Software Architecture Document

Version 2.6

for

Soen6461

Prepared by

Pargol Poshtareh	26428126	Pargol.Poshtareh@gmail.com
Richard Grand'Maison	26145965	Rich.Grandmaison@gmail.com
Ravneet Singh Brar	40078628	rippybrar1994@gmail.com
Gurjot Singh	40091208	gurjot5076@gmail.com
Adrien Dat	40072965	ad_dat@encs.concordia.ca

Instructor:	Dr. Constantinos Constantinides
Course:	Soen 6461
Date:	sep 2019

Document history

Date	Version	Description	Author
1 Oct 19	1.0	Purpose, Scope	Gurjot Singh
2 Oct 19	1.1	Architectural Representation	Ravneet Singh Brar
2 Oct 19	1.2	Sequence Diagrams	Ravneet Singh Brar
3 Oct 19	1.3	Class Diagram	Gurjot Singh
4 Oct 19	1.4	Package Diagram	Gurjot Singh
4 October	1.5	More Sequence Diagrams	Ravneet Singh Brar
5th October	1.6	give last modifications	Pargol /Richard
22nd October	2.1	evaluate the missing part	Pargol
24th October	2.2	Added System Sequence Diagrams for admin	Ravneet Singh Brar
24th October	2.3	Updated the package diagram according to new additions	Gurjot Singh
26th October	2.4	Updated Class Diagram	Ravneet Singh Brar
26th October	2.5	Added System Sequence Diagrams for admin	Ravneet Singh Brar
26th October	2.6	Review (Iteration 3 release)	Gurjot/Ravneet

Table of contents

List of figures

1. Introduction

The introduction of the Software Architecture Document provides an overview of the entire document.

Purpose

This document describes the overall structure of our Rental application, which comprises of multiple architectural views including logical view, use case view, process view etc. Different stakeholders can make use of this document as a reference for understanding the application from different perspectives or carrying out various tasks. For example, the maintainers can refer to the class structure, method signatures, data types etc, for carrying out maintenance or feature addition activities. Similarly, the client can take a broader view of the application by referring to the UML diagram.

Scope

This document encapsulates the project structure in different views. Additionally, the behaviors and relationships among different project modules is described. It demonstrates the efficient structure and robustness of our application. At the same time, it will help the programmers and maintainers of this application to easily understand the interactions happening in the system among different modules, which will furthermore facilitate easy maintenance and new feature additions.

Definitions, acronyms, and abbreviations

Below are the abbreviations used in this document:-

UML: Unified Modeling Language

SAD: Software Architecture Document

MVVM: Model View ViewModel

SRS: System requirement specification

2. Architectural representation

In the given application , Model View ViewModel (MVVM) Architecture is being implied.It has three basic components named as Model, View and ViewModel.

a. Model - It contains all the data of the application and has nothing to do with the actual logic of the application

b. View - View is what the user sees on the screen. It represents the data in the model for the user and takes the input by the user and forwards it to the ViewModel via the data binding that is defined between the View and Model.

c. ViewModel - In MVVM , the view has an automatic binding to the ViewModel and can send and receive updates automatically. This means ViewModel does not have a reference to the view.

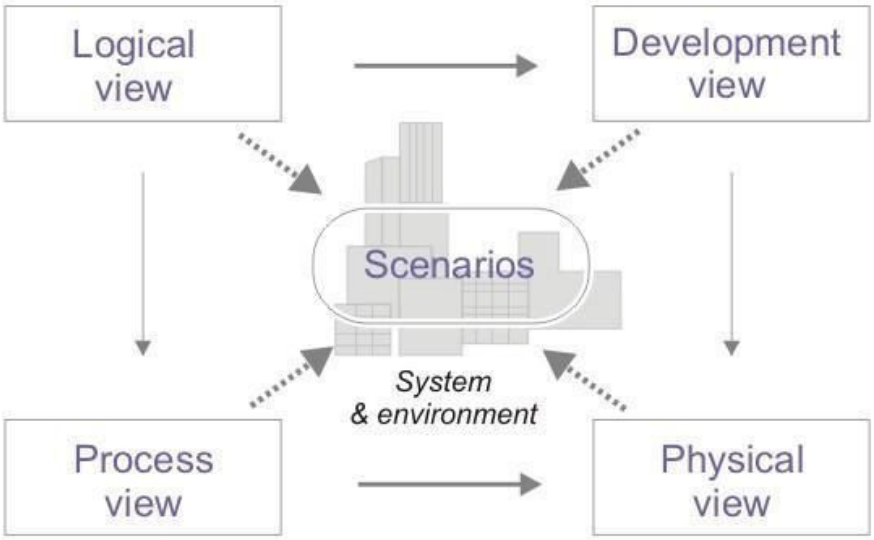
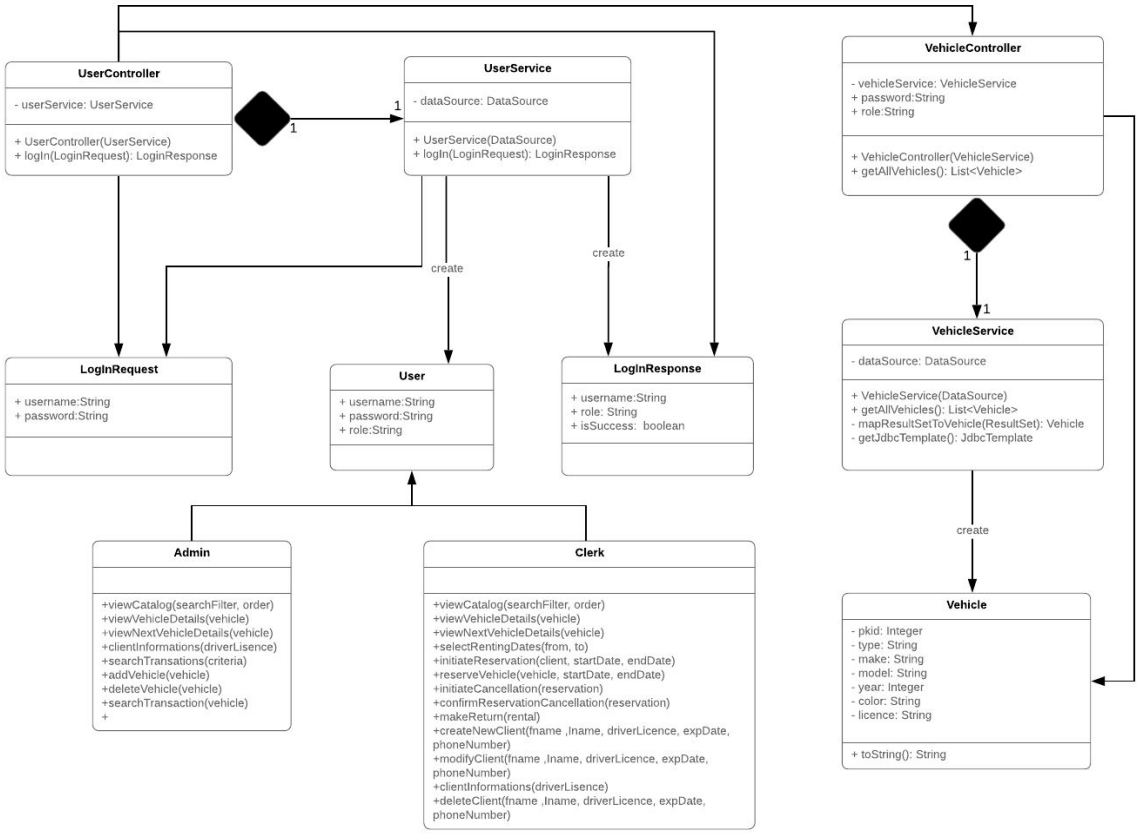


Figure 1: The 4+1 view model.

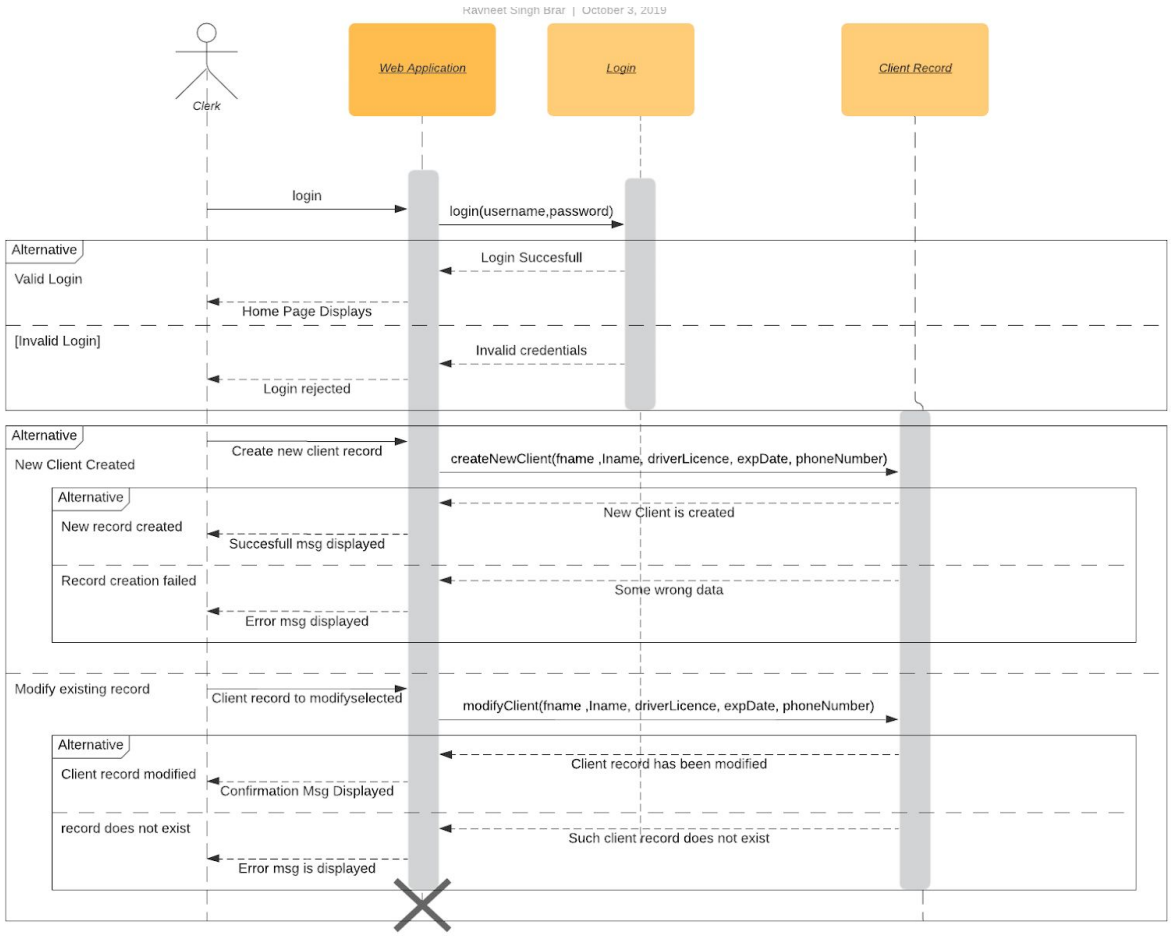
1. **Logical view** :

CLASS DIAGRAM:-

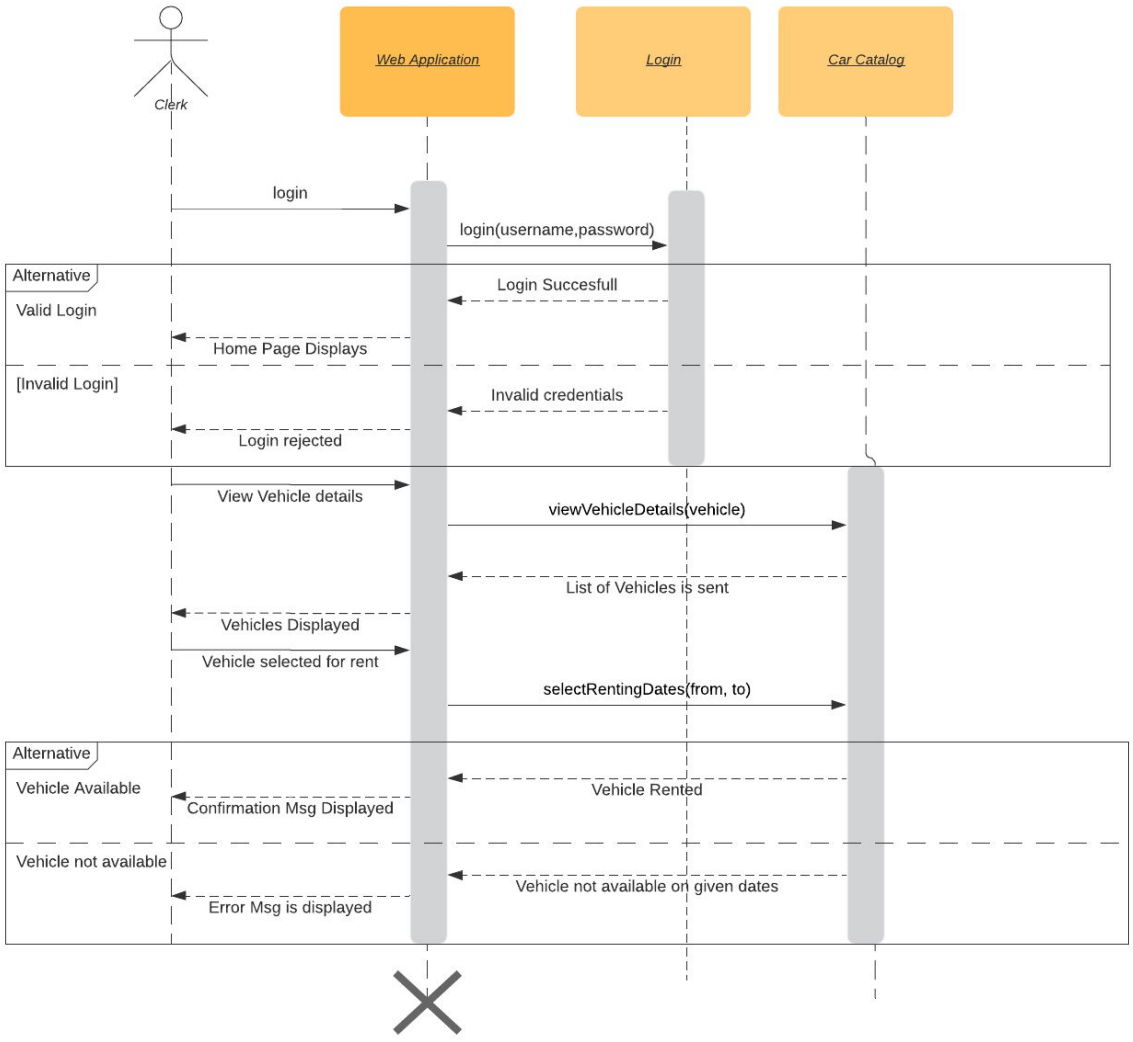


SEQUENCE DIAGRAM :-

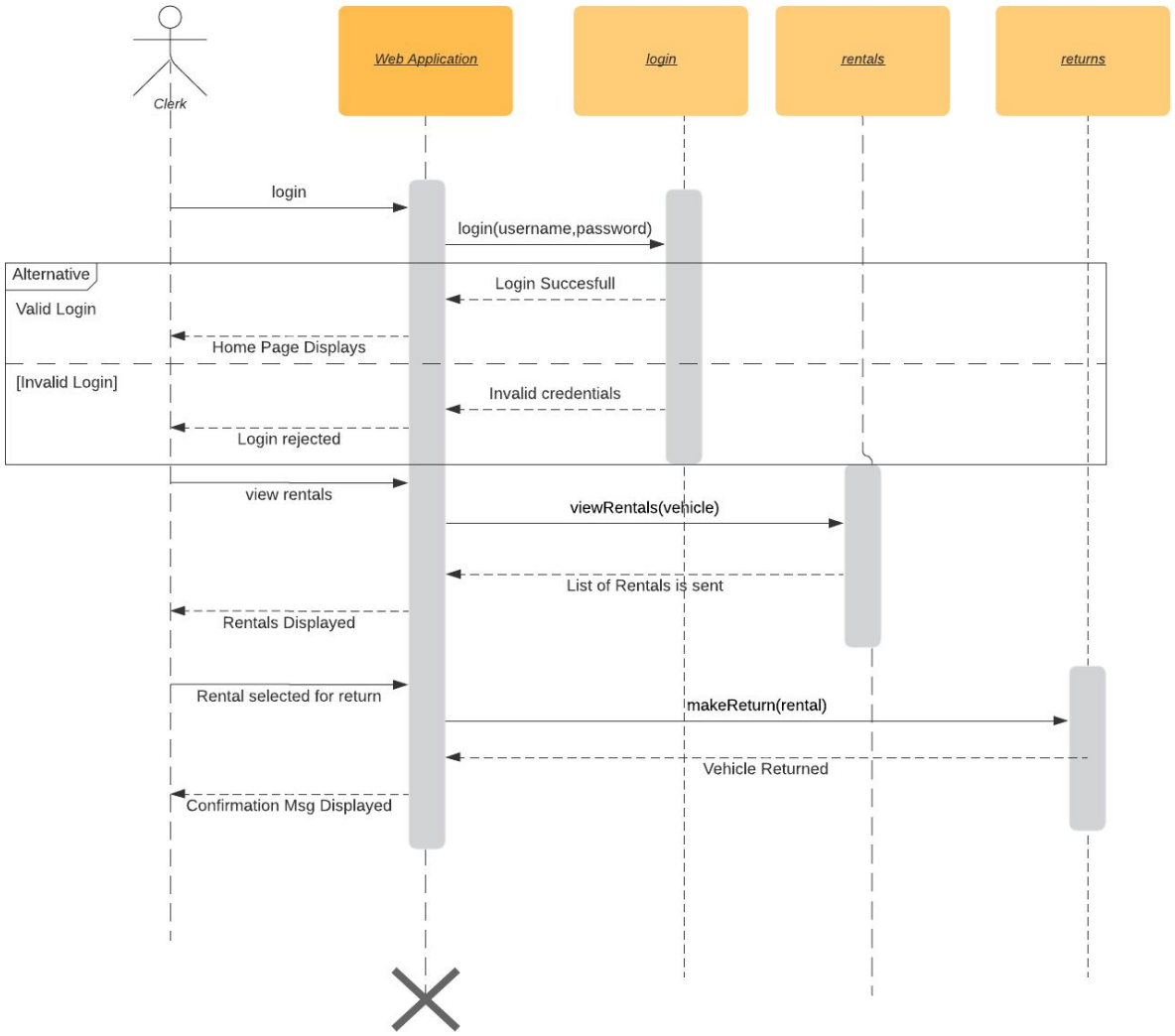
1. **Creating/Modifying Client Record**



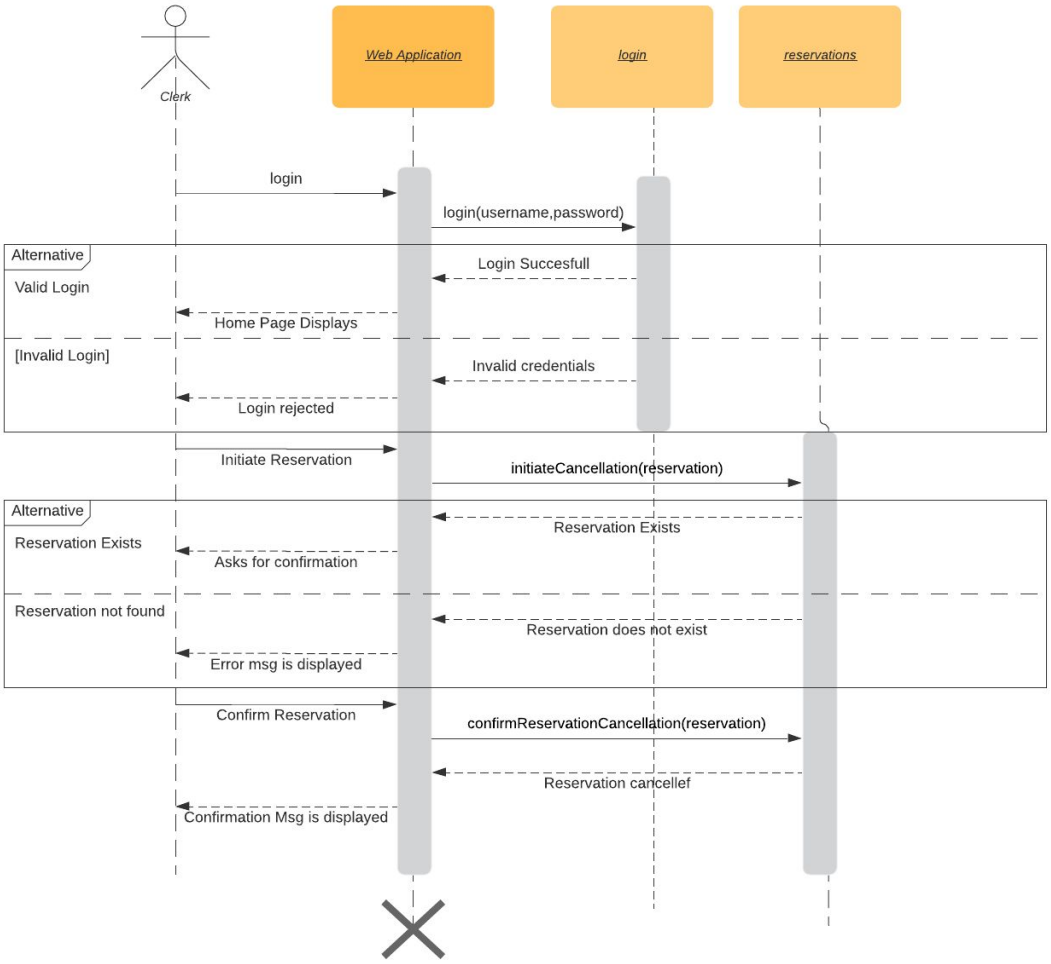
2. Renting car



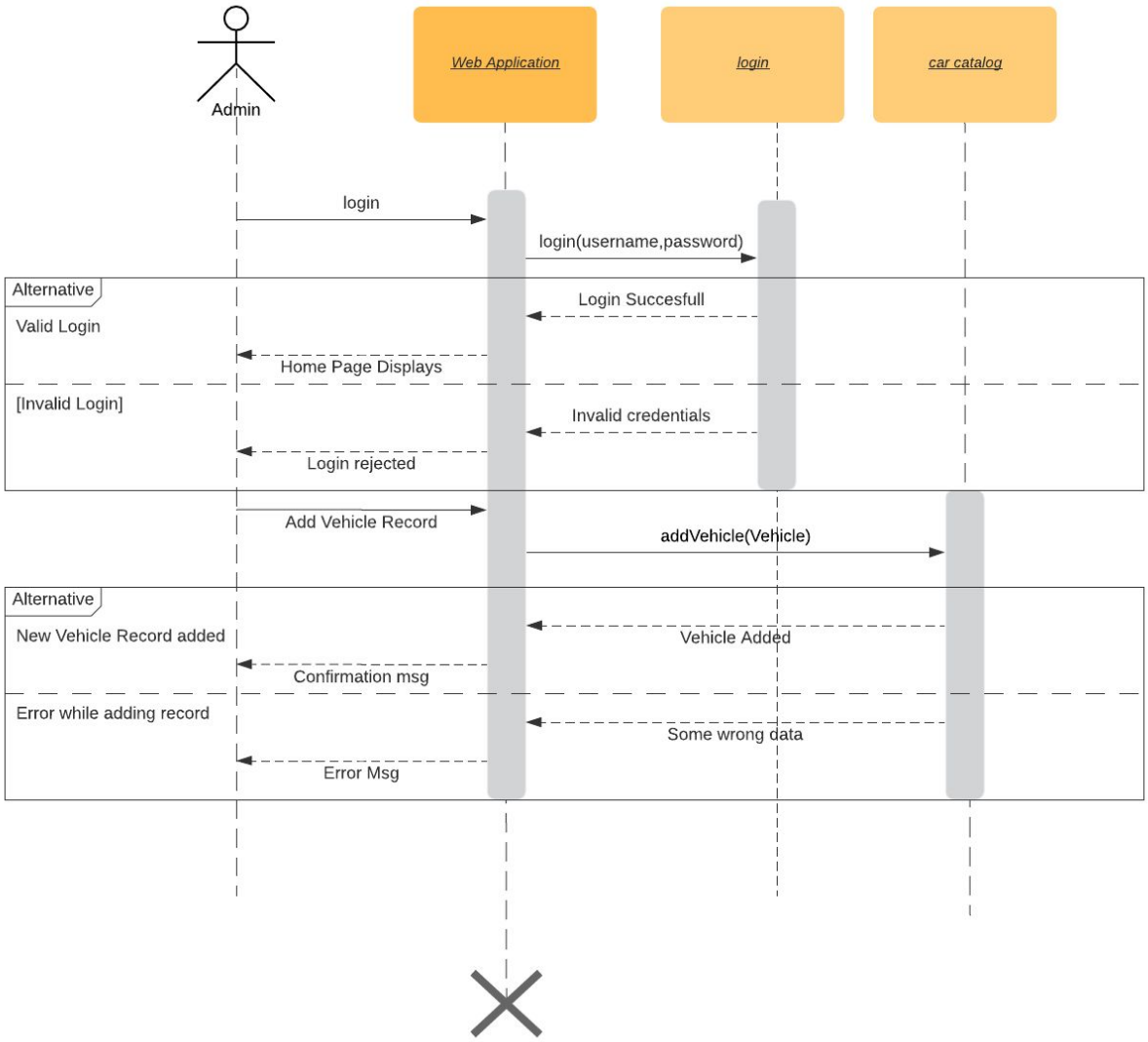
3. Handle Return


Clerk
Web Application
login
rentals
returns
login
login(username,password)
Alternative
Valid Login
Login Succesfull
Home Page Displays
[Invalid Login]
Invalid credentials
Login rejected
view rentals
viewRentals(vehicle)
List of Rentals is sent
Rentals Displayed
Rental selected for return
makeReturn(rental)
Vehicle Returned
Confirmation Msg Displayed

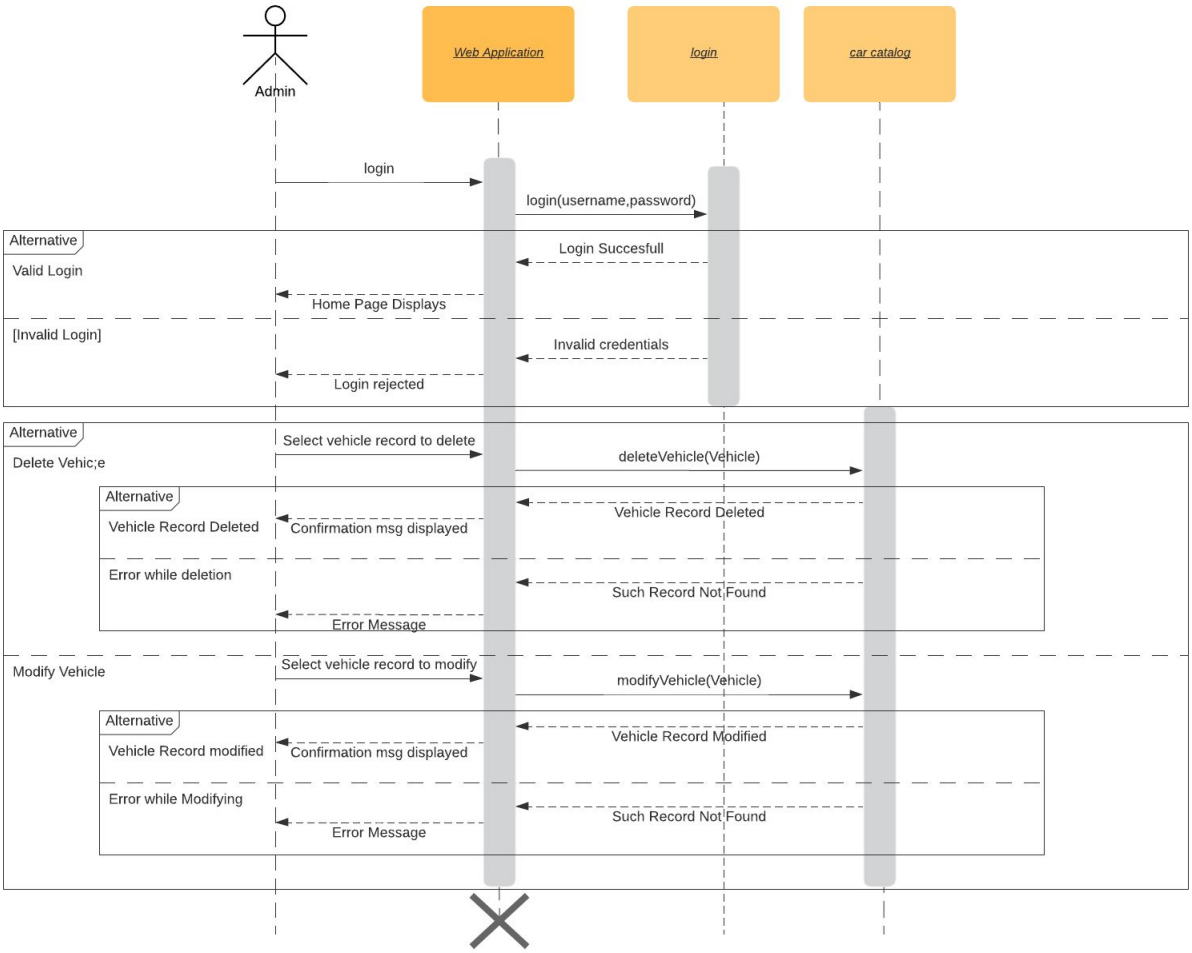
4. Cancel Reservation



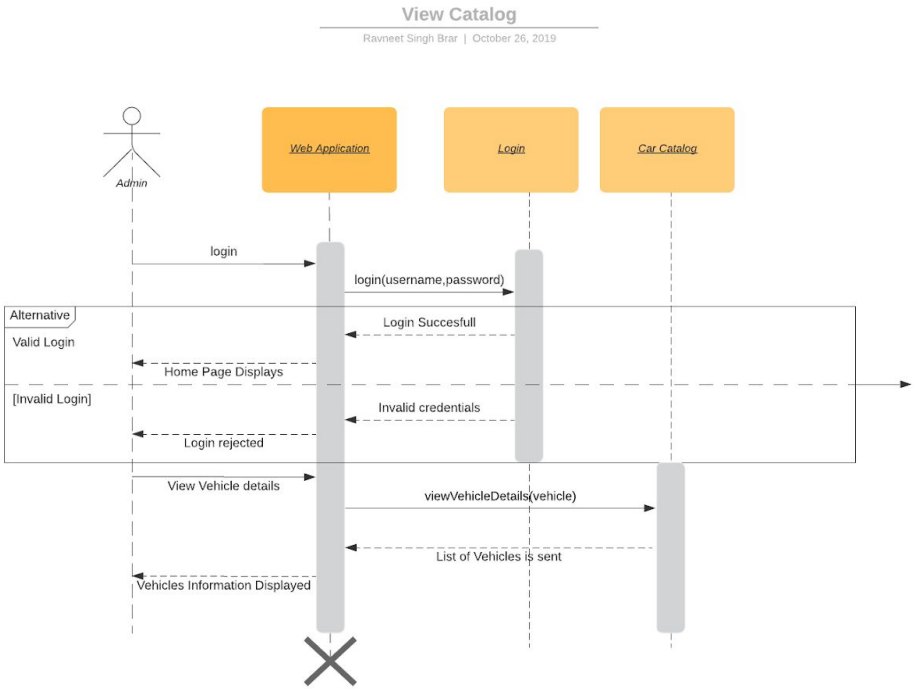
5. Create Vehicle Record



6. Delete/Modify Vehicle Record

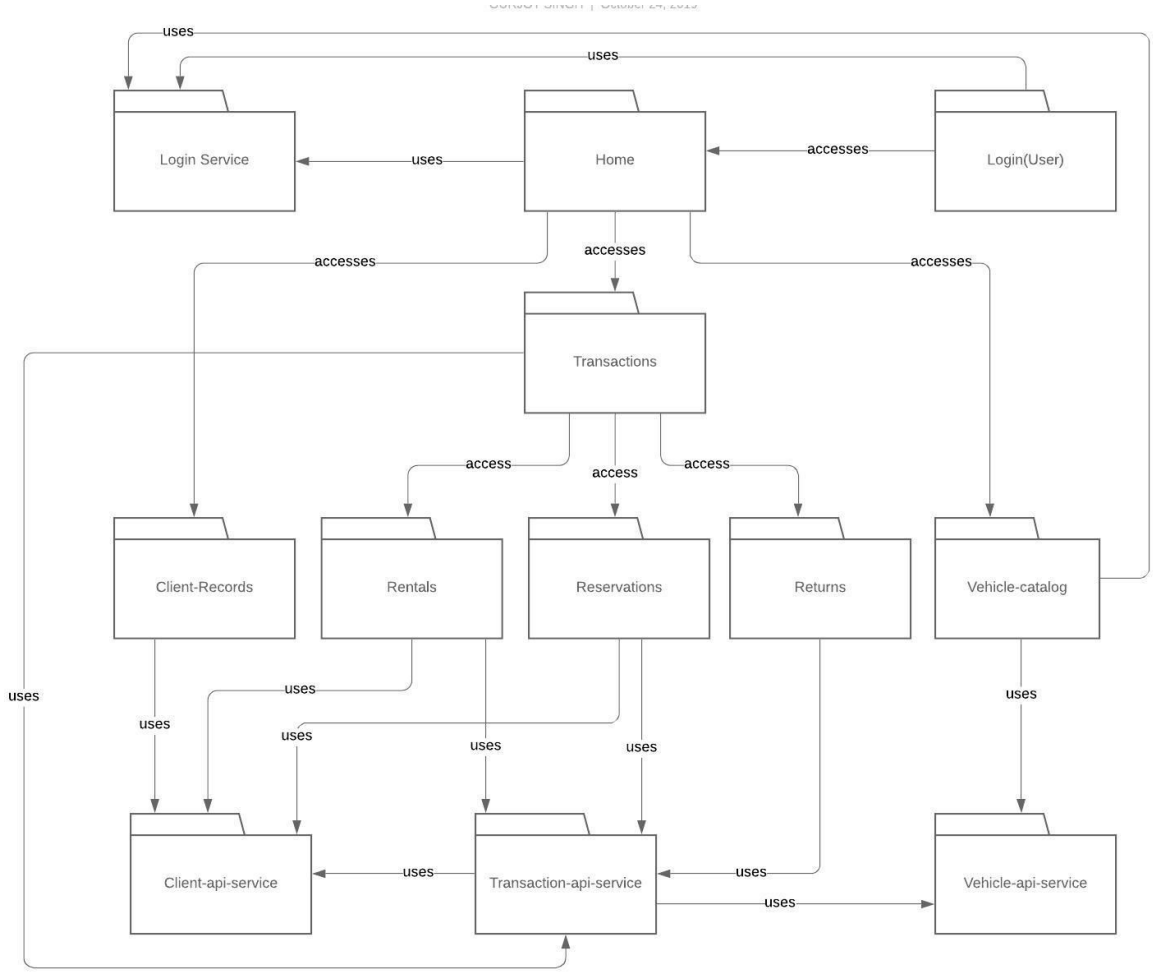


7. View Catalog(Admin)



2. **Development view** (also known as Implementation view):

PACKAGE DIAGRAM:-



3. **Process view** : 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**.

4. **Physical view** (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**.

5. **Use case view** (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**.

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

In this iteration, persistence is not required, therefore, we hardcoded the client and vehicles data, therefore we did not include database as of now.

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).

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.
Use Case	Search Catalog	We need to make a link from Data to View (using MVVM Angular)	

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.

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.

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 actorsd and the system (seen as a black box) as well as between the system and external systemsTthe *UML use case diagram* is used to capture this view.

9. 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.

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

Subsystems

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

Architecturally significant design packages

Desribe packages of individual subsystems that are architecturally significant. For each package nclude 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.

10. 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.

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 illustrates the system's processes, focusing on the runtime behavior of the system. The view illustrates parallelism and concurrency. Deploy a *UML activity diagram* to capture this view.

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). Use a *UML deployment diagram* to capture this view.

Name	Type	Description
Name of the node.	Node type.	Technical specifications.

13. Data view (optional)

An enterprise software system would additonally 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 : 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 deploy onto any J2EE application server

Security:

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