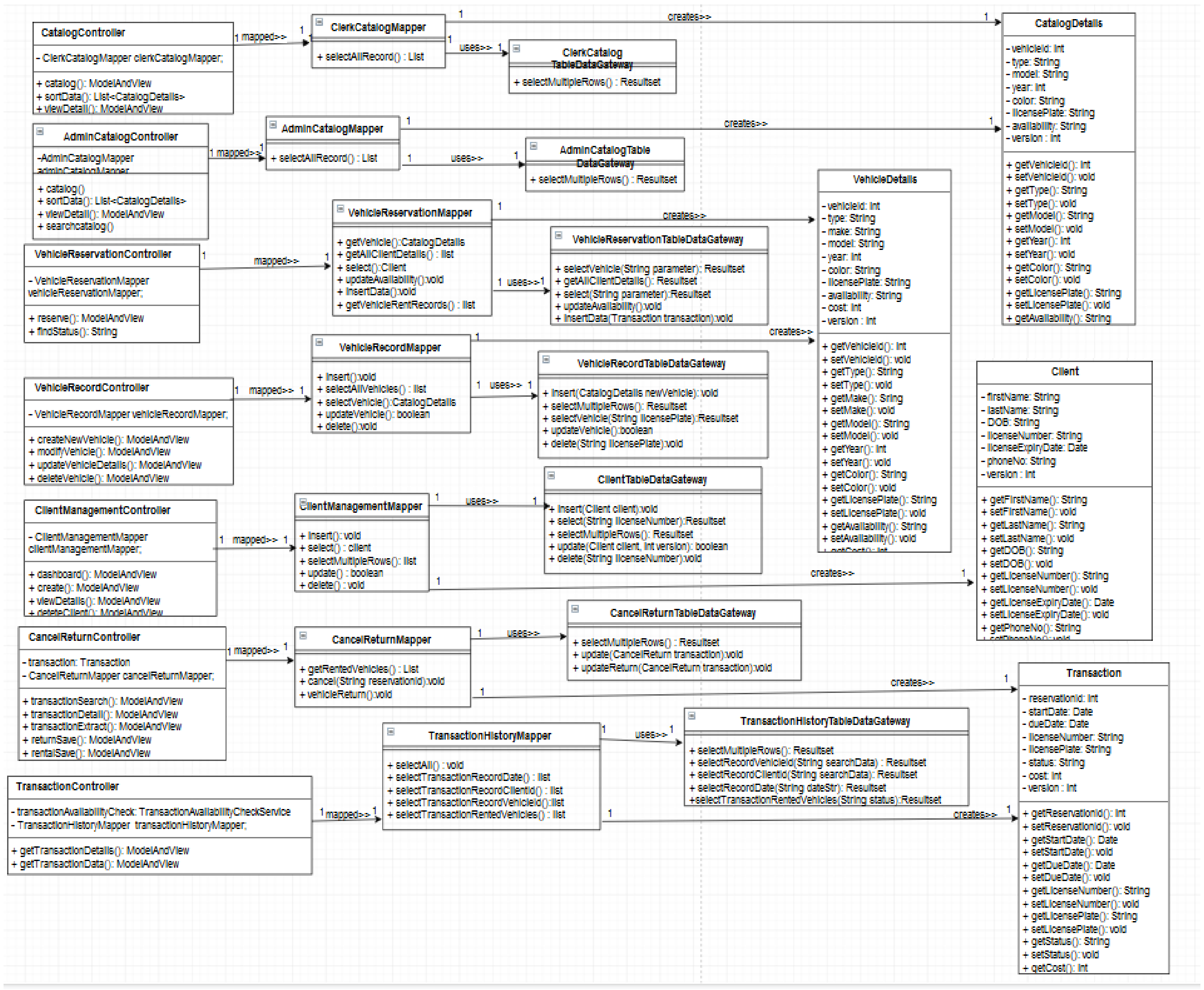
Usually, the non-functional requirements are already in place and can be referenced here. This document is not meant to be the source of non-functional requirements, but to address them. Provide a reference per requirement, and where the requirement is addressed.

8. 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.

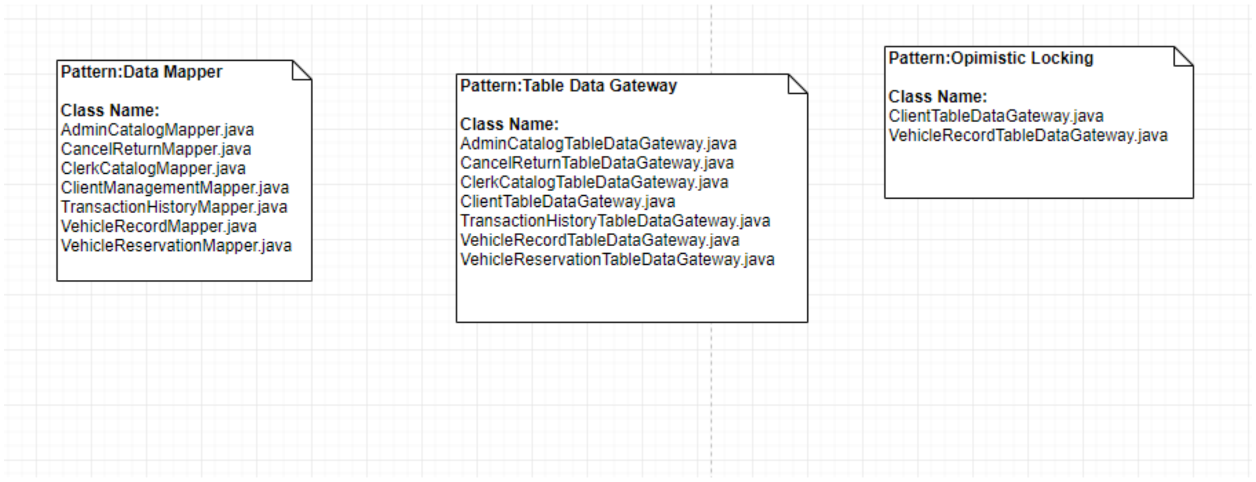


9. Logical view



CatalogController
- ClerkCatalogMapper clerkCatalogMapper;
+ catalog(): ModelAndView
+ sortData(): List<CatalogDetails>
+ viewDetail(): ModelAndView
mapped>>
ClerkCatalogMapper
+ selectAllRecord() : List
uses>>
ClerkCatalog TableDataGateway
+ selectMultipleRows() : Resultset
creates>>
CatalogDetails
- vehicleId: int
- type: String
- model: String
- year: int
- color: String
- licensePlate: String
- availability: String
- version : int
+ getVehicleId(): int
+ setVehicleId(): void
+ getType(): String
+ setType(): void
+ getModel(): String
+ setModel(): void
+ getYear(): int
+ setYear(): void
+ getColor(): String
+ setColor(): void
+ getLicensePlate(): String
+ setLicensePlate(): void
+ getAvailability(): String
AdminCatalogController
-AdminCatalogMapper adminCatalogMapper
+ catalog()
+ sortData(): List<CatalogDetails>
+ viewDetail(): ModelAndView
+ searchcatalog()
AdminCatalogMapper
+ selectAllRecord() : List
AdminCatalogTable DataGateway
+ selectMultipleRows() : Resultset
VehicleDetails
- vehicleId: int
- type: String
- make: String
- model: String
- year: int
- color: String
- licensePlate: String
- availability: String
- cost: int
- version : int
+ getVehicleId(): int
+ setVehicleId(): void
+ getType(): String
+ setType(): void
+ getMake(): Sring
+ setMake(): void
+ getModel(): String
+ setModel(): void
+ getYear(): int
+ setYear(): void
+ getColor(): String
+ setColor(): void
+ getLicensePlate(): String
+ setLicensePlate(): void
+ getAvailability(): String
+ setAvailability(): void
VehicleReservationController
- VehicleReservationMapper vehicleReservationMapper;
+ reserve(): ModelAndView
+ findStatus(): String
VehicleReservationMapper
+ getVehicle():CatalogDetails
+ getAllClientDetails() : list
+ select():Client
+ updateAvailability():void
+ insertData():void
+ getVehicleRentRecords() : list
VehicleReservationTableDataGateway
+ selectVehicle(String parameter): Resultset
+ getAllClientDetails(): Resultset
+ select(String parameter):Resultset
+ updateAvailability():void
+ insertData(Transaction transaction):void
VehicleRecordController
- VehicleRecordMapper vehicleRecordMapper;
+ createNewVehicle(): ModelAndView
+ modifyVehicle(): ModelAndView
+ updateVehicleDetails(): ModelAndView
+ deleteVehicle(): ModelAndView
VehicleRecordMapper
+ insert():void
+ selectAllVehicles() : list
+ selectVehicle():CatalogDetails
+ updateVehicle(): boolean
+ delete():void
VehicleRecordTableDataGateway
+ insert(CatalogDetails newVehicle): void
+ selectMultipleRows(): Resultset
+ selectVehicle(String licensePlate):Resultset
+ updateVehicle():boolean
+ delete(String licensePlate):void
Client
- firstName: String
- lastName: String
- DOB: String
- licenseNumber: String
- licenseExpiryDate: Date
- phoneNo: String
- version : int
+ getFirstName(): String
+ setFirstName(): void
+ getLastName(): String
+ setLastName(): void
+ getDOB(): String
+ setDOB(): void
+ getLicenseNumber(): String
+ setLicenseNumber(): void
+ getLicenseExpiryDate(): Date
+ setLicenseExpiryDate(): void
+ getPhoneNo(): String
ClientManagementController
- ClientManagementMapper clientManagementMapper;
+ dashboard(): ModelAndView
+ create(): ModelAndView
+ viewDetails(): ModelAndView
+ deleteClient(): ModelAndView
ClientManagementMapper
+ insert(): void
+ select() : client
+ selectMultipleRows(): list
+ update() : boolean
+ delete() : void
ClientTableDataGateway
+ insert(Client client):void
+ select(String licenseNumber):Resultset
+ selectMultipleRows(): Resultset
+ update(Client client, int version): boolean
+ delete(String licenseNumber):void
CancelReturnController
- transaction: Transaction
- CancelReturnMapper cancelReturnMapper;
+ transactionSearch(): ModelAndView
+ transactionDetail(): ModelAndView
+ transactionExtract(): ModelAndView
+ returnSave(): ModelAndView
+ rentalSave(): ModelAndView
CancelReturnMapper
+ getRentedVehicles() : List
+ cancel(String reservationId):void
+ vehicleReturn():void
CancelReturnTableDataGateway
+ selectMultipleRows() : Resultset
+ update(CancelReturn transaction):void
+ updateReturn(CancelReturn transaction):void
Transaction
- reservationId: int
- startDate: Date
- dueDate: Date
- licenseNumber: String
- licensePlate: String
- status: String
- cost: int
- version : int
+ getReservationId(): int
+ setReservationId(): void
+ getStartDate(): Date
+ setStartDate(): void
+ getDueDate(): Date
+ setDueDate(): void
+ getLicenseNumber(): String
+ setLicenseNumber(): void
+ getLicensePlate(): String
+ setLicensePlate(): void
+ getStatus(): String
+ setStatus(): void
+ getCost(): int
TransactionController
- transactionAvailabilityCheck: TransactionAvailabilityCheckService
- TransactionHistoryMapper transactionHistoryMapper;
+ getTransactionDetails(): ModelAndView
+ getTransactionData(): ModelAndView
TransactionHistoryMapper
+ selectAll() : void
+ selectTransactionRecordDate() : list
+ selectTransactionRecordClientId() : list
+ selectTransactionRecordVehicleId():list
+ selectTransactionRentedVehicles() : list
TransactionHistoryTableDataGateway
+ selectMultipleRows(): Resultset
+ selectRecordVehicleId(String searchData) : Resultset
+ selectRecordClientId(String searchData): Resultset
+ selectRecordDate(String dateStr): Resultset
+selectTransactionRentedVehicles(String status):Resultset

Please find patterns implemented and respective class names below:



Pattern:Data Mapper
Class Name:
AdminCatalogMapper.java
CancelReturnMapper.java
ClerkCatalogMapper.java
ClientManagementMapper.java
TransactionHistoryMapper.java
VehicleRecordMapper.java
VehicleReservationMapper.java
Pattern:Table Data Gateway
Class Name:
AdminCatalogTableDataGateway.java
CancelReturnTableDataGateway.java
ClerkCatalogTableDataGateway.java
ClientTableDataGateway.java
TransactionHistoryTableDataGateway.java
VehicleRecordTableDataGateway.java
VehicleReservationTableDataGateway.java
Pattern:Opimistic Locking
Class Name:
ClientTableDataGateway.java
VehicleRecordTableDataGateway.java

9.1 Layers, tiers etc.

Describe the top-level architecture style. Deploy a *UML class diagram*.

9.2 Subsystems

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

9.3 Architecturally significant design packages

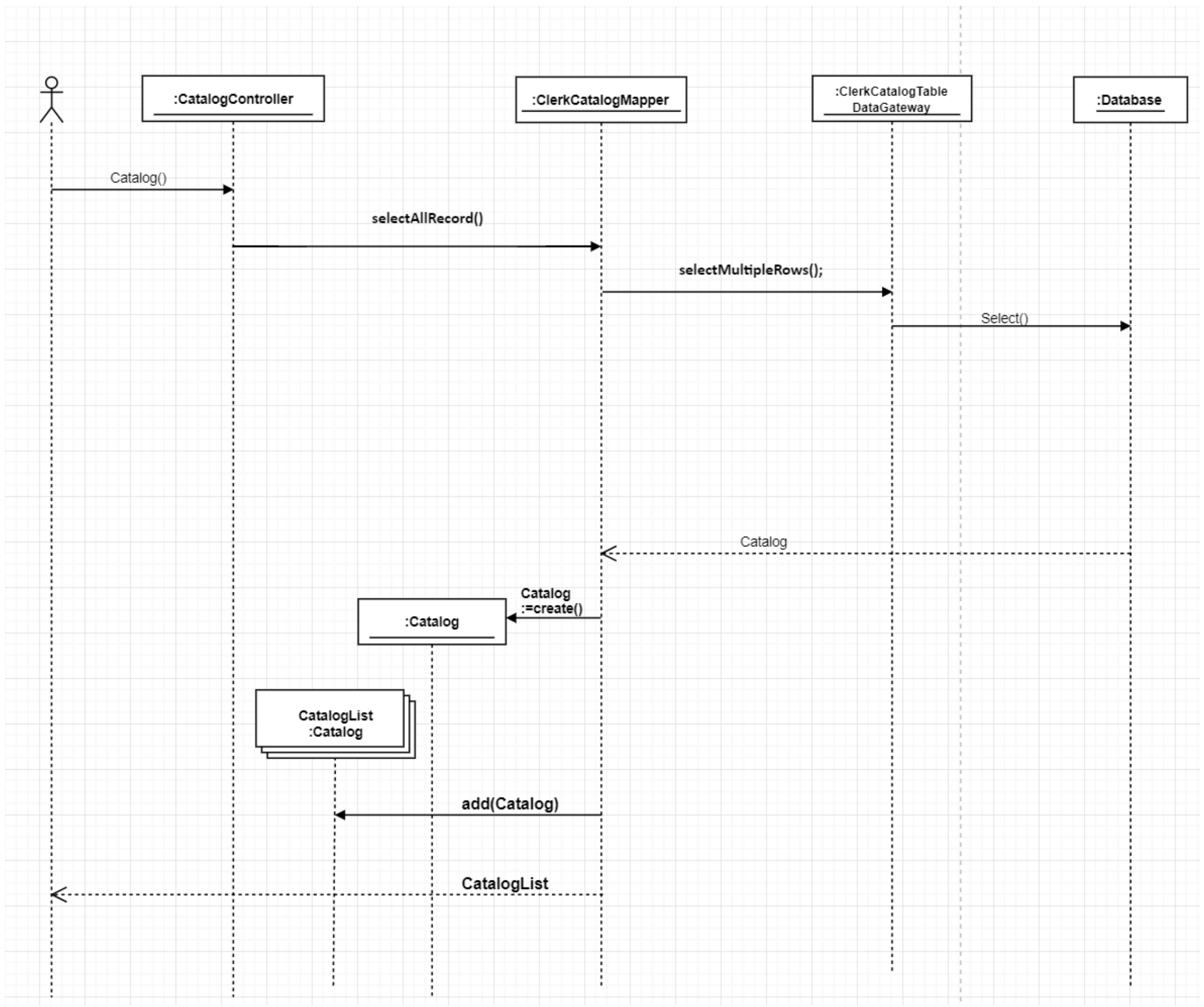
Describe packages of individual subsystems that are architecturally significant. For each package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

9.4 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.

9.4.1 UC1: 1. The instance of List<Catalog> is created.

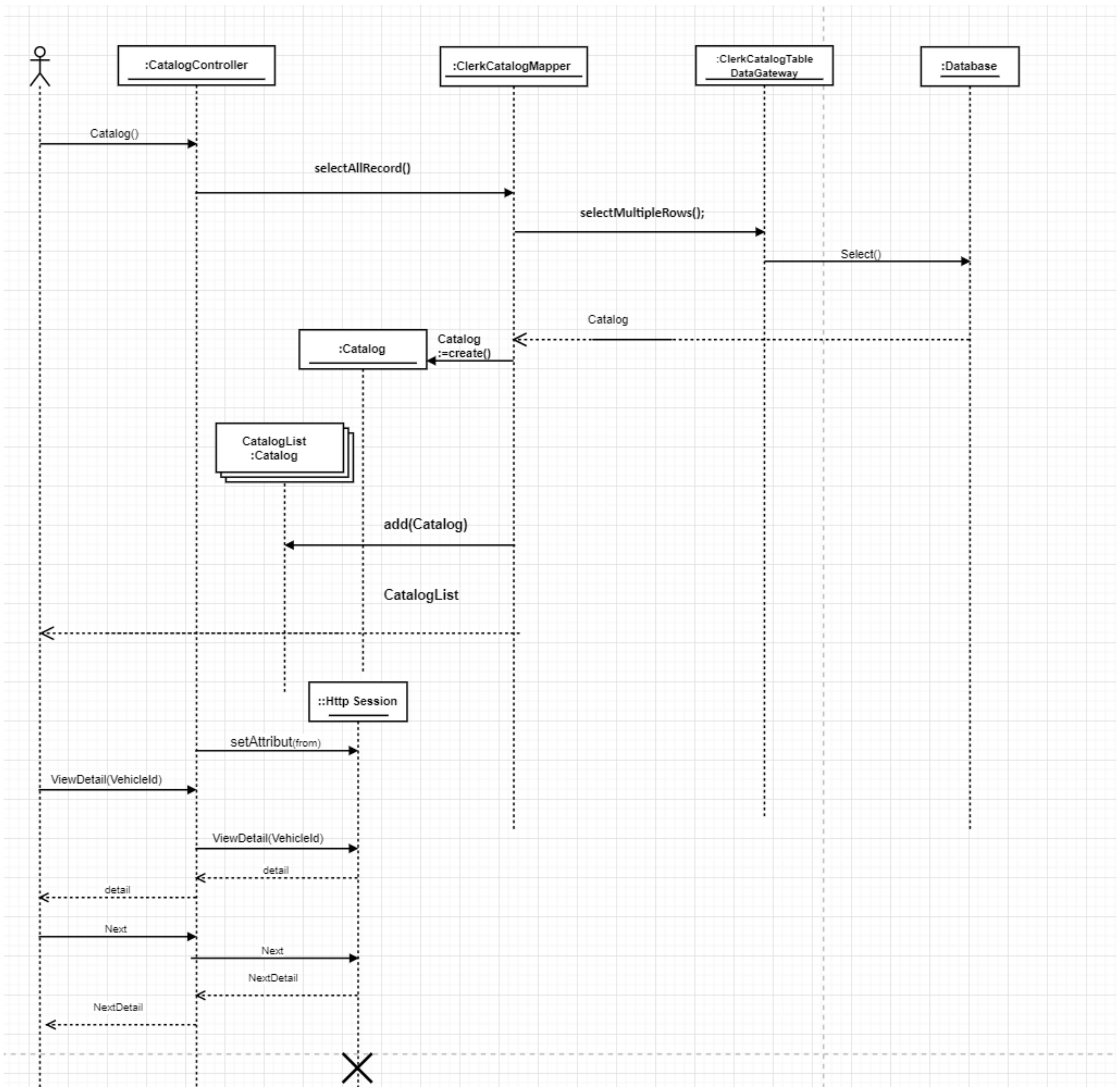
2. instance will be associated with CatalogMapper.



9.4.2 UC2:1. The instance of List<Catalog> instance is created.

2. instance will be associated with CatalogMapper.

3.instance was associated with httpsession.

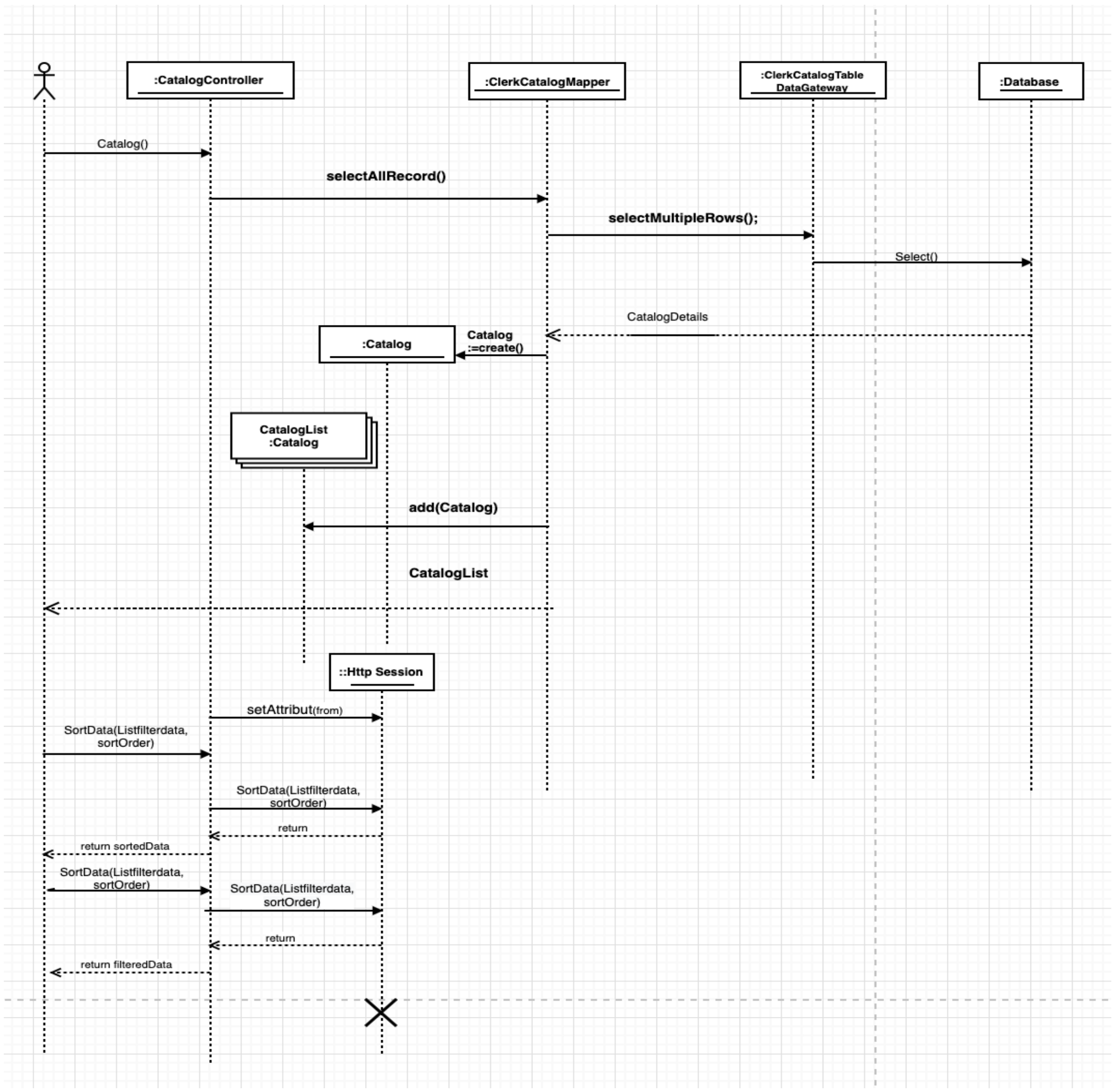


9.4.3 UC3: 1. Instance of vehicle will be created.

2.Instance will be associated with rowMapper.

3.Instance will be modified.

4.instance will be persisted in database

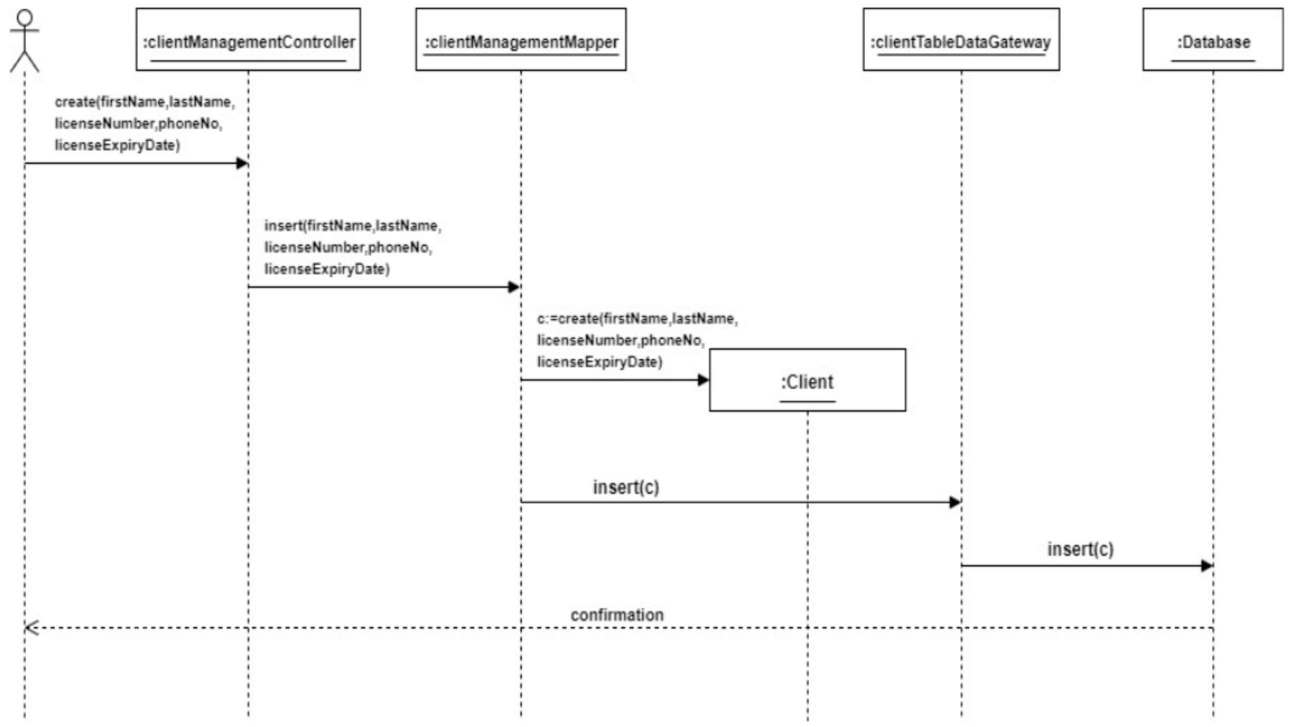


9.4.4 UC4: 1. The instance of client will be created.

2. The instance will be associated to clientManagementMapper

3.Then instance will be modified.

4.Instance will be persisted in database.

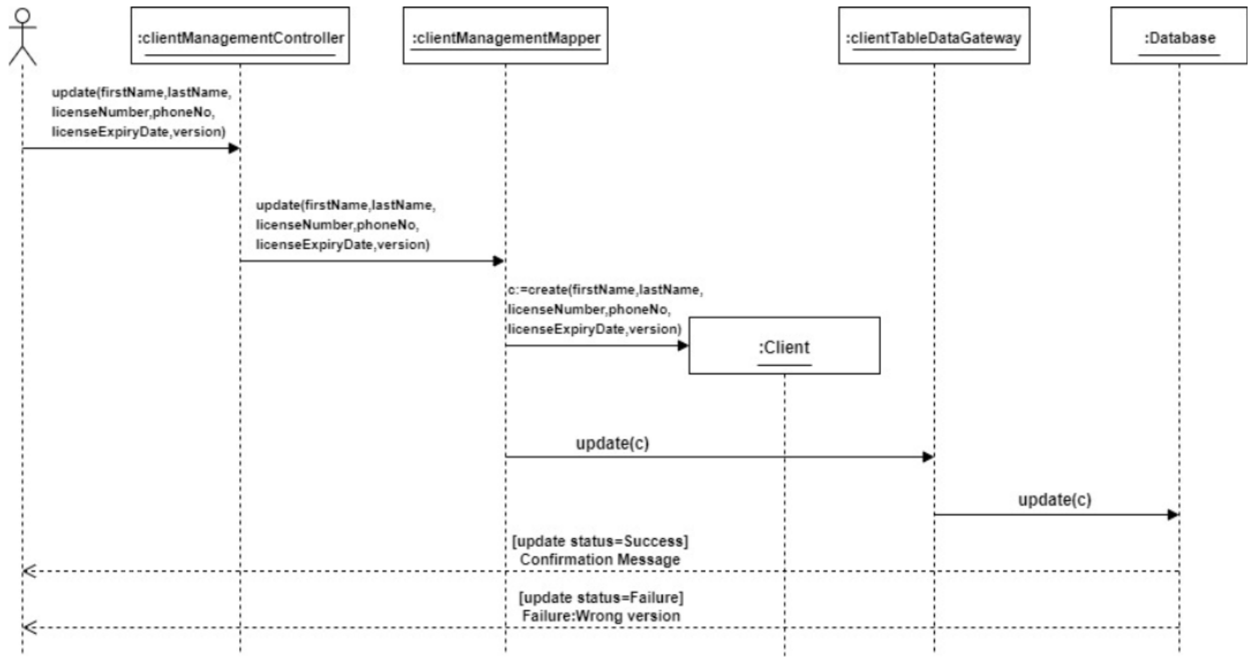

:clientManagementController
:clientManagementMapper
:clientTableDataGateway
:Database
create(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate)
insert(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate)
c:=create(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate)
:Client
insert(c)
insert(c)
confirmation

9.4.5 UC5: 1. The instance of client will be created.

2. The instance will be associated to clientManagementMapper

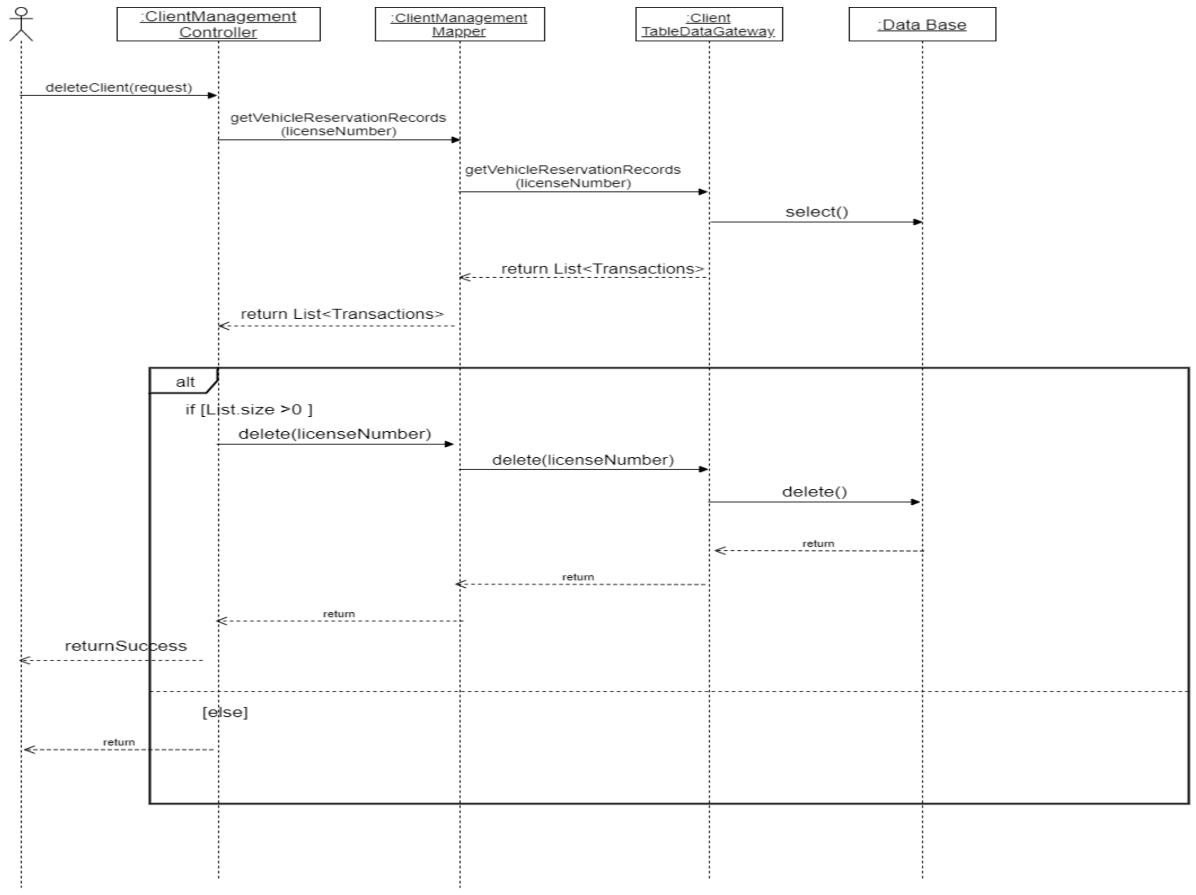
3.Then instance will be updated.

4.Instance will be persisted in database.


:clientManagementController
:clientManagementMapper
:clientTableDataGateway
:Database
update(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate,version)
update(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate,version)
c:=create(firstName,lastName, licenseNumber,phoneNo, licenseExpiryDate,version)
:Client
update(c)
update(c)
[update status=Success] Confirmation Message
[update status=Failure] Failure:Wrong version

9.4.6 UC6:

1. The instance of client model will be created.
2. The instance is associated with ClientManagementMapper
3. client record will be deleted from the database.



9.4.7 UC7: 1. The reservation instance will be created.

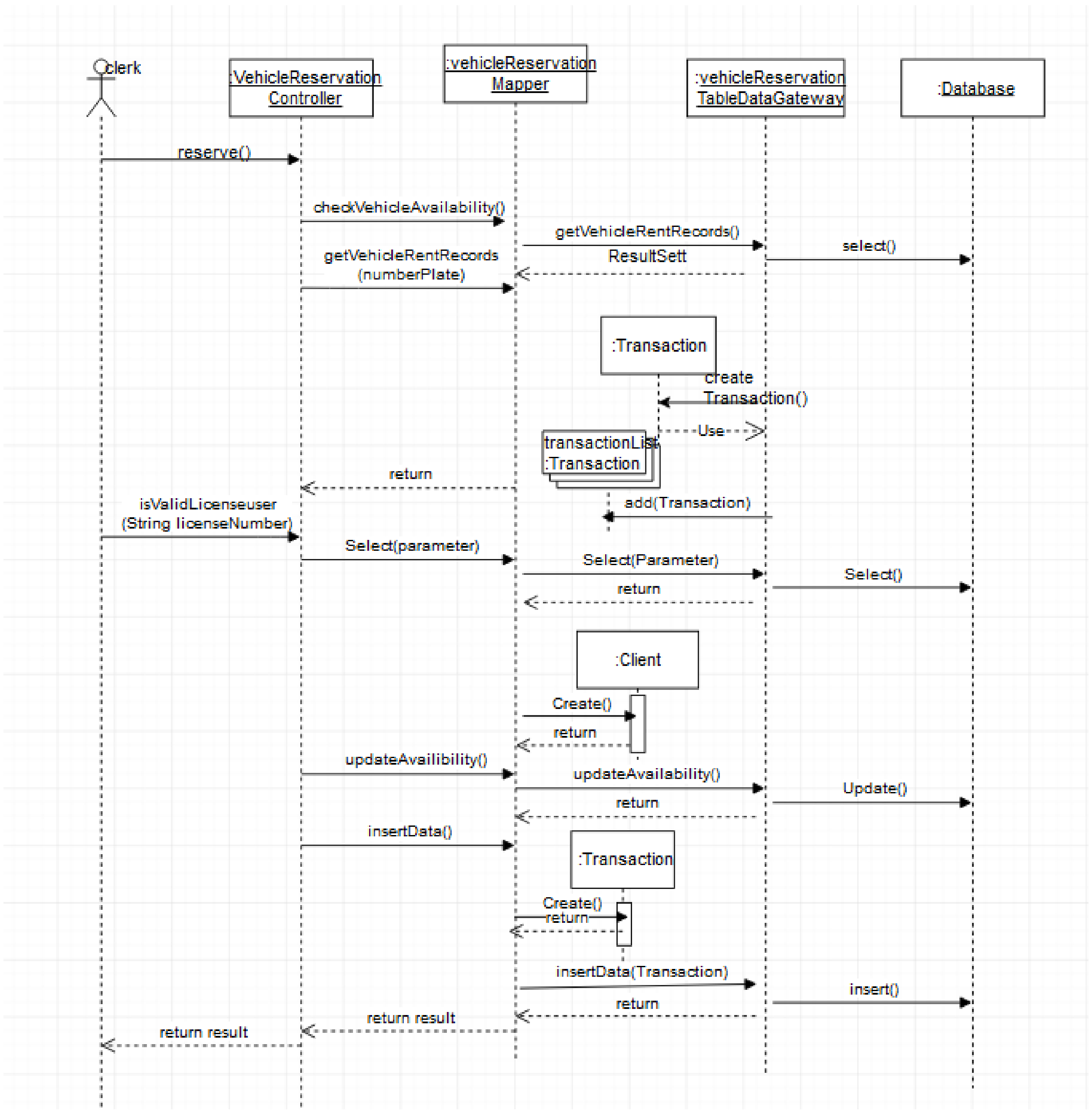
2 . Instance created is associated with VehicleReservationMapper.

3. The reservation instance is inserted to the database.

4. The related vehicle instance is fetched from database.

5. The status of the fetched vehicle must be modified to “NO”.

6. The modified instance is updated in the database.

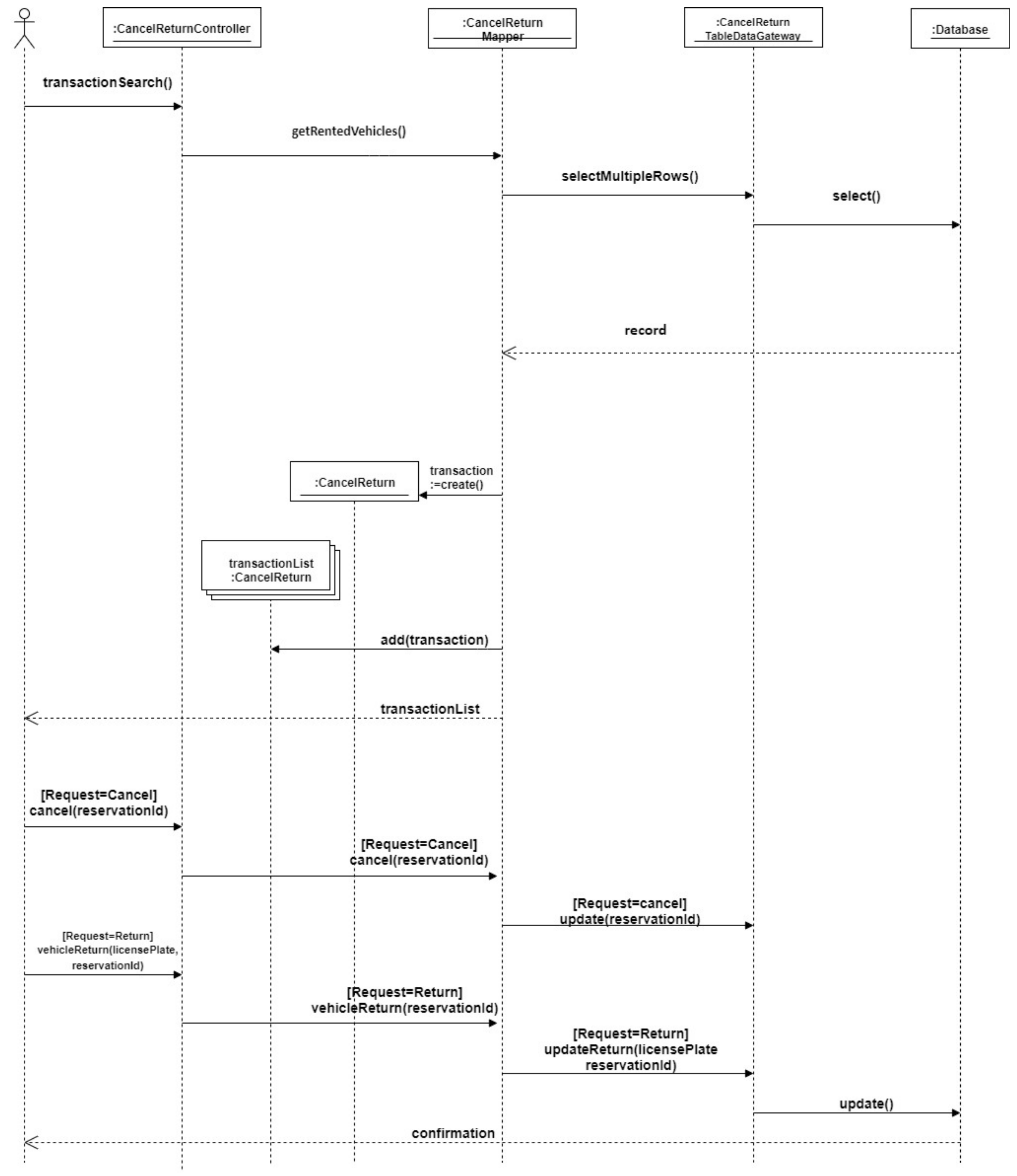


9.4.8 UC8: 1. Instance of cancelReturn model will be created.

2.Instance will be associated with rowMapper.

3.Instance will be modified.

4.instance will be persisted in database

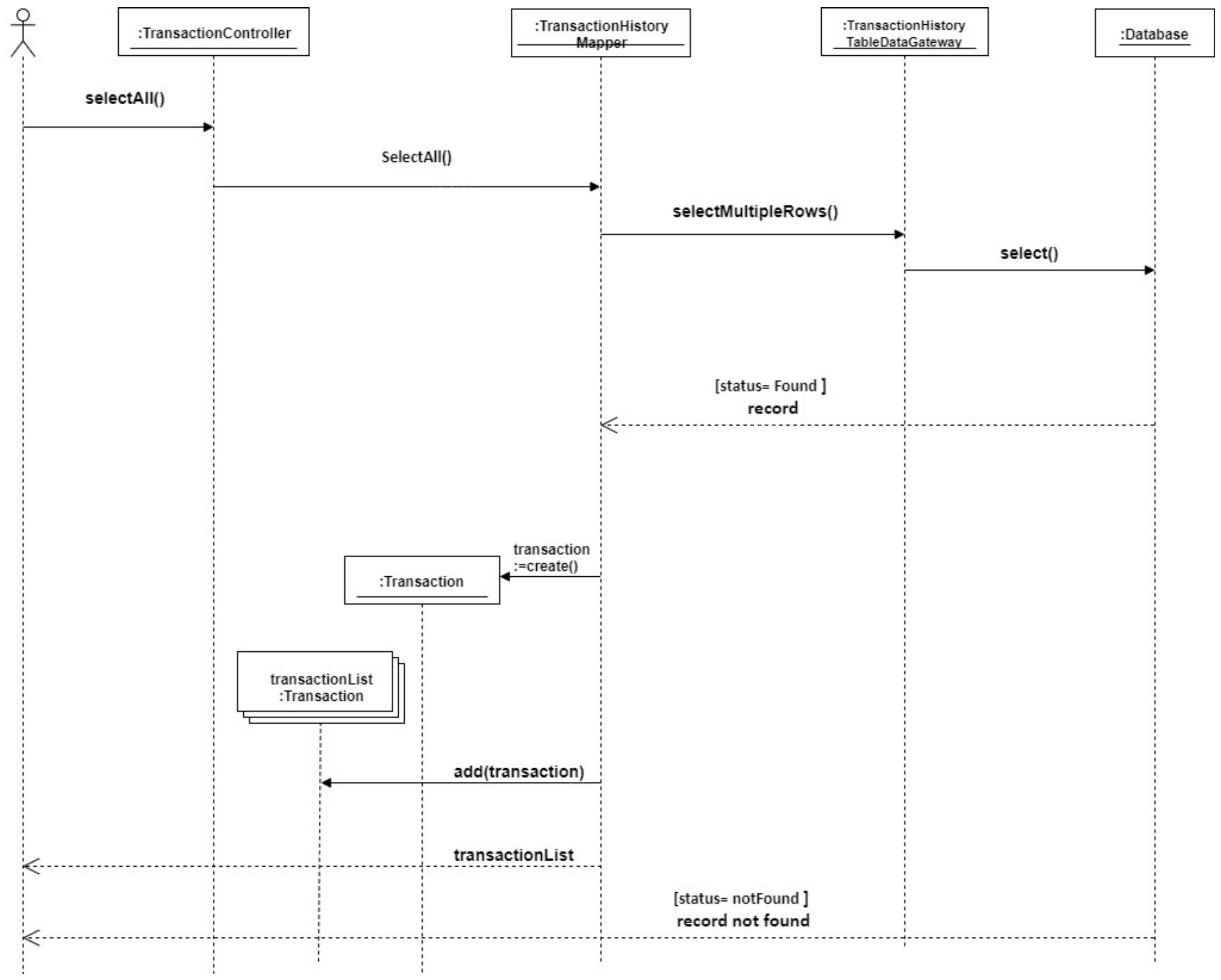

:CancelReturnController
:CancelReturn Mapper
:CancelReturn TableDataGateway
:Database
transactionSearch()
getRentedVehicles()
selectMultipleRows()
select()
record
:CancelReturn
transaction :=create()
transactionList :CancelReturn
add(transaction)
transactionList
[Request=Cancel] cancel(reservationId)
[Request=Cancel] cancel(reservationId)
[Request=cancel] update(reservationId)
[Request=Return] vehicleReturn(licensePlate, reservationId)
[Request=Return] vehicleReturn(reservationId)
[Request=Return] updateReturn(licensePlate reservationId)
update()
confirmation

9.4.9:- UC9:

1. The list of instances of Transactions will be created.

2.All the transaction history records will be fetched from the database.

3.The instances attributes will updated with data from database.



9.5.0 :- UC10:

1. The list of instances of Transactions will be created.

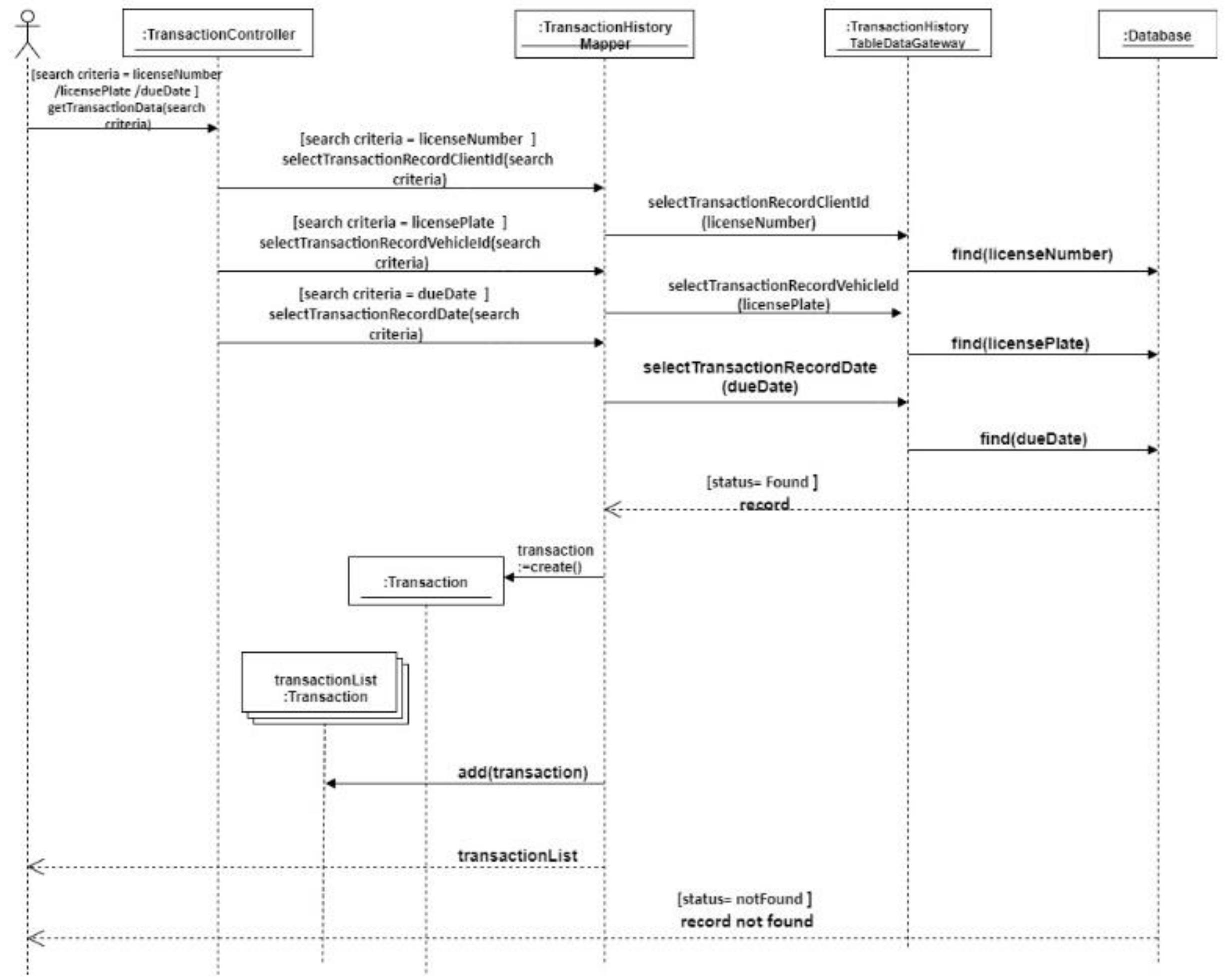
2.Then list of instances will be associated with transactionHistoryMapper.

3.All the transaction history records with respect to search criteria will be fetched from the database.

4.The instances attributes will updated with data from database.

Vehicle Renting System	Version: 5.0
Software Architecture Document	Date:17/11/2019

5.Then data will be displayed on the screen.

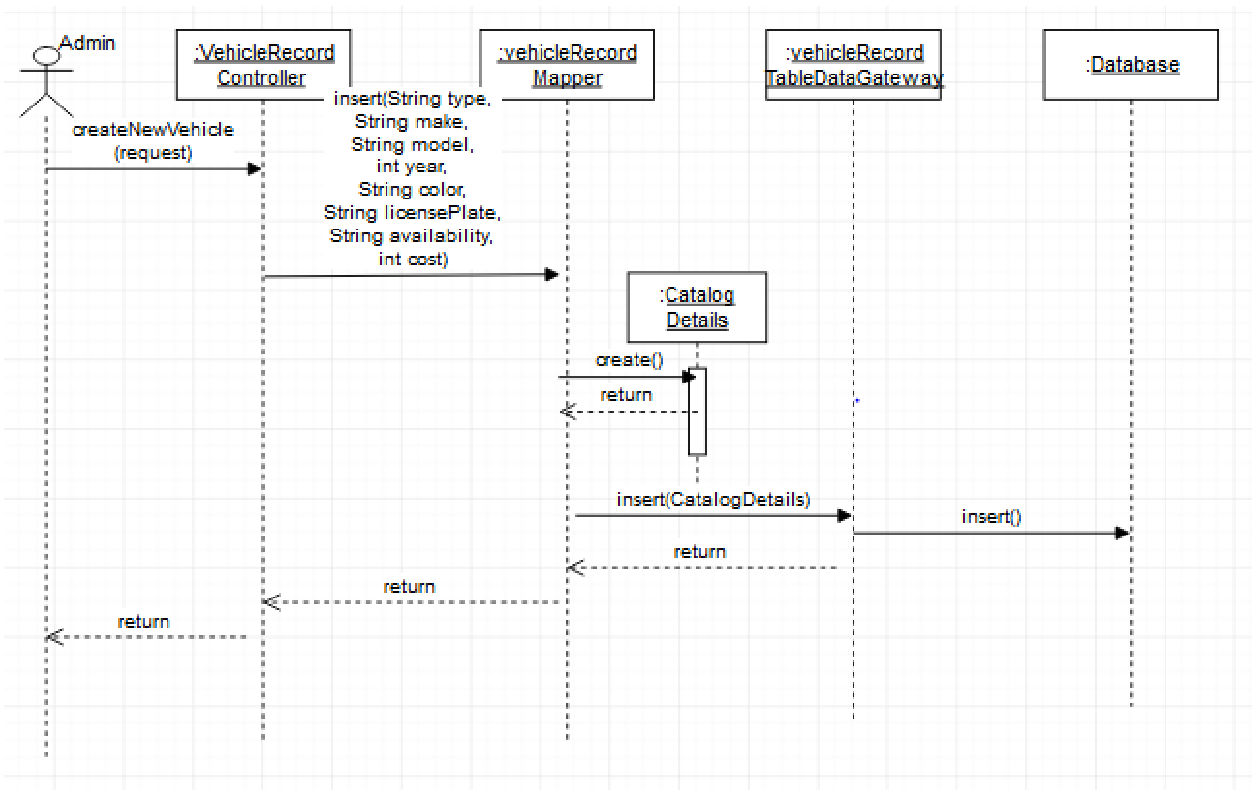

:TransactionController
:TransactionHistory Mapper
:TransactionHistory TableDataGateway
:Database
[search criteria = licenseNumber /licensePlate /dueDate] getTransactionData(search criteria)
[search criteria = licenseNumber] selectTransactionRecordClientId(search criteria)
selectTransactionRecordClientId (licenseNumber)
find(licenseNumber)
[search criteria = licensePlate] selectTransactionRecordVehicleId(search criteria)
selectTransactionRecordVehicleId (licensePlate)
find(licensePlate)
[search criteria = dueDate] selectTransactionRecordDate(search criteria)
selectTransactionRecordDate (dueDate)
find(dueDate)
[status= Found] record
transaction :=create()
:Transaction
transactionList :Transaction
add(transaction)
transactionList
[status= notFound] record not found

9.5.1 :- UC11:

1. Vehicle instance should be created.

2. It should be associated to the VehicleRecordMapper.

3. Instance created is inserted into the database.


Admin
:VehicleRecord Controller
:vehicleRecord Mapper
:vehicleRecord TableDataGateway
:Database
createNewVehicle (request)
insert(String type, String make, String model, int year, String color, String licensePlate, String availability, int cost)
:Catalog Details
create()
return
insert(CatalogDetails)
insert()
return
return
return

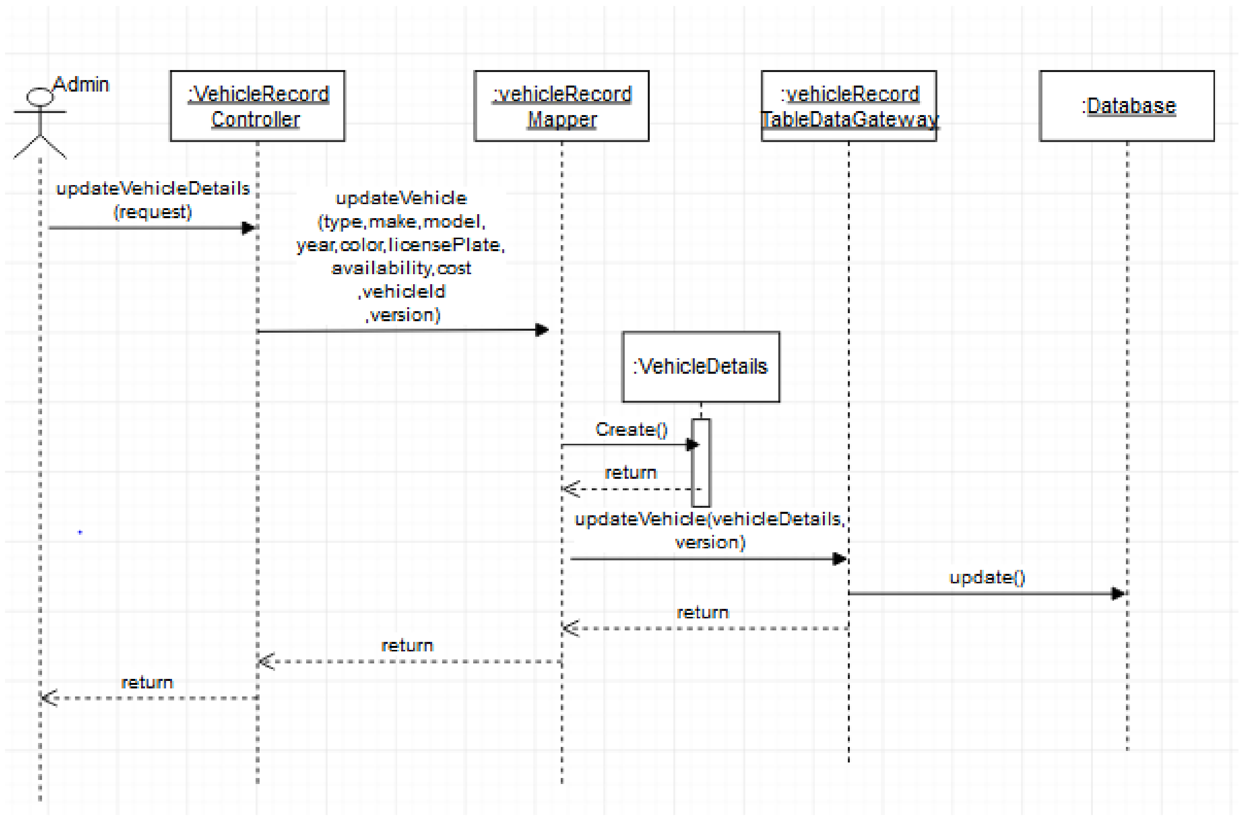
9.5.2: UC12

1. Instance of vehicle will be created.

2.Instance will be associated with rowMapper.

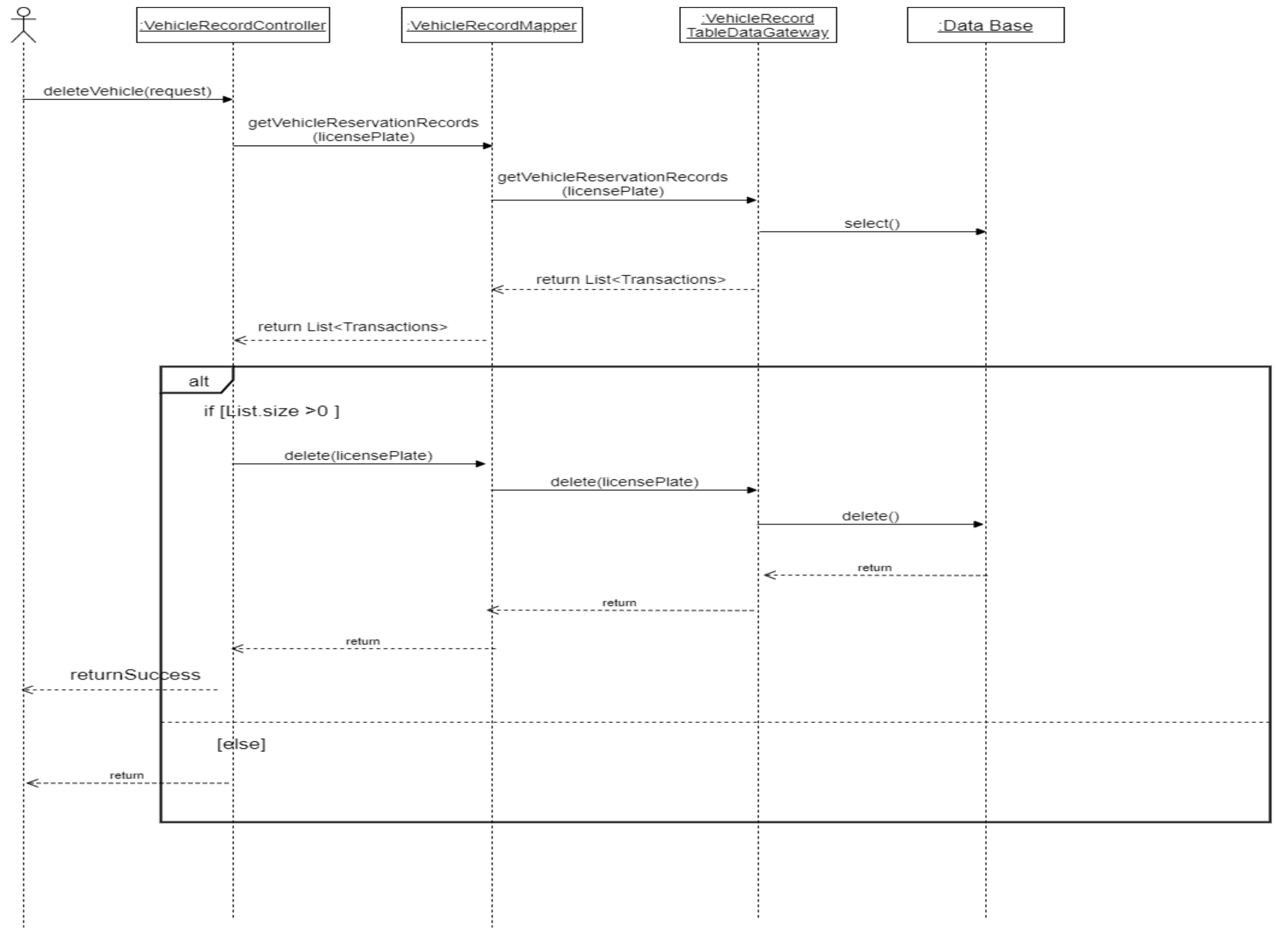
3.Instance will be modified.

4.instance will be persisted in database


Admin
:VehicleRecord Controller
:vehicleRecord Mapper
:vehicleRecord TableDataGateway
:Database
updateVehicleDetails (request)
updateVehicle (type,make,model, year,color,licensePlate, availability,cost ,vehicleId ,version)
:VehicleDetails
Create()
return
updateVehicle(vehicleDetails, version)
update()
return
return
return

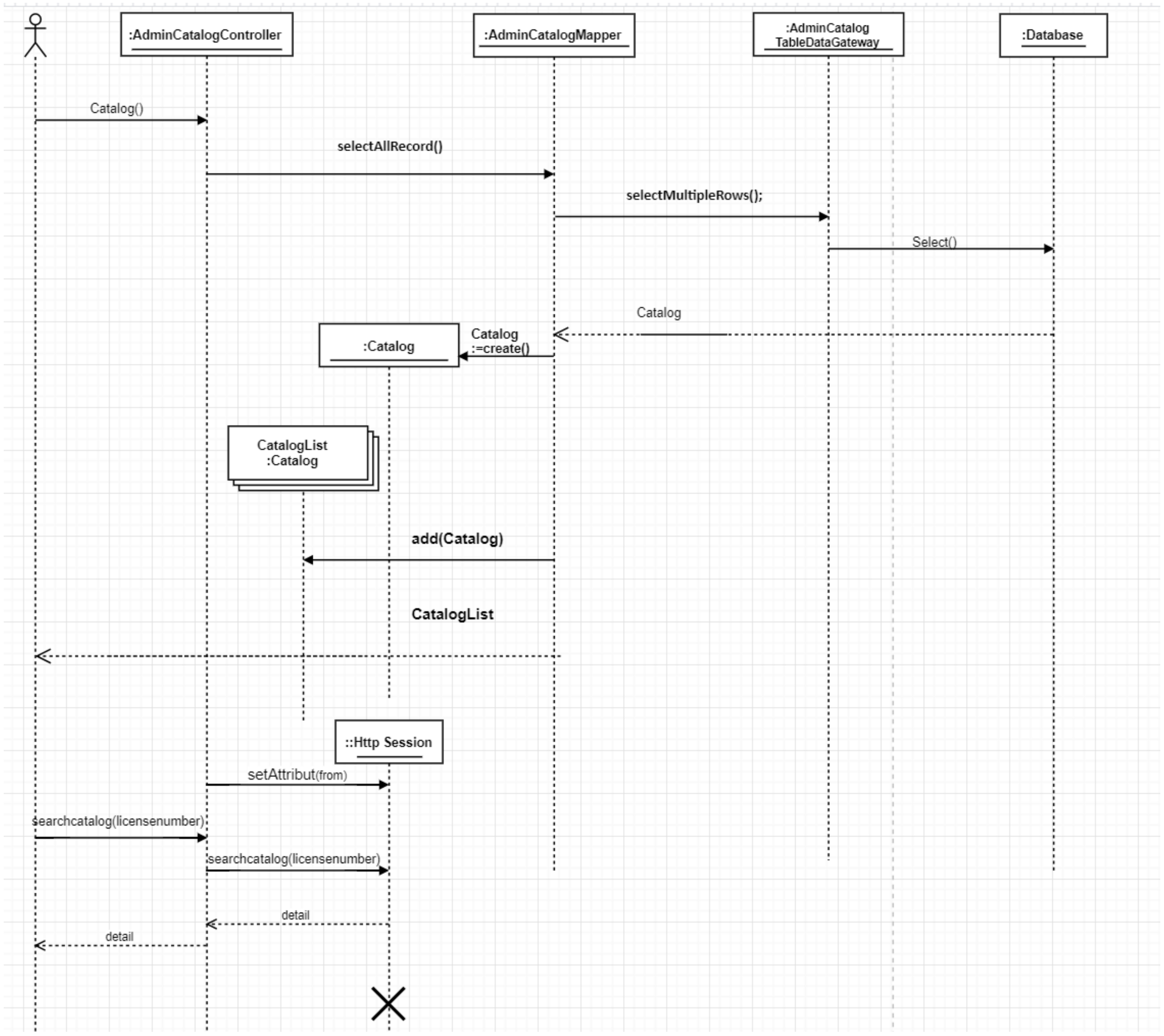
9.5.3:-UC13

1. The instance of vehicle model will be created.
2. The instance is associated with VehicleRecordMapper.
3. Vehicle record will be deleted from the database.

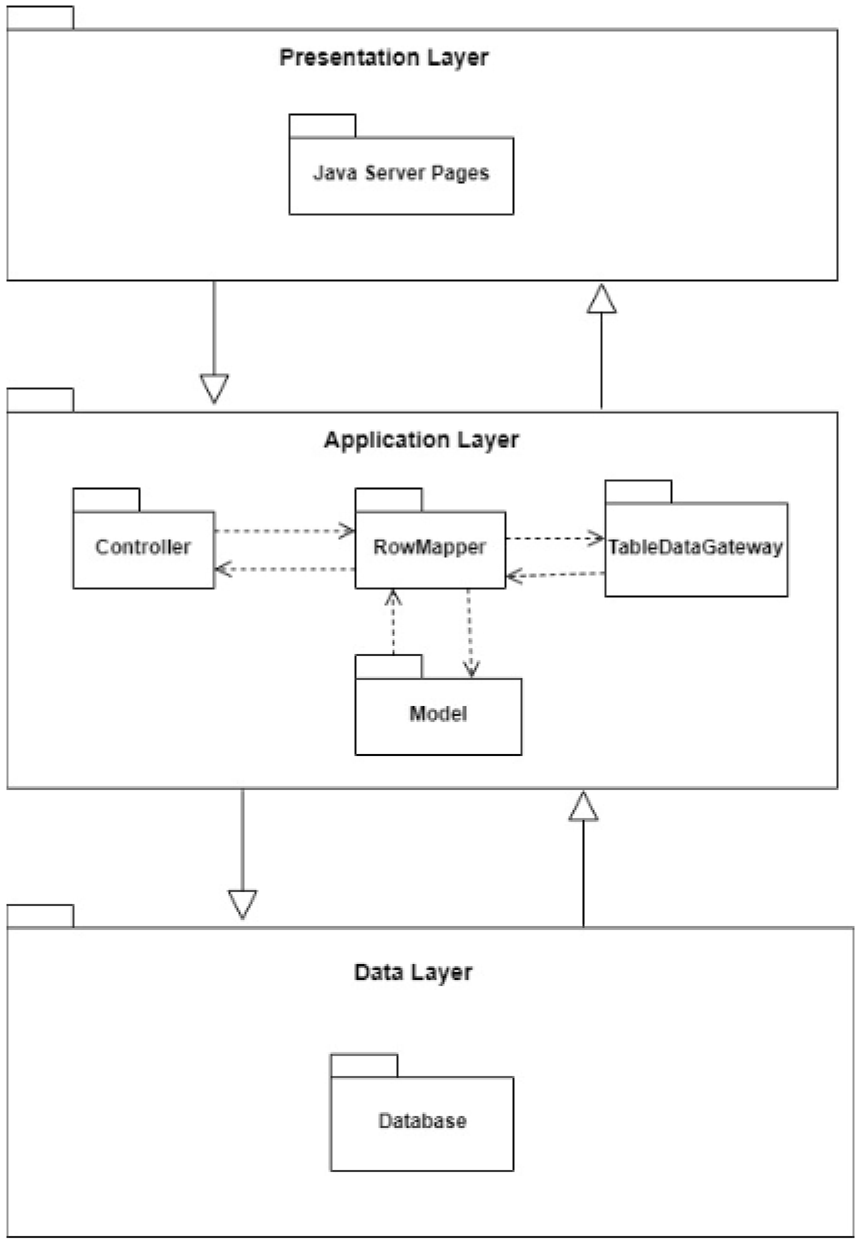


9.5.4:-UC14

1. The instance of List<Catalog> instance is created.
2. instance was associated with Vehicle rowmapper.
3. instance was associated with httpsession.
4. The system displays the desired result on the basis of search criteria.


:AdminCatalogController
:AdminCatalogMapper
:AdminCatalog TableDataGateway
:Database
Catalog()
selectAllRecord()
selectMultipleRows();
Select()
Catalog
:Catalog
Catalog :=create()
CatalogList :Catalog
add(Catalog)
CatalogList
::Http Session
setAttribut(from)
searchcatalog(licensenumber)
searchcatalog(licensenumber)
detail
detail

10. Development (Implementation) view



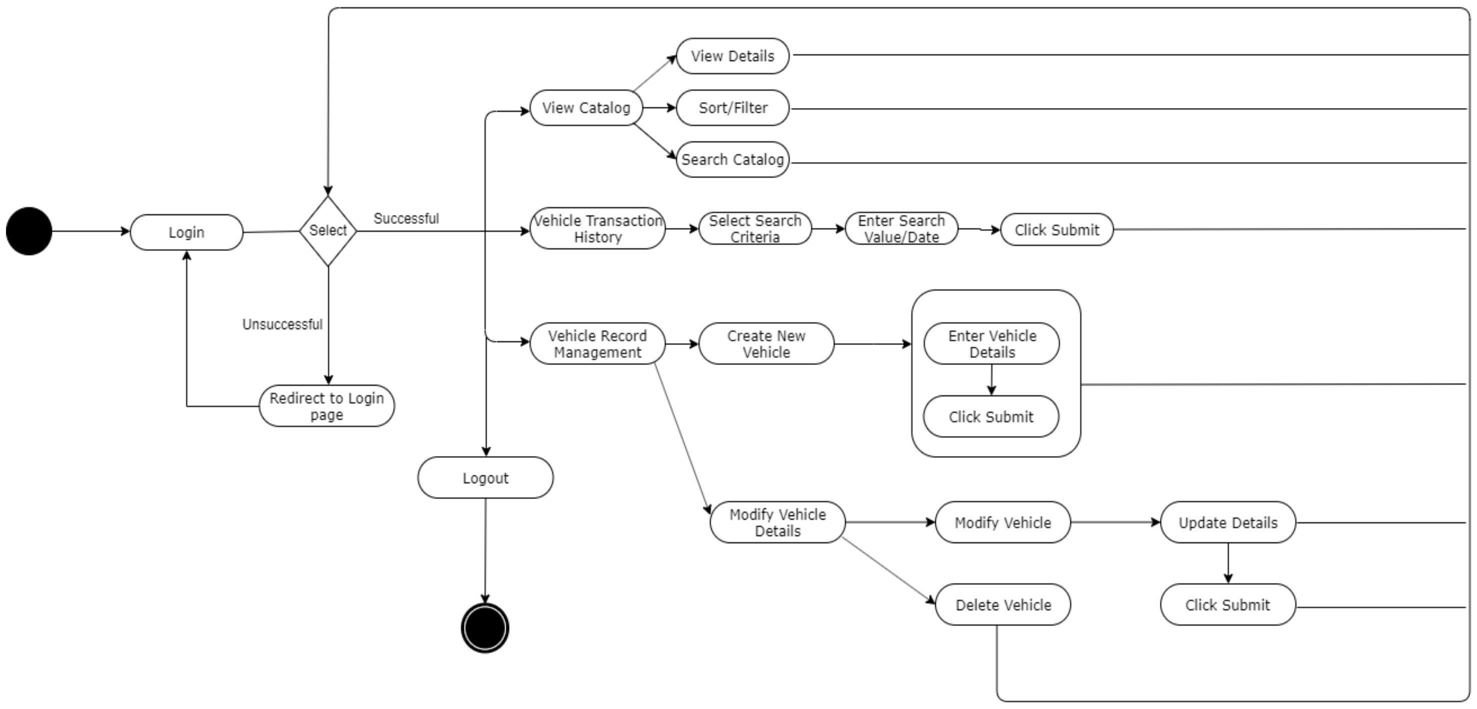
Reuse of components and frameworks

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

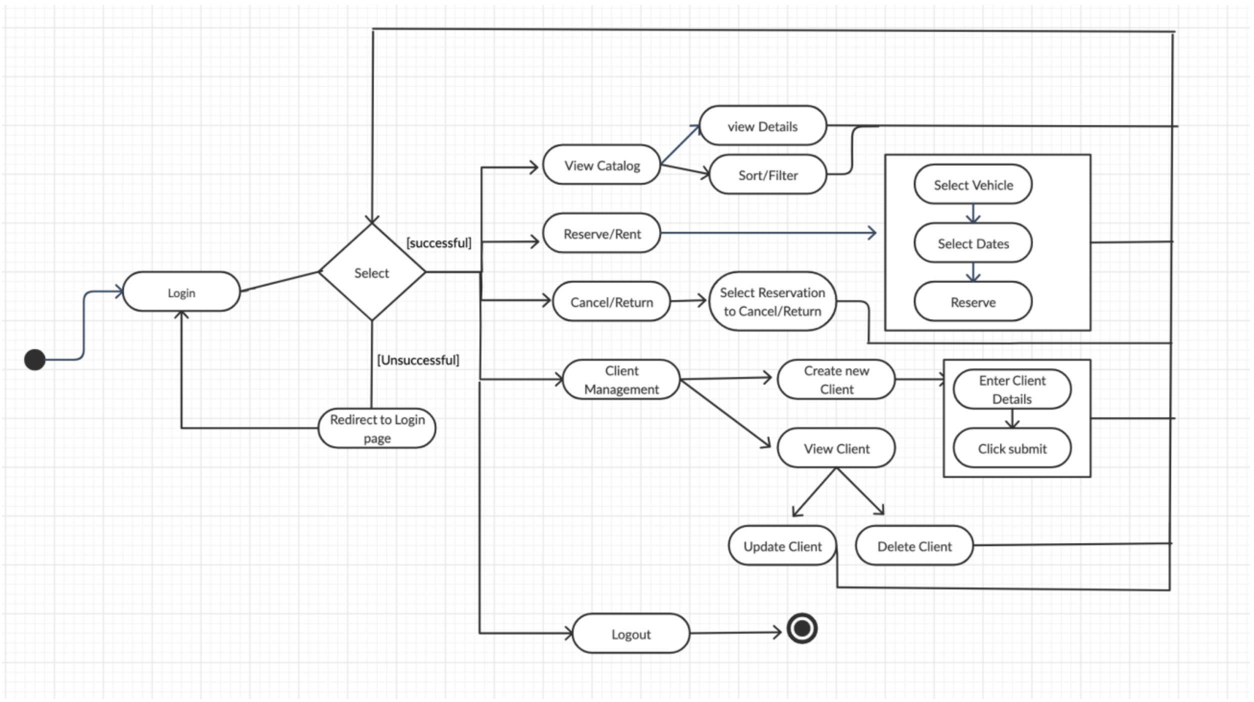
11. Process view

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.

Admin Activity Diagram

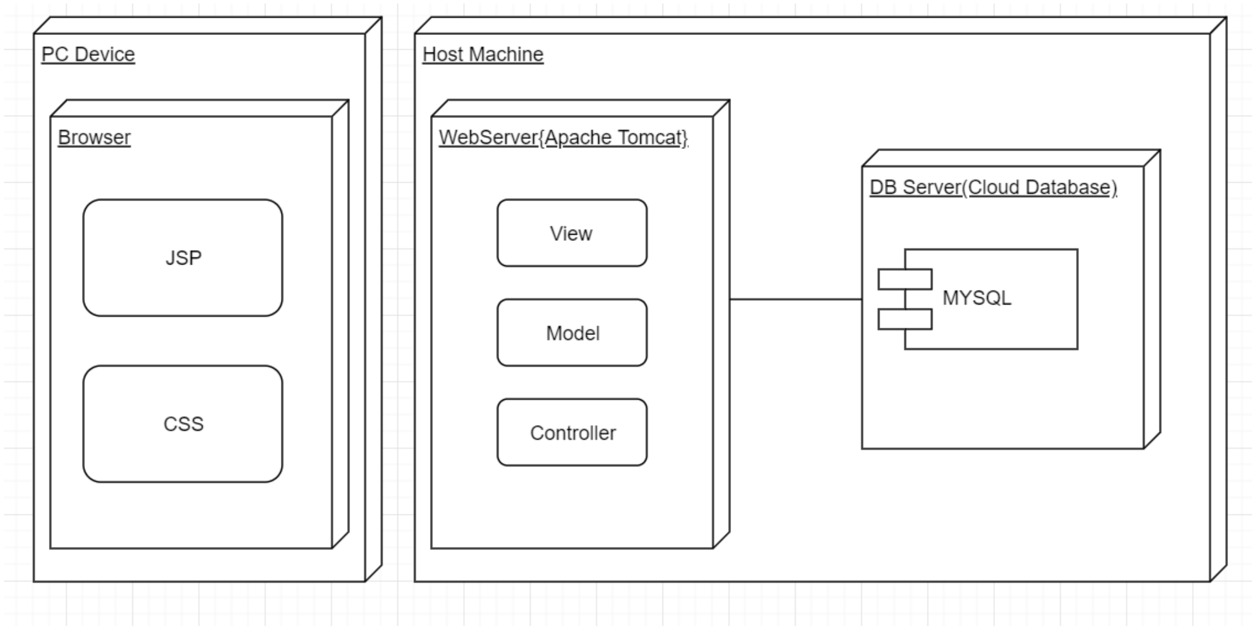


Clerk Activity Diagram:



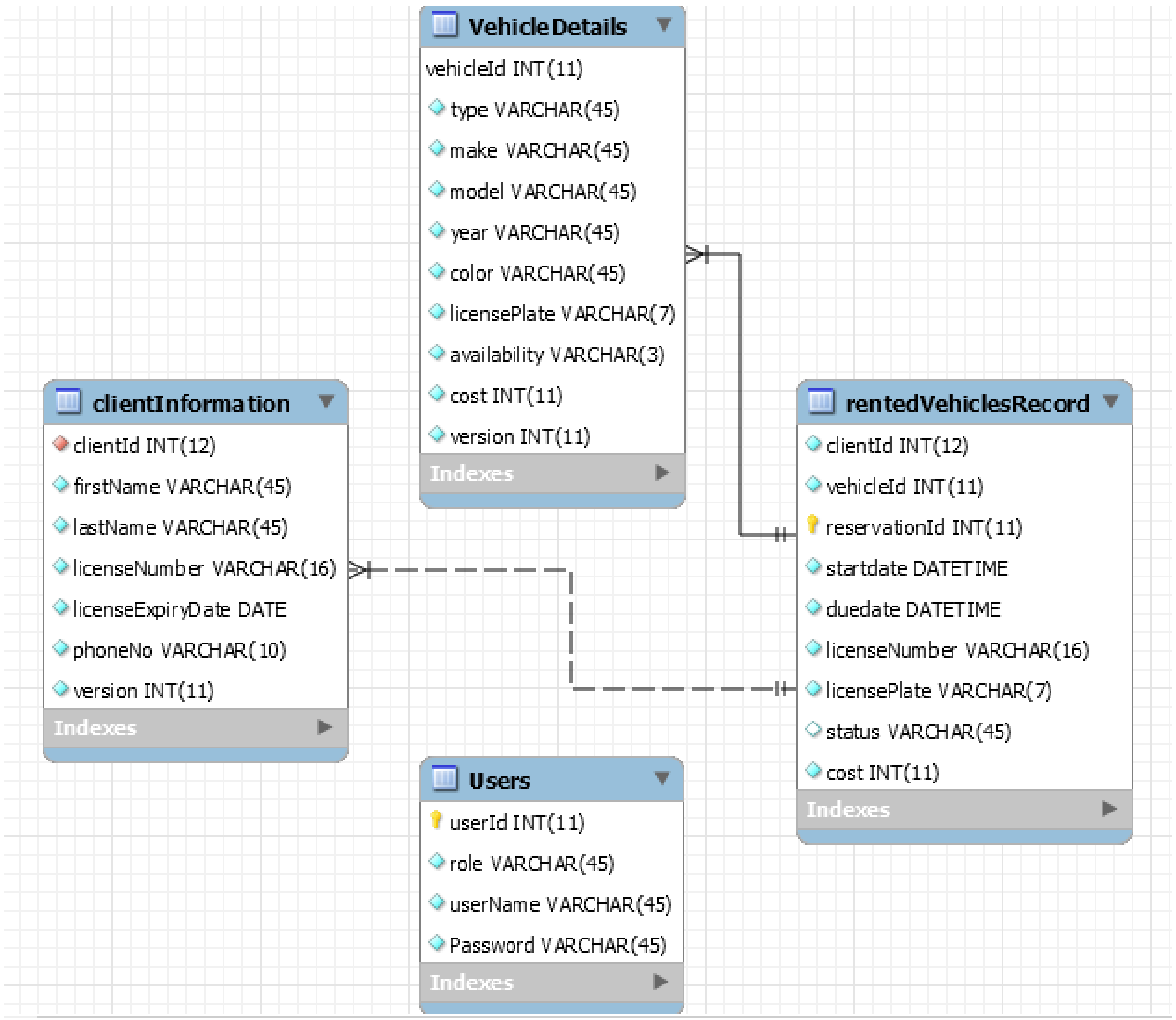
12. Deployment (Physical) view

The deployment (or physical) view illustrates the physical components of the architecture, their connectors and their topology. Describe the physical network and hardware configurations on which the software will be deployed. This includes at least the various physical nodes (computers, CPUs), the interaction between (sub)systems and the connections between these nodes (bus, LAN, point-to-point, messaging, SOAP, http, http).Below is the deployment diagram for this project



13. 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.



14. 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 : The system supports several workload management techniques.

Reliability, Availability:

- Description : Transparent failover mechanism, mean-time-between-failure
- Solution: All possible error exceptions are handled to check that all inputs are in the correct formats using try-catch mechanism and supported by several test cases.
-

Portability:

- Description : Ability to be reused in another environment
- Solution : The system is designed to be easily moved from one computing environment to another, such as browsers and different operating systems.

Security:

- Description : Authentication and authorization mechanisms
- Solution : The system has a login form to verify if the clerk and administrator have the authorization for using the system or not, it only allows the registered predefined clerks and admins to use the application, so no malicious or theft could occur.

15.Pattern List:

S.no	Pattern Name	Class Name/Package Name	Description
1	Singleton	Class Name: JdbcConnectionManager.java	This pattern is implemented for jdbc connection.So this class will have one instance and provide global access to all.
2	DataMapper	Package: com.concordia.soen.sdm.mapper	Mapper class helps to map a data between controller and tabledatagateway by creating a objects.
3	Table Data Gateway	Package: com.concordia.soen.sdm.tableDataG	Table data gateway class contains a sql queries and

		ateway	helps to persist and retrieve data from database. It acts as a bridge between datamapper and database.
4	Optimistic Locking	Class Name:ClientTableDataGateway.java, VehicleRecordTableDataGateway.java	We are using version number as a checkpoint while updating a data in database.If version mismatches data will not get updated.