Software Architecture Documentation

Version 1.0

for

SOEN6461 Coursework Project

Prepared by

Nirav Ashvinkumar Patel	40081268	niravjdn@gmail.com
Charles Jebalitherson Augustin Moses	40084105	charlesjebalitherson@gmail.com
Jemish Kishor Paghadar	40080723	jemish.paghadar27@gmail.com
Vikramjit Singh	40075774	vikramsandhu008@gmail.com
Avinash Damodaran	40078258	avinashdamu323@gmail.com

Document history

Date	Version	Description	Author
09/18/2019	1.0	Initial Version	Nirav Patel
09/18/2019	1.0	Introduction	Avinash Damodaran
09/23/2019	1.0	Added Non-functional Requirements	Jemish Paghadar
09/24/2019	1.0	Added Description for Architecture and Goal	Nirav Patel
09/24/2019	1.0	Added quality attributes	Charles Jebalitherson
09/24/2019	1.0	Added Architectural representation (4+1 View)	Jemish Paghadar
09/24/2019	1.0	Added Functional requirements and use case scenarios	Nirav Patel
09/25/2019	1.0	Data Model	Jemish Paghadar
09/25/2019	1.0	Activity Diagram	Avinash Damodaran
09/25/2019	1.0	Process View Diagram	Avinash Damodaran
09/25/2019	1.0	Activity Diagram	Vikramjit Singh
09/25/2019	1.0	Added Description for deployment diagram	Nirav Patel
09/26/2019	1.0	Added deployment diagram	Nirav Patel

09/27/2019	1.0	Added development view	Charles Jebalitherson
09/27/2019	1.0	Added package diagram for implementation view	Charles Jebalitherson
10/02/2019	1.0	Added Data Model Diagram	Vikramjit Singh
10/03/2019	1.0	Updated Development view	Charles Jebalitherson
10/06/2019	1.0	Added Class Diagram	Vikramjit Singh
10/06/2019	1.0	Updated Quality attributes	Charles Jebalitherson
10/06/2019	2.0	Updated Class Diagram	Vikramjijt Singh

List of figures

Figure 6: Process View

Figure 7: Deployment Diagram

1. Introduction

This Software Architecture Document provides a comprehensive overview of the Vehicle Rental System. It provides the purpose, scope, definitions, acronyms, abbreviations and references used for this document.

Purpose

The Vehicle Rental System Software Architecture Document is used to help stakeholders identify requirements that affect high-level structure of the system. It is composed of many different sections (see Table of Contents), each having a role in describing and illustrating a particular piece of the system by using design artifacts. It also describes any relations between 1 or more software elements. This document is intended for all stakeholders, but mostly developers, who wish to understand more about the design decisions and patterns used to build the system.

Scope

The scope of this document is the entire Vehicle Rental System architecture. It does not describe how to deploy Vehicle Rental System onto a production server. All other processes are described in this document. It describes The class structure and how they communicate. It describes the actions taken by each class for every critical use case.

Definitions, acronyms, and abbreviations

Provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the Software Architecture Document. This information may be provided by reference to the project's Glossary. For example:

- **UML:** Unified Modeling Language
- **SAD:** Software Architecture Document
- **SRS:** Software Requirements Specifications

2. Architectural representation

Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements of the system, as well as some other, non-functional requirements such as reliability, scalability, portability, and availability.

Software architecture = {Elements, Forms, Rationale/Constraints}

Software architecture deals with abstraction, with decomposition and composition, with style and esthetics. To describe a software architecture, we use a model composed of multiple views or perspectives. In order to eventually address large and challenging architectures, the model we propose is made up of five main views: The logical view, which is the object model of the design (when an object-oriented design method is used), the process view, which captures the concurrency and synchronization aspects of the design, the physical view, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect, the development view, which describes the static organization of the software in its development environment.

The description of an architecture(the decisions made) can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios.

The top-level architectural style of the system and the view model that we adopted is based on many enterprise software systems using the 4+1 view illustrated in Figure 1.

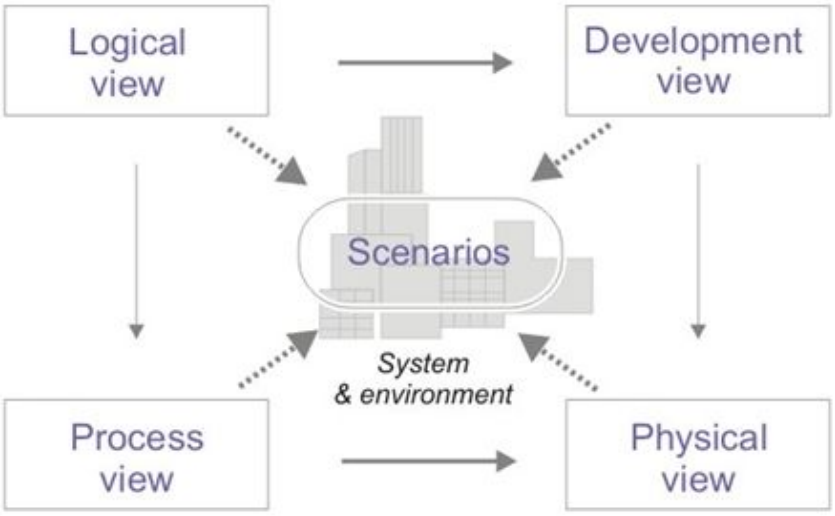

Figure 1: 4+1 View Model

1. **Logical View** (The Object-Oriented Decomposition)

   Audience: Designers
   Related Artifacts : Logical Model
   The logical architecture primarily supports the functional requirements - what the system should provide in terms of services to its users. The system is decomposed into a set of key abstractions, taken (mostly) from the problem domain, in the form of objects or object classes. They exploit the principles of abstraction, encapsulation, and inheritance. This decomposition is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across the various parts of the system. We use the Rational/Booch approach for representing the logical architecture, by means of class diagrams and class templates. A class diagram shows a set of classes and their logical relationships: association, usage, composition, inheritance, and so forth. Sets of related classes can be grouped into class categories. Class templates focus on each individual class; they emphasize the main class operations, and identify key object characteristics.

2. **Development View** (Subsystem Decomposition)

   Audience: Programmers
   Related Artifacts : Development Model
   The development architecture focuses on the actual software module organization on the software development environment. The software is packaged in small chunks—program libraries, or subsystems - that can be developed by one or a small number of developers. The subsystems are organized in a hierarchy of layers, each layer providing a narrow and well-defined interface to the layers above it. The development architecture of the system is represented by module and subsystem diagrams, showing the ‘export’ and ‘import’ relationships. The complete development architecture can only be described when all the elements of the software have been identified. It is, however, possible to list the rules that govern the development architecture: partitioning, grouping, visibility. For the most part, the development architecture takes into account internal requirements related to the ease of development, software management, reuse or commonality, and to the constraints imposed by the toolset, or the programming language. The development view serves as the basis for requirement allocation, for allocation of work to teams (or even for team organization), for cost evaluation and planning, for monitoring the progress of the project, for reasoning about software reuse, portability and security. It is the basis for establishing a line-of-product. UML Diagrams to represent development view include the package diagram.

3. **Process View** (The Process Decomposition)

Audience: Integrators
Related Artifacts : Process Model
The process architecture takes into account some non-functional requirements, such as performance and availability. It addresses issues of concurrency and distribution, of system's integrity, of fault-tolerance, and how the main abstractions from the logical view fit within the process architecture - on which thread of control is an operation for an object actually executed. The process architecture can be described at several levels of abstraction, each level addressing different concerns. UML Diagrams to represent process view include the activity diagram.

4. **Physical View** (Mapping the software to the hardware)

Audience: Deployment managers
Related Artifacts : Physical Model
The physical architecture takes into account primarily the non-functional requirements of the system such as availability, reliability (fault-tolerance), performance (throughput), and scalability. The software executes on a network of computers, or processing nodes. The various elements identified - networks, processes, tasks, and objects - need to be mapped onto the various nodes. We expect that several different physical configurations will be used: some for development and testing, others for the deployment of the system for various sites or for different customers. The mapping of the software to the nodes therefore needs to be highly flexible and have a minimal impact on the source code itself. UML Diagrams used to represent physical view include the deployment diagram.

5. **Use Case View** (Putting it all together)

Audience: all the stakeholders of the system, including the end-users
Related Artifacts : Use Case Model
The elements in the four views are shown to work together seamlessly by the use of a small set of important scenarios - instances of more general use cases - for which we describe the corresponding scripts (sequences of interactions between objects, and between processes). The scenarios are in some sense an abstraction of the most important requirements. Their design is expressed using object scenario diagrams.
This view is redundant with the other ones (hence the "+1"), but it serves two main purposes: 1) as a driver to discover the architectural elements during the architecture design 2) as a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

6. **Data View**

Audience: Data specialists, Database administrators
Related Artifacts : Data Model
A data model is commonly created to describe the structure of the data handled in information systems and persisted in database management systems. That structure is

often represented in entity-relationship diagrams or UML class diagrams. These diagrams basically show data entities and their relationships. The data model for a given system can be seen as an architectural view. Code units (e.g., classes, packages) and runtime components (e.g., processes, threads) are most commonly regarded as software architecture elements. The data model showing the structure of the database in terms of data entities and their relationships is another example. Among other practical purposes, the data model serves as the blueprint for the physical database, helps implementation of the data access layer of the system, and has a strong impact on performance and modifiability.

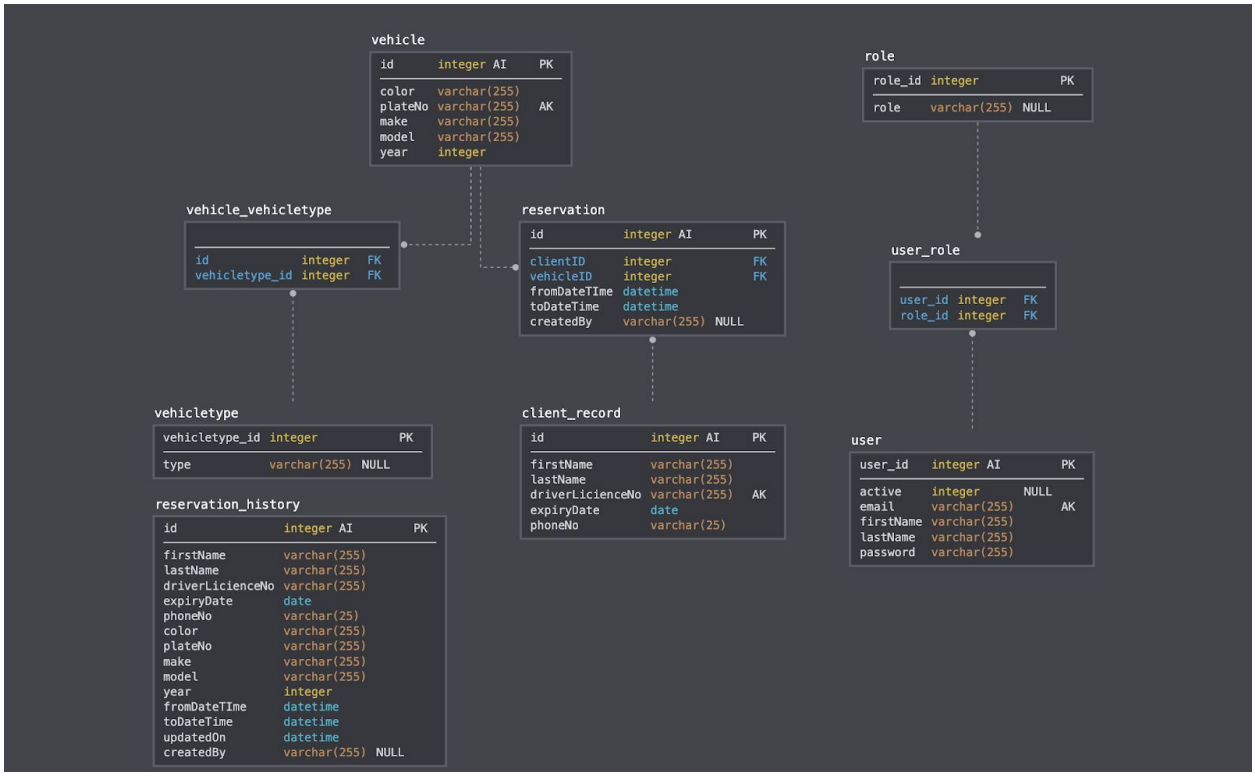

Figure 2: Data Model

3. Architectural requirements: goals and constraints

This section contains all requirements in detail: Functional as well as non-functional requirements (quality attributes and constraints). The quality attributes are listed according to the ISO/IEC 25010 standard that classifies software quality in a structured set of characteristics and sub-characteristics. These requirements are already described in SRS. In this section describe key requirements and constraints that have a significant impact on the architecture.

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system is meant as a proof of concept for a more complete project prediction system to be built in the future. Therefore, one of the primary stakeholders in this document and the system as a whole are future architects and designers, not necessarily users as is normally the case. As a result, one goal of this document is to be useful to future architects, programmers and designers.

2. The system will be written using Java Spring(5.1) with Java 8 technologies but will use an open source RDBMS system (MySQL) for data persistence and will be deployed to a Linux webserver running Tomcat 8. These special deployment requirements require additional consideration in the development of the architecture.

3. Section 3.3(Non-Functional Requirements such as scalability) of the Software Requirements Specification outlines a number of anticipated changes that the application could face over time. One of the primary goals of the system architecture is to minimize the impact of these changes by minimizing the amount of code that would need to be modified to implement them. The architecture seeks to do this through the use of modularization and information hiding to isolate components that are likely to change from the rest of the system.

The purpose of the use-case view is to give additional context surrounding the usage of the system and the interactions between its components. For the purposes of this document, each component is considered a use-case actor. In section 3.2, the most common use-cases are outlined and illustrated using UML use-case diagrams and sequence diagrams to clarify the interactions between components.

The key design goals would be:
- Usability
- Stability -
- Platform independence
- 3-Tier design methodologies to enable efficient and responsive system

3.1.Functional Requirements (Use Case View)

Clerk Primary Tasks

- Log in

Use case name	Log in
Scenario	Log in to scenario using username(email) and password
Triggering event	User enters a username and password and submit
Brief description	When user enters correct username and password, user is redirect to proper dashboard or page depending on the type of the user, else error is shown saying Credentials are incorrect.
Actor	Clerk
Related use case	None
Pre-condition	The user should be valid and active user.
Post-condition	The user would be redirected to proper home page based on user type.
Flow of event	User loads login page. User enters username and password. If this is valid user then user is redirect to proper home page.
Exception Condition	If entered username and password id wrong, user shall be asked to login again.

- Change Password

Use case name	Change password
Scenario	User wants to change his/her login password.
Triggering event	User clicks on change password button.

Brief description	When user enters a new password two times in two separate input field and they both are the same, password of user will be changed.
Actor	Clerk
Related use case	Login to system.
Pre-condition	User should be already logged in.
Post-condition	User password is changed.
Flow of event	User clicks on change password link. Form asks new password to be entered two times to make sure it is entered continuously. If both fields have the same value, user's password is changed.
Exception Condition	If both fields does not have the same value, then it shows an error saying both fields should have the same value.

- Log out

Use case name	Log out
Scenario	User who has currently logged into the system logs out.
Triggering event	User clicks logout button
Brief description	Whcn a user who has already logged in to thc system clicks "logout" he'll get log out from the system, so that the session variables will get reset.
Actor	Clerk
Related use case	Login to the system.
Pre-condition	User should be logged in.
Post-condition	User should get redirected to the login page

Flow of event	User clicks "logout". System removes the session values for this user User gets redirected to the login page.
Exception Condition	None

- Make Reservations

Use case name	Clerk makes a reservation for a client.
Scenario	Clerk makes a reservation for a client for a particular vehicle with specifications for date from when vehicle is allocated to client with its due date.
Triggering event	User selects client and vehicle and fills up details for start date of reservation with due date of reservation and click on submit.
Brief description	Clerk makes a reservation for a client for a particular date to lent a vehicle to client.
Actor	Clerk
Related use case	None
Pre-condition	User is logged in to the system.
Post-condition	A reservation has been created for a client with associated vehicle.
Flow of event	Clerk selects a vehicle. Clerk selects a client. Clerk fills date for the beginning of reservation. Clerk fills date for due date of the return. Clerk clicks on submit.

	If the vehicle is available for specified date range, a success message is shown to the client.
Exception Condition	If vehicle is not available for specified date range, a failure/error message is shown to clerk saying that "vehicle is not available for specified date range."

- Add Client

Use case name	Add client to catalog.
Scenario	A new client is added to the system by the clerk.
Triggering event	Clerk fill the details of client and click submit button.
Brief description	Clerk fills the details of the client to the system.
Actor	Clerk
Related use case	None
Pre-condition	Clerk is logged in to the system.
Post-condition	Client details is added to the system.
Flow of event	Clerk fill the details of client. Clerk clicks submit button. System check if same user exist based on driving license number. If client is new, it is added to the system.
Exception Condition	If client already exists, the System shows the error saying "Client with same driving licence number already exists."

- Return vehicle

Use case name	Return vehicle
Scenario	Clerk returns a vehicle for a particular client which was lent out
Triggering event	Clerk click on Return vehicle button.
Brief description	Clerk search for reservation which needs to be closed by completing return of a vehicle.
Actor	Clerk
Related use case	Make Reservation
Pre-condition	Client is logged and vehicle is lent out.
Post-condition	Vehicle is returned and available for another possible reservation.
Flow of event	Clerk selects a particular reservation which needs to be closed. Clerk clicks on return car button. System shows that operation has been performed successfully.
Exception Condition	None

Clerk Secondary Tasks

- Search vehicle using various filter for query

Use case name	Search Vehicle Catalog

Scenario	Clerk search for vehicles by filtering them using several parameters.
Triggering event	Clerk Selects filters and fill needed data for filtering and click on submit.
Brief description	Clerk fills the attribute value to filter vehicle catalog and click submit.
Actor	Clerk
Related use case	None
Pre-condition	Clerk is logged in.
Post-condition	System returns a page with filtered data.
Flow of event	Clerk puts attribute value in fields. Clerk clicks on submit button. System return a page with filtered data based on provided attribute and attribute value.
Exception Condition	None

- Search reservation using various filter for query

Use case name	Search Reservation Catalog
Scenario	Clerk search for reservation by filtering them using several parameters.
Triggering event	Clerk Selects filters and fill needed data for filtering and click on submit.
Brief description	Clerk fills the attribute value to filter reservation catalog and click submit.
Actor	Clerk

Related use case	None
Pre-condition	Clerk is logged in.
Post-condition	System returns a page with filtered data.
Flow of event	Clerk puts attribute value in fields. Clerk clicks on submit button. System return a page with filtered data based on provided attribute and attribute value.
Exception Condition	None

- View details of vehicle with additional details of its availabilities

Use case name	View Details of vehicle
Scenario	Clerk click on vehicle item to see its details.
Triggering event	Clerk clicks on item.
Brief description	Clerk clicks on vehicle item from vehicle catalog in order to see its details.
Actor	Clerk
Related use case	Clerk Search the vehicle catalog.
Pre-condition	Clerk is logged in.
Post-condition	Clerk gets a detail view of the system.
Flow of event	Clerk gets a vehicle catalog. Clerk clicks on one of the items. Clerk gets a detailed view of the item.
Exception Condition	None

3.2. Non-Functional Requirements

Achieving non-functional requirements must be considered throughout the design, implementation, and deployment. Architecturally relevant non-functional requirements can be characterized into mainly three types namely qualities of the system, business qualities and qualities that are about the architecture itself. These are largely dependent on architectural decisions and all design involves tradeoffs in system qualities.

Source : Software Requirement Specification

Name : Non-functional Requirements

Architectural Relevance : Architecture is critical to the realization of many qualities of interest in a system, and these qualities should be designed in and can be evaluated at the architectural level. Architecture, by itself, is unable to achieve qualities. It provides the foundation for achieving quality, but this foundation will be to no avail if attention is not paid to the details.

Addressed in : All the architectural relevance qualities are addressed in Section 10 in this document.

- **System Qualities**

  Functionality
  The ability of the system to do the work for which it was intended. This system shall provide all the functionalities such as view and search vehicle catalog, manage vehicle record, manage reservation, filtering by date etc.

  Performance
  The response time, utilization and throughput behaviour of the system. It depends on how much communication and interaction is required between components of the systems.

  Security
  It is a measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users. Our system shall provide confidentiality by making data accessible only to those authorized who have access through password privacy and prevent unauthorized access and modification of data.

  Usability
  It is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified

context of use. This system shall be easy to use for input preparation, operation, and interpretation of output and shall provide consistent user interface standards or conventions with our other frequently used systems.

Modifiability
It affects the architecture very much. The ease with which a software system can accommodate changes to its software. This system shall support modularization by which it is to support changes without affecting other modules.

Reusability
The degree to which existing applications can be reused in new applications. It is related to architecture since the reusable elements are components. Other reusable elements can be considered tactics, documentation, trade-off analysis etc.

Testability
The ease with which system can be made to demonstrate its faults. It depends on architectural issues such as documentation, separation of concerns, encapsulation etc.

- **Business Qualities**

Cost and Schedule
The cost of the system with respect to time to market, expected project lifetime, and utilization of legacy and COTS systems.

Marketability
The use of the system with respect to market competition.

Appropriateness for Organization
Availability of the human input, allocation of expertise and alignment of team and software structure.

- **Architectural Qualities**

Conceptual Integrity
The integrity of the overall structure that is composed of a number of small architectural structures. It is the most important consideration in system design.

Correctness and completeness are essential for the architecture to allow for all of the system's requirements and runtime resource constraints to be met. A formal evaluation is the architect's best hope for a correct and complete architecture.

Buildability allows the system to be completed by the available team in a timely manner and to be open to certain changes as development progresses. To have a common repository for all project files available and updated remotely, distributed version control system(Git) is maintained.

4. Logical view

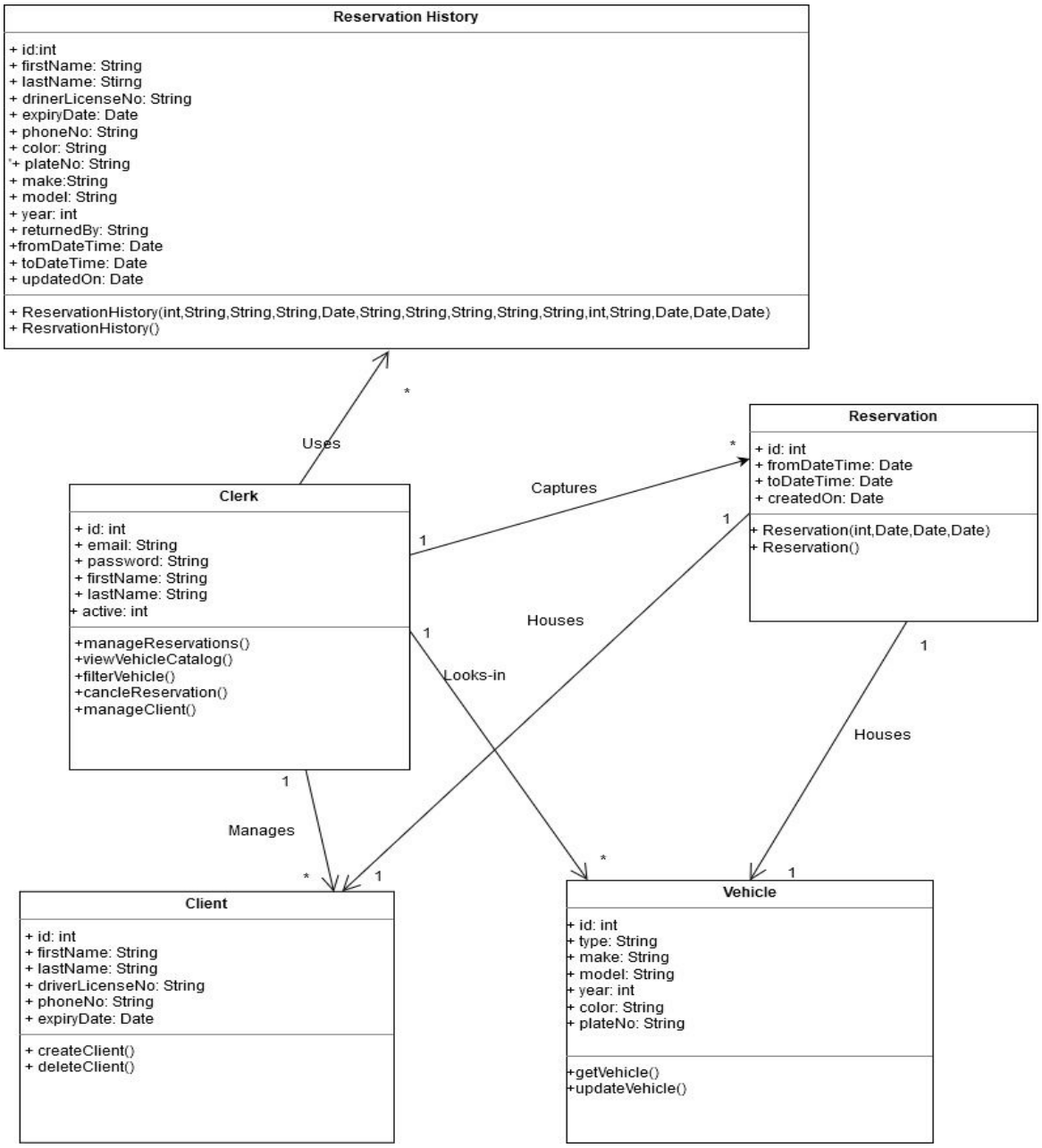

Figure 3: Class Diagram

5. Activity Diagram

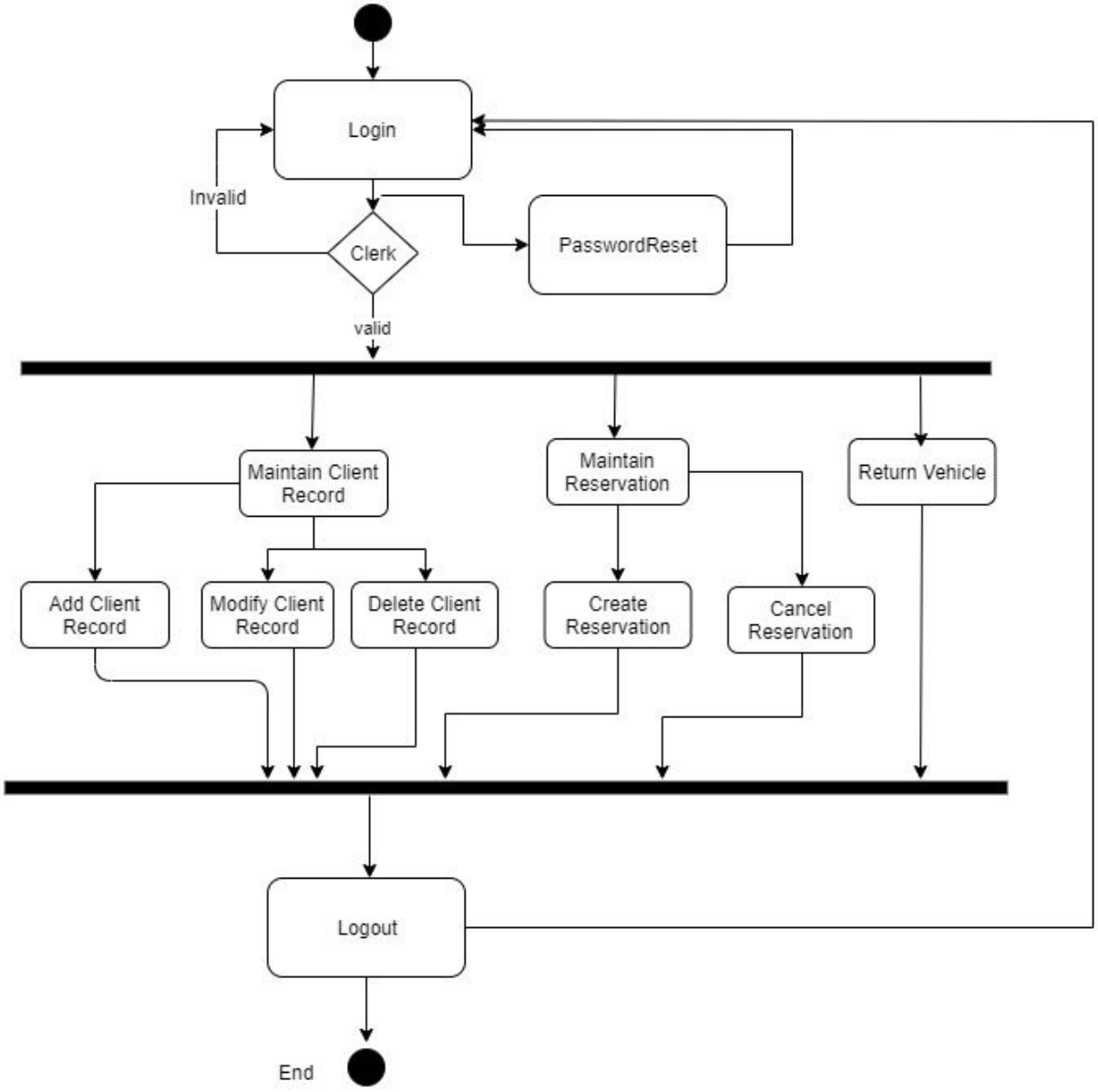

Figure 4(a): Activity Diagram

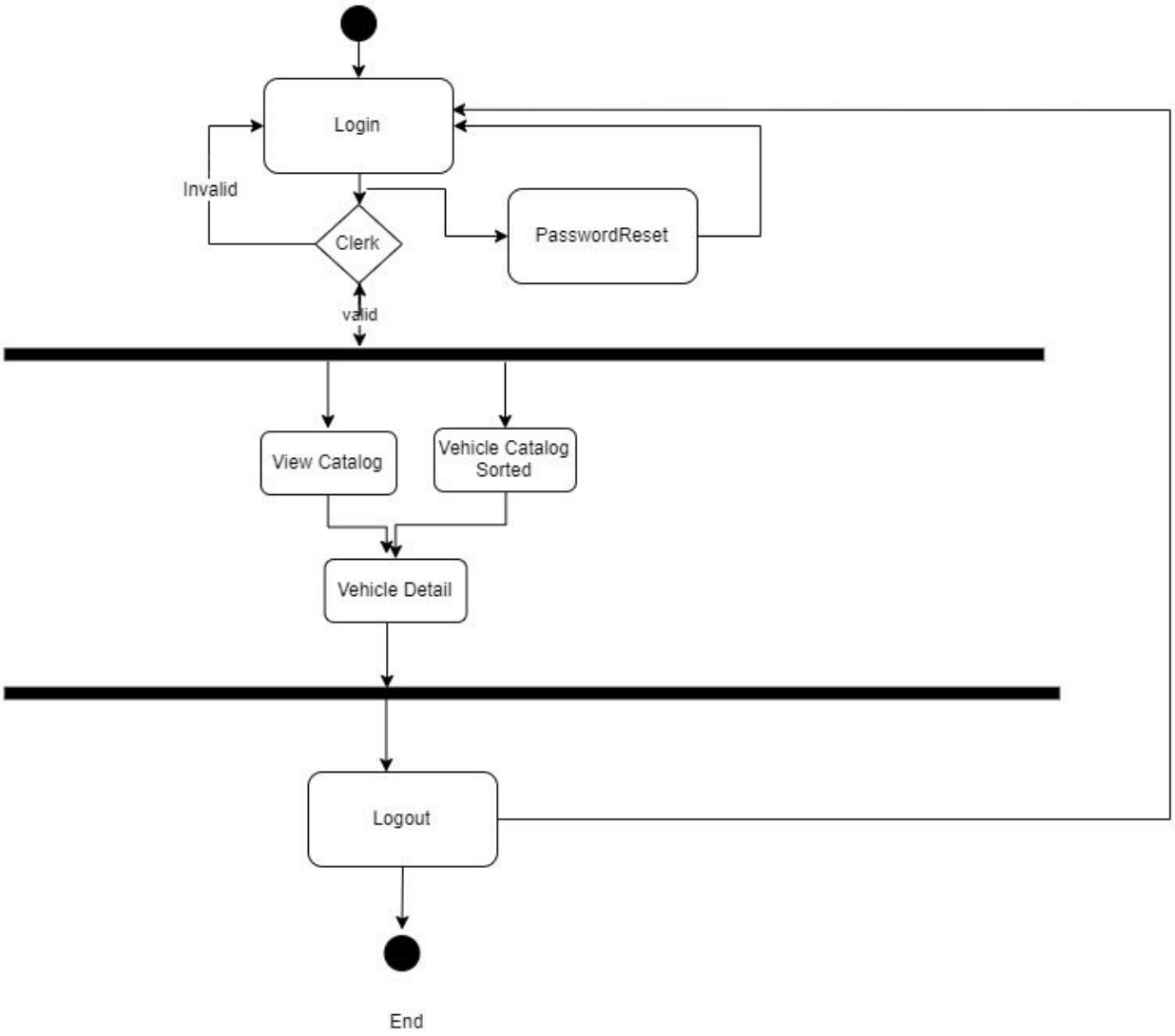

Figure 4(b): Activity Diagram

6. Development (Implementation) view

The purpose of implementation view is to identify the architectural decisions taken during the development phase. Here, package diagrams are used to realize the development view.

Clerk Functionality Development View

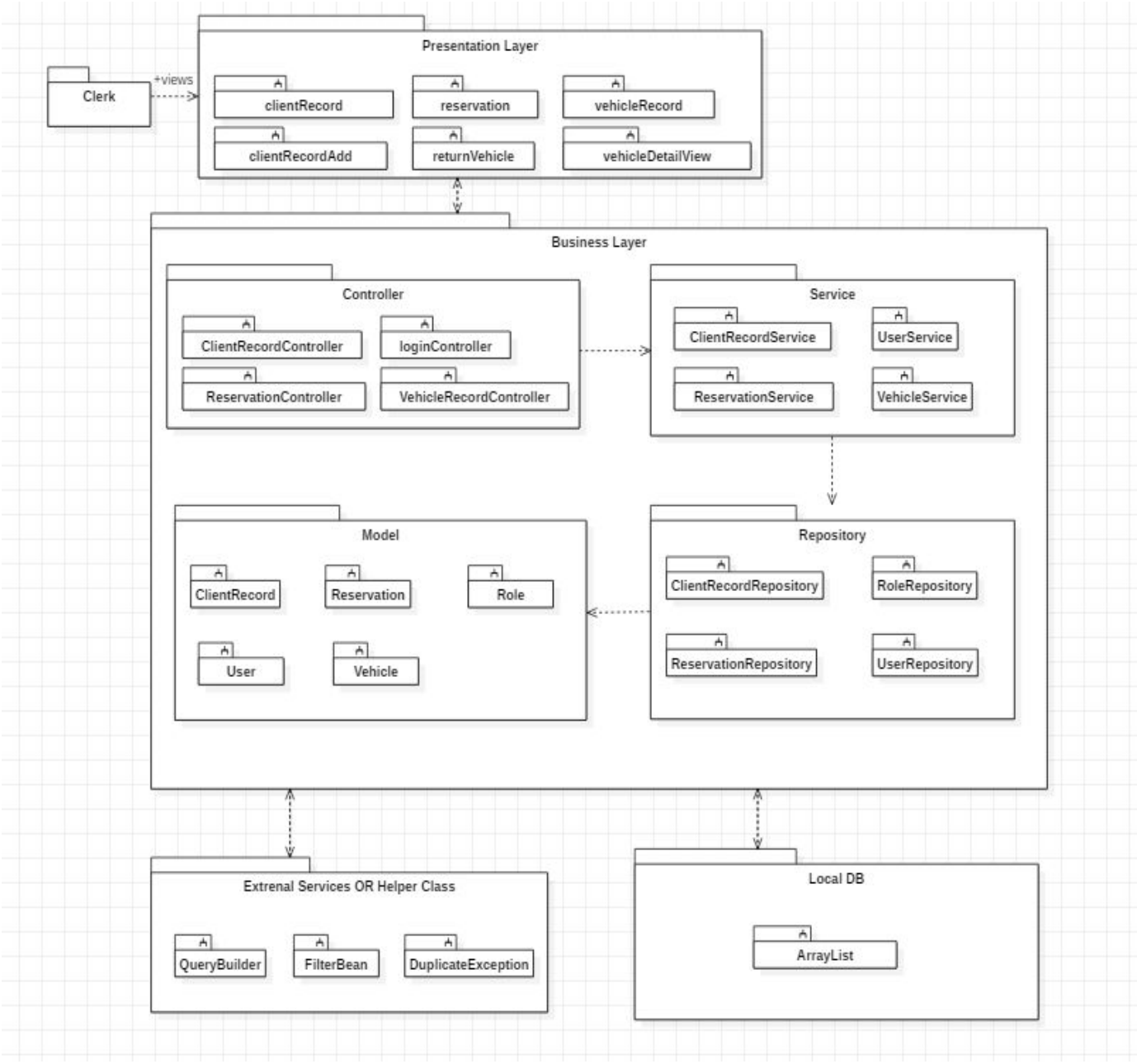

Figure 5: Package Diagram

7. Process view

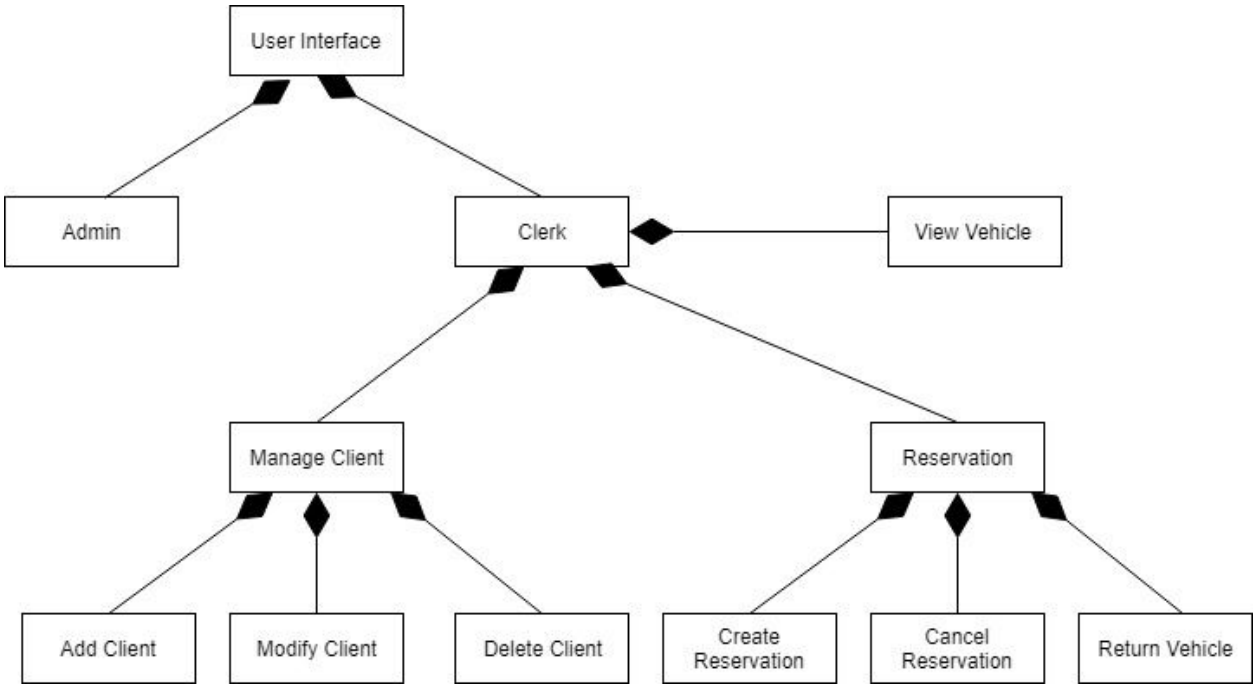

Figure 6: Process View

8. Deployment (Physical) view

The web application will be hosted on a single physical server. An Apache Tomcat web server running on Java 8 will be used to serve the application pages. In addition, a MySql Server instance will also be hosted on the physical server to aid the application in persisting data.

The application's deployment specifics can be seen below.

Node	Type	Description
Device - Linux 1 GB Server	Device	An instance of Linux Virtual machine or VPS running Linux distro with 1 GB RAM

Tomcat 8	Environment	An environment for hosting web application developed in Java.
Catalina Servlet Container	Environment	Catalina is Tomcat's servlet container. Catalina implements Sun Microsystems's specifications for servlet and JavaServer Pages (JSP). In Tomcat, a Realm element represents a "database" of usernames, passwords, and roles (similar to Unix groups) assigned to those users.
MYSQL Server	Device	MySQL is an open-source relational database management system (RDBMS).

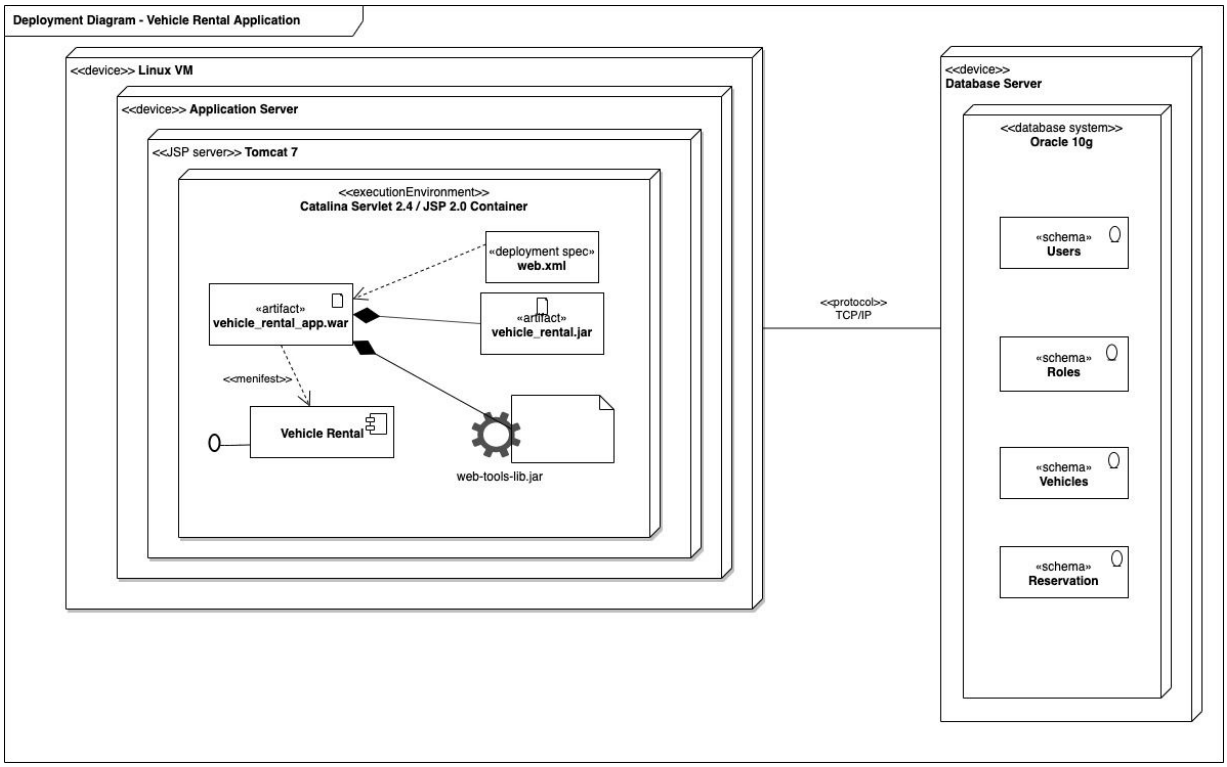

Figure 7: Deployment Diagram

9. Data view

Data modeling is a common activity in the software development process of information systems, which typically use database management systems to store information. The output of this activity is the data model, which describes the static information structure in terms of data entities and their relationships. This structure is often represented graphically in entity-relationship diagrams (ERDs) or UML class diagrams. Architectural views found in software architecture documents usually concentrate on describing the organization and dependencies among implementation/functional units, structure and interaction of runtime elements, the hardware infrastructure, and the correspondence among all of these. The data model of a system is also an architectural view. In fact, some multi-view approaches for architecture documentation geared towards information systems prescribe a data view, data architecture view, or information viewpoint that embodies the data model. The data model for a given system contains important architectural information and serves the following practical purposes:

- Provide a conceptual description of the objects (e.g., Client, Catalog) in the system's domain and their relationships.
- Provide a blueprint for creating the database structure.
- Guide implementation of code units that access the database.
- Guide performance enhancements in data access operations.
- Serve as input for automatic generation of database schema and data access code.
- Facilitate stakeholder communication in domain analysis and requirements elicitation tasks.

10. Quality

Security: Vehicle renting system implements user authentication by providing access only to authorized users. Communication between client and server is securely handled by the HTTP protocol.

Scalability: The web application will receive large volumes of client request. With the catalina servlet container and apache tomcat server. All the requests are handled appropriately and responses are sent back to the client without any delay.

Availability: The system under development is highly reliable. When the tomcat server is fired up, it automatically connects with the database and fetches the required information. The connection between database and application will be maintained until the web browser is closed or logged out from the application. Hence the application is highly available.

Portability: Our web application uses tomcat server and java to implement the server side business logic and control flow of the application. Both of them are open source and widely used across different platforms. Hence, migrating the application from platform to another won't be difficult and requires minimal work.

Analyzability: The ability to assess the impact on application due to the changes made in other parts of application, to recover software where changes caused failures and to discern if the change is feasible. Our application is designed in such a way that, the system is adaptable to future valid changes. With the help of simple coding techniques and comments in appropriate places to better understand the code, we achieved analyzability.

Modularity: Our system is divided into atomic components, where each component/ package functions on its own independent of other packages. The change in one package will have minor impact on other packages. We achieved modularity by having high cohesion and low coupling between the modules in the application.

Liveness: One of the interesting features in our application is the live nature of executable application. Once the application is hosted in server, and if we make local changes in

the system and refresh the web page in a browser, all the new changes can be viewed without restarting the application or server (feature of spring application).

Privacy: In this application, we implemented the local user authentication for the first iteration. Database based user authentication (highly secure compared to local authentication) will be implemented in upcoming releases of project.

Non-repudiability: Non-repudiability means that the developed system works as per requirement specifications and operation contract plotted during the design phase. Each operation contract is implemented separately to ensure the non- repudiability quality attribute in our application.

Reusability: The ability of software application to reuse a functionality multiple times across the application. For eg. We used a single instance of filter bean class to filter the vehicle resultset applied for the fields namely make, type ,year ,model and color respectively.