

Secure and Scalable EV Data Protection using RBAC and Identity-Based Cryptography

A PROJECT REPORT

Submitted by

MOHANAPRIYA S (510121104025)

SHRUTHI S (510121104038)

SUJITHA S (510121104044)

VARALAKSHMI R (510121104049)

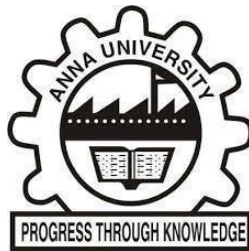
in partial fulfillment for the award the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

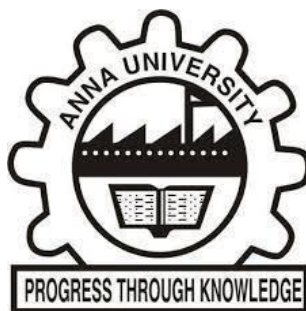


**ADHIPARASAKTHI COLLEGE OF ENGINEERING,
G.B NAGAR, KALAVAI.**

ANNA UNIVERSITY::CHENNAI 600 02

April 2025

ANNA UNIVERSITY::CHENNAI 600 025



BONAFIDE CERTIFICATE

Certified that this project report “**Secure and Scalable EV Data Protection using RBAC and Identity-Based Cryptography**” is the bonafide work of **MHONAPRIYA S (510121104025)** ,**SHRUTHI S(510121104038)**, **SUJITHA S (510121104044)**, **VARALAKSHMI R(510121104049)**who carried out the project work under my supervision.

SIGNATURE

Mr.B.SUKKRIVAN,M.Tech.,(Ph.D).,

HEAD OF THE DEPARTMENT,

Associate Professor,
Dept.of Computer Science and Engg,
Adhiparasakthi College of Engg.
G.BNagar,Kalavai.

SIGNATURE

Mr.G.JAYACHANDRAN,M.E.,(Ph.D).,

SUPERVISOR,

Associate Professor,
Dept.of Computer Science and Engg,
Adhiparasakthi College of Engg.
G.BNagar,Kalavai.

Submitted for the project and viva-voce held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

With the divine blessings of Goddess **Adhiparasakthi**, we express our deep gratitude to His **Holiness Arul Thiru Padma Shri Bangaru Adigalar**, Founder President and **Thirumathi Lakshmi Bangaru Adigalar**, Vice President for providing an amazing environment for the development and promotion of this under graduate education in our college under ACMEC Trust.

We are very grateful to **Sakthi Thirumathi Dr. B. Umadevi**, Correspondent, APCE for her encouragement and inspiration. We are very grateful to **Sakthi Thiru R. Karunanidhi**, Secretary for his continuous support. We are highly indebted to our Principal **Prof. Dr. MohanaMurugan** for his valuable guidance.

We wish to place our sincere gratitude to **Prof. Mr. B. Sukkrivan**, Head of the Department of computer science and engineering for his motivation and permitting us to do this work.

We are especially indebted to our Supervisor **Mr. G.JAYACHANDRAN**, Associate Professor , Department of Computer science and engineering for his valuable advice, sustained interest help extended towards us for the completion of this work successfully. We are thankful to all teaching and non-teaching staff of our department for their constant cooperation and encouragement in pursuing our project work.

ABSTRACT

Role-based Access Control (RBAC) offers an efficient method for managing authorization to resources, such as electric vehicle (EV) data stored in the cloud. In this context, the access control system must ensure both strong security and efficiency, particularly with regard to EV user authentication. Authentication serves as a crucial prerequisite for access control, ensuring only legitimate users can access sensitive EV data. To enhance the security and scalability of the system, this work integrates identity-based cryptography (IBC) with RBAC. IBC is used to securely authenticate EV users, ensuring that their identity is tied to their role in the system. Each EV user's authentication is coupled with a unique identity, which is also used in the key exchange process. This combination of RBAC and IBC ensures that access to the cloud-based EV data is tightly controlled. By linking users' roles with their identities, the system can effectively manage access permissions based on both their role and verified identity, enhancing security and streamlining the management of user access. This approach offers a robust and scalable solution for EV data protection.

TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE
	ABSTRACT	i
	LIST OF FIGURES	v
	LIST OF SYMBOLS	vi
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 About the Project	1
2	SYSTEM ANALYSIS	2
	2.1 Existing system	2
	2.1.1 Problem Definition	2
	2.2 Proposed system	2
	2.2.1 Advantage	3
3	REQUIREMENTS SPECIFICATION	4
	3.1 Introduction	4
	3.2 Hardware and Software specification	5
	3.3 Technologies Used	5
	3.3.1 Java	8

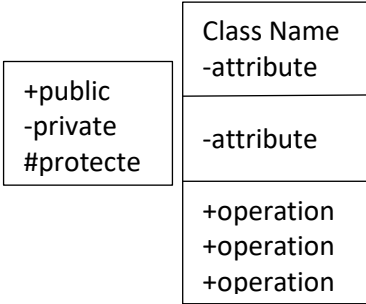
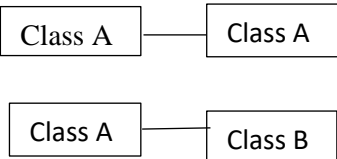
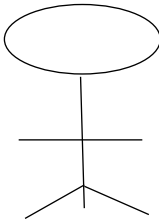
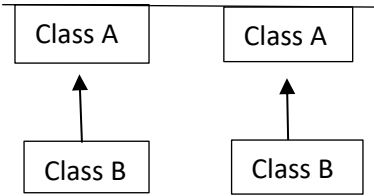
	3.3.1.1 Introduction to Java	9
	3.3.1.2 Working of Java	10
4	Project Purpose and Scope	21
	4.1 Design and Implementation Constraints	21
	4.2 Other Nonfunctional Requirements	22
5	SYSTEM DESIGN	24
	5.1 System Design	24
	5.2 Sequence Diagram	25
	5.3 Use Case Diagram	26
	5.4 Activity Diagram	27
	5.5 Collaboration Diagram	28
	5.6 Data Flow Diagram	29
	5.7 Class Diagram	30
6	SYSTEM DESIGN – DETAILED	31
	6.1 Modules	31
	6.2 Module explanation	31
7	CODING AND TESTING	33
	7.1 Coding	33

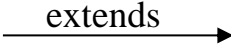



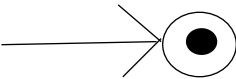
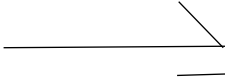
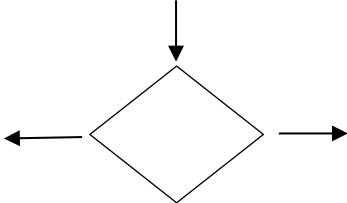
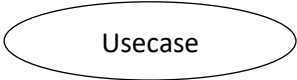
7.2 Coding standards	33
7.3 Test procedure	35
7.4 Test data and output	36
APPENDIES	45
SCREEN SHOT	64
REFERENCES	71

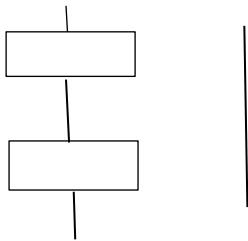
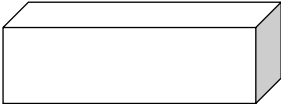
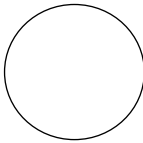

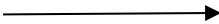

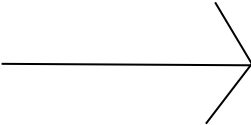
LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
5.1	Architecture Diagram	24
5.2	Sequence diagram	25
5.3	Use case Diagram	26
5.4	Activity Diagram	27
5.5	Collaboration diagram	28
5.6	Data Flow Diagram	29
5.7	Class diagram	30

LIST OF SYMBOLS

SNO	NOTATION NAME	NOTATION	DESCRIPTION
1.	Class		Represents a collection of similar entities grouped together.
2.	Association		Associations represents static relationships between classes. Roles represents the way the two classes see each other
3.	Actor		It aggregates several classes into a single classes
4.	Aggregation		Interaction between the system and external environment
5.	Relation (Uses)	Uses	Used for additional process communication

6.	Relation(extends)		Extends relationship is used when one use case is similar to another use case but does a bit more.
7.	Communication		Communication between various use cases.
8.	State		State of the processs
9.	Initial State		Initial state of the object
10.	Final State		Final state of the object
11.	Control Flow		Represents various control flow between the states.
12.	Decision Box		Represents decision making process from a constraint
13.	UseCsae		Interact ion between the system and external environment.

14.	Component		Represents physical modules which is a collection of components
15.	Node		Represents physical modules which are a collection of components
16.	Data Process/State		A circle in DFD represents a state or process which has been triggered due to some event or action.
17.	External entity		Represents external entities such as keyboard, sensors, etc.
18.	Transition		Represents communication that occurs between processes
19.	Object Lifeline		Represents the vertical dimensions that the object communications.
20.	Message		Represents the message exchanged.

LIST OF ABBREVIATIONS

JDK	Java Development Toolkit
DEX	Dalvik Executables
TCP	Transmission Control Protocol
IP	Internet Protocol
HTTP	Hyper Text Transfer Protocol
ADT	Android Development Tool

CHAPTER 1

INTRODUCTION

Aim:

The aim of this work is to develop a robust and efficient authorization management system for accessing electric vehicle (EV) data stored in a cloud server by integrating Role-based Access Control (RBAC) with identity-based cryptography (IBC).

Synopsis:

Role-based Access Control (RBAC) promises an efficient authorization management system in accessing resources including electric vehicle (EV) data stored in the cloud server. In this EV data security implementation, access control has to be strong and efficient with respect to EV user authentication information, thus access control mechanism mandatorily relies on authentication as the system access prerequisite. In this work, identity-based cryptography (IBC) is incorporated with RBAC to invent an EV user role-based access control immersed in his/her identity as an internal EV user's authentication and key exchange. Contribution to our work is in a simple way involving only the EV user's signature to verify simultaneously both important aspects of authentication and authorization. In this case, authentication is carried out based on identity while authorization is activated based on EV user's role. We formally prove that the proposed protocol satisfies the security requirements of both authentication and authorization outright by verifying EV user's signature. The evaluation results show that the total computational cost for authentication and key exchange process between EV user and the server is practical enough and it only consumes approximately 800 ms.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

The existing system for managing access to electric vehicle (EV) data primarily utilizes Role-based Access Control (RBAC), which assigns permissions based on user roles established by the National Institute of Standards and Technology (NIST). While RBAC provides a structured framework for access management, it heavily relies on authentication as a prerequisite for accessing resources. This system allows for dynamic role assignments during user sessions, but often suffers from complexity and inefficiency in authentication processes, which can lead to delays and user frustration. Additionally, traditional RBAC lacks advanced mechanisms for robust identity verification, making it vulnerable to unauthorized access. Although there are efforts to minimize computational costs, existing systems may still face challenges in real-time responsiveness. Overall, while effective in its basic form, the existing RBAC system requires enhancements to better secure EV data and improve user experience.

2.1.1 Problem definition

As a result, offloading actions taken by the mobile user may fail. However, as stated in intermittent connectivity issues may exist due to mobility, heterogeneous network environment and smart devices connection policies. A non-persistent connection is a key feature distinguishing mobile cloud with conventional cloud systems. Energy consumption and performance is low.

2.2 PROPOSED SYSTEM

The proposed system enhances the management of electric vehicle (EV) data by incorporating a secure access control mechanism that integrates Role-based Access

Control (RBAC) with identity-based cryptography (IBC). In this system, a unique private key is generated for each owner and driver, which is securely sent to their registered email addresses. This private key serves as a vital tool for authentication, ensuring that only authorized users can access sensitive vehicle information.

Owners have the ability to add vehicles to their profiles seamlessly, providing an intuitive management interface. They can also view a comprehensive list of associated drivers, enhancing their oversight and control. When owners wish to access detailed vehicle data, an additional layer of security is implemented through One-Time Password (OTP) authorization. This ensures that sensitive information is only available after verifying the owner's identity with an OTP, adding a crucial layer of protection.

Drivers also benefit from the system, as they can utilize their private keys for authentication while being granted specific access based on their roles. By leveraging IBC and OTP for secure access, the proposed system not only simplifies vehicle management but also ensures that all data remains protected, providing a secure and efficient platform for both owners and drivers to manage their electric vehicles confidently.

2.2.2 Advantage

As a result, offloading actions taken by the mobile user may fail. However, as stated in intermittent connectivity issues may exist due to mobility, heterogeneous network environment and smart devices connection policies. A non-persistent connection is a key feature distinguishing mobile cloud with conventional cloud systems. Energy consumption and performance is low.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 INTRODUCTION

There exists an effective access control algorithm to protect either information or any other resources in small-scale, medium-scale, large-scale and even enterprise systems called RBAC [1]. National Institute of Standard and Technology (NIST) proposed the model as a widespread RBAC standard [2]. This RBAC standard associated permissions or access rights into the roles of a user which means that particular roles designated to the corresponding access rights are acquired to a user. Regarding [1], the access right of roles can be determined and enabled during user's session and can be given to the user when it is necessary. The administrator of RBAC system has responsibility to manage authorization mechanism to access information or resources. Therefore, access control has to be strong and efficient with respect to user authentication information, thus access control mechanism mandatorily relies on authentication as the system access prerequisite. On the other hand, lightening the complexity of both computational cost and access control scenario is considered as well. Many authentication systems have already considered computational costs when applied in the mobile communication and networking environment ensure the security of keys used by both communicating parties. Currently, public key infrastructure (PKI) [22] with its users' public key certificates management support has been widely deployed in the users' authentication system. It sets up, arranges, and manages the system as well. However, this system still remains many problems of cost expanding and pitiful management when managing users' public key certificates. There exists an effective alternative authentication system using identity-based signature (IBS) which has been widely applied.

3.2 HARDWARE AND SOFTWARE SPECIFICATION

3.2.1 HARDWARE REQUIREMENTS

Hard Disk : 250GB and Above
RAM : 6GB and Above
Processor : i3 and Above

3.2.2 SOFTWARE REQUIREMENTS

- Windows 10 and above
- JDK 11.0
- J2EE
- MySQL8

3.3 Technology Stack:

- **Java**
- **Spring Boot Framework**
- **Thymeleaf Framework**
- **Html**
- **Css**
- **Javascript**
- **Jquery**

Modules:

- **Admin Module**
- **Owner Module**
- **Driver Module**

Admin Module:

The Admin Management Module serves as a vital component of the electric vehicle (EV) data management system, enabling administrators to efficiently oversee and manage user and vehicle information. Upon secure login, administrators gain access to functionalities that allow them to add vehicle details directly to the cloud, ensuring that all information is centralized and easily retrievable. This module also includes tools for verifying the identities of vehicle owners and drivers, allowing administrators to confirm that individuals are authorized to access specific data. By managing role assignments and permissions, administrators can ensure that users have appropriate access levels based on their responsibilities. Overall, the Admin Management Module streamlines administrative tasks, enhances security measures, and provides a comprehensive overview of the EV data ecosystem, fostering a reliable and organized management environment.

Owner Module:

The Owner Module is a key feature of the electric vehicle (EV) data management system, designed to provide vehicle owners with seamless access to their vehicle information and driver details. Using a unique private key sent to their registered email, owners can securely access and view their vehicle data, ensuring that sensitive information remains protected. This module allows owners to easily see detailed information about their vehicles, specifications. Additionally, owners can view the details of drivers associated with their vehicles, enhancing their oversight and management capabilities. Furthermore, the module enables owners to add new vehicles to their profiles effortlessly. With a user-friendly interface, they can input essential vehicle information, ensuring that their records remain current and comprehensive.

Overall, the Owner Management Module empowers vehicle owners with the tools they need to effectively manage their assets while maintaining robust security measures.

Driver Module:

The Driver Management Module is an essential part of the electric vehicle (EV) data management system, designed to facilitate the registration, login, and data access for drivers. Upon registration, drivers can create their profiles by providing necessary personal information, ensuring that their identities are verified within the system. Once registered, drivers can securely log in using their credentials to access the module's features. They are allowed to view specific electric vehicle (EV) data.

Future Enhancement:

The future enhancements of the electric vehicle (EV) data management system will focus on further improving security, usability, and overall efficiency. Building on the proposed system that integrates identity-based cryptography (IBC) and One-Time Password (OTP) authorization, additional features will be introduced to enhance user experience and streamline access control.

One potential enhancement is the implementation of biometric authentication methods, such as fingerprint or facial recognition, which would provide an extra layer of security for both owners and drivers. This could further simplify the login process while ensuring that only authorized individuals can access sensitive EV data.

Another enhancement could involve the development of a mobile application that allows for real-time notifications and updates regarding vehicle status and driver activities. This would enable users to stay informed and engaged with their vehicles, improving the overall management experience.

Conclusion:

In summary, the existing system for managing access to electric vehicle (EV) data, primarily based on Role-based Access Control (RBAC), provides a foundational framework for access management. However, it faces significant challenges, including

complex authentication processes, inefficiencies, and vulnerabilities in identity verification. These limitations can lead to delays, user frustration, and potential unauthorized access, underscoring the need for enhancements to ensure the security and usability of EV data management.

The proposed system addresses these shortcomings by integrating identity-based cryptography (IBC) and implementing One-Time Password (OTP) authorization. By sending unique private keys to owners and drivers via email, the system improves the authentication process and facilitates secure access to vehicle data. Additionally, features such as the ability for owners to easily add vehicles and view driver lists enhance user experience while maintaining stringent security protocols.

Ultimately, the proposed system not only strengthens the security framework for managing EV data but also streamlines operations for both owners and drivers. This integrated approach marks a significant advancement over the existing RBAC model, ensuring that access control is both robust and user-friendly, thereby fostering a more secure environment for all stakeholders involved in the management of electric vehicles.

3.3.1 JAVA

Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C.

3.3.1.1 INTRODUCTION TO JAVA

Java has been around since 1991, developed by a small team of Sun Microsystems developers in a project originally called the Green project. The intent of the project was to develop a platform-independent software technology that would be used in the consumer electronics industry. The language that the team created was originally called Oak. The first implementation of Oak was in a PDA-type device called Star Seven (*7) that consisted of the Oak language, an operating system called GreenOS, a user interface, and hardware. The name *7 was derived from the telephone sequence that was used in the team's office and that was dialed in order to answer any ringing telephone from any other phone in the office.

Around the time the First Person project was floundering in consumer electronics, a new craze was gaining momentum in America; the craze was called "Web surfing." The World Wide Web, a name applied to the Internet's millions of linked HTML documents was suddenly becoming popular for use by the masses. The reason for this was the introduction of a graphical Web browser called Mosaic, developed by ncSA. The browser simplified Web browsing by combining text and graphics into a single interface to eliminate the need for users to learn many confusing UNIX and DOS commands. Navigating around the Web was much easier using Mosaic.

It has only been since 1994 that Oak technology has been applied to the Web. In 1994, two Sun developers created the first version of Hot Java, and then called Web Runner, which is a graphical browser for the Web that exists today. The browser was

coded entirely in the Oak language, by this time called Java. Soon after, the Java compiler was rewritten in the Java language from its original C code, thus proving that Java could be used effectively as an application language. Sun introduced Java in May 1995 at the Sun World 95 convention. Web surfing has become an enormously popular practice among millions of computer users. Until Java, however, the content of information on the Internet has been a bland series of HTML documents. Web users are hungry for applications that are interactive, that users can execute no matter what hardware or software platform they are using, and that travel across heterogeneous networks and do not spread viruses to their computers. Java can create such applications.

3.3.1.2 WORKING OF JAVA

For those who are new to object-oriented programming, the concept of a class will be new to you. Simplistically, a class is the definition for a segment of code that can contain both data (called attributes) and functions (called methods).

When the interpreter executes a class, it looks for a particular method by the name of **main**, which will sound familiar to C programmers. The main method is passed as a parameter an array of strings (similar to the argv [] of C), and is declared as a static method.

To output text from the program, we execute the **println** method of **System.out**, which is java's output stream. UNIX users will appreciate the theory behind such a stream, as it is actually standard output. For those who are instead used to the Wintel platform, it will write the string passed to it to the user's program.

Java consists of two things :

- Programming language
- Platform

3.3.1.3 THE JAVA PROGRAMMING LANGUAGE

Java is a high-level programming language that is all of the following:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable
- High-performance
- Multithreaded
- Dynamic

The code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

Java is unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called **Java byte codes** – the platform independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how it works:

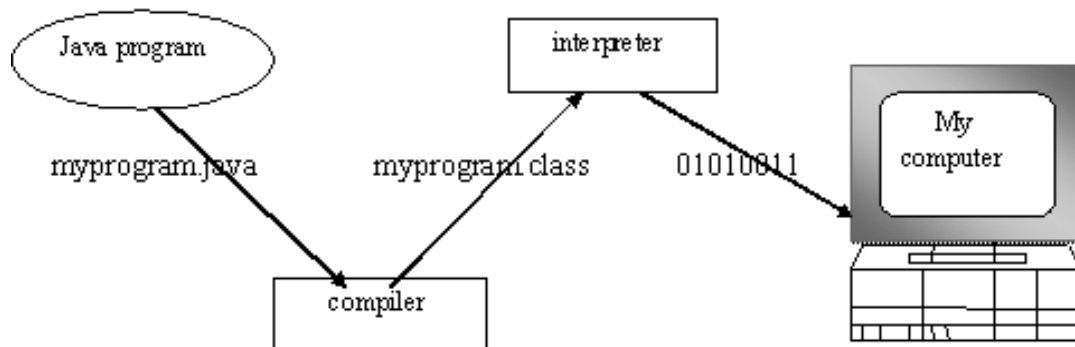


Fig.3.1

You can think of Java byte codes as the machine code instructions for the **Java Virtual Machine (JVM)**. Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of JVM. That JVM can also be implemented in hardware. Java byte codes help make “write once, run anywhere” possible.

You can compile your Java program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the JVM. For example, that same Java program can be run on Windows NT, Solaris and Macintosh.

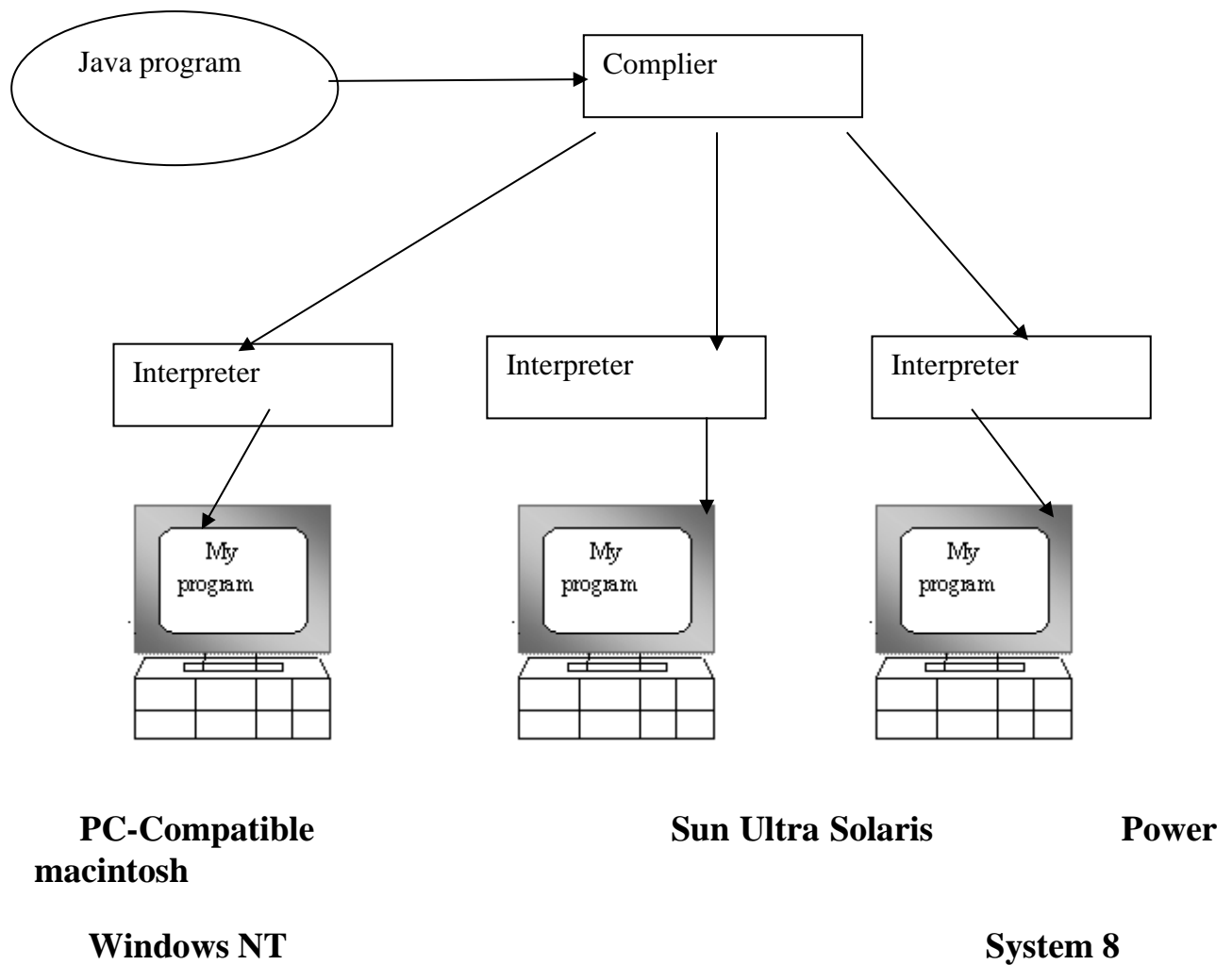


Fig 3.2.1

3.3.1.4 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components :

- The Java Virtual Machine (JVM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the JVM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (**packages**) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

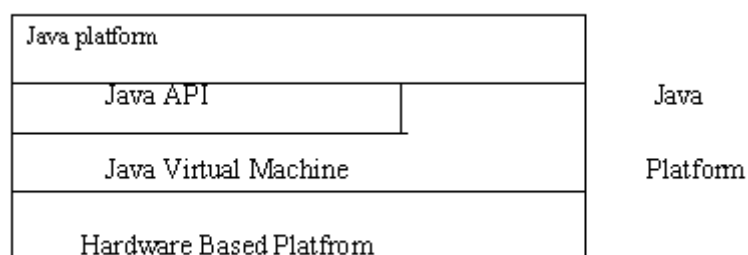


Fig.3.3

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, weel-tuned interpreters, and just-in-time byte compilers can bring Java's performance close to that of native code without threatening portability.

3.3.2 APACHE TOMCAT SERVER

Apache Tomcat (formerly under the Apache Jakarta Project; Tomcat is now a top level project) is a web container developed at the Apache Software Foundation. Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Because Tomcat includes its own HTTP server internally, it is also considered a standalone web server.

Environment

Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets. The Tomcat servlet engine is often used in combination with an Apache web server or other web servers. Tomcat can also function as an independent web server. Earlier in its development, the perception existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception no longer exists; Tomcat is increasingly used as a standalone web server in high-traffic, high-availability environments. Since its developers wrote Tomcat in Java, it runs on any operating system that has a JVM.

Product features

Tomcat 3.x (initial release)

- implements the Servlet 2.2 and JSP 1.1 specifications
- servlet reloading
- basic HTTP functionality Tomcat 4.x
- implements the Servlet 2.3 and JSP 1.2 specifications
- servlet container redesigned as Catalina
- JSP engine redesigned as Jasper
- Coyote connector
- Java Management Extensions (JMX), JSP and Struts-based administration
- Tomcat 5.x
- implements the Servlet 2.4 and JSP 2.0 specifications
- reduced garbage collection, improved performance and scalability
- native Windows and Unix wrappers for platform integration
- faster JSP parsing

History

Tomcat started off as a servlet specification implementation by James Duncan Davidson, a software architect at Sun. He later helped make the project open source and played a key role in its donation by Sun to the Apache Software Foundation.

Davidson had initially hoped that the project would become open-sourced and, since most open-source projects had O'Reilly books associated with them featuring an animal on the cover, he wanted to name the project after an animal. He came up with

Tomcat since he reasoned the animal represented something that could take care of and fend for itself. His wish to see an animal cover eventually came true when O'Reilly published their Tomcat book with a tomcat on the cover.

3.3.3 Android Introduction:

Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. The operating system has developed a lot in last 15 years starting from black and white phones to recent smart phones or mini computers. One of the most widely used mobile OS these days is android. The android is software that was founded in Palo Alto of California in 2003.

The android is a powerful operating system and it supports large number of applications in Smartphones. These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it. The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

The android development supports with the full java programming language. Even other packages that are API and JSE are not supported. The first version 1.0 of

android development kit (SDK) was released in 2008 and latest updated version is jelly bean.

The android is a operating system and is a stack of software components which is divided into five sections and four main layers that is



3.3.3.1 Linux kernel:

The android uses the powerful Linux kernel and it supports wide range of hardware drivers. The kernel is the heart of the operating system that manages input and output requests from software. This provides basic system functionalities like process management, memory management, device management like camera, keypad, display

etc the kernel handles all the things. The Linux is really good at networking and it is not necessary to interface it to the peripheral hardware. The kernel itself does not interact directly with the user but rather interacts with the shell and other programs as well as with the hardware devices on the system.

3.3.3.2 Libraries:

On top of a Linux kernel there is a set of libraries including open source web browser such as webkit, library libc. These libraries are used to play and record audio and video. The SQLite is a data base which is useful for storage and sharing of application data. The SSL libraries are responsible for internet security etc.

3.3.3.3 Android Runtime:

The android runtime provides a key component called Dalvik Virtual Machine which is a kind of java virtual machine. It is specially designed and optimized for android. The Dalvik VM is the process virtual machine in the android operating system. It is software that runs apps on android devices.

The Dalvik VM makes use of Linux core features like memory management and multithreading which is in a java language. The Dalvik VM enables every android application to run its own process. The Dalvik VM executes the files in the .dex format.

3.3.3.4 Application framework:

The application framework layer provides many higher level services to applications such as window manager, view system, package manager, resource

manager etc. The application developers are allowed to make use of these services in their application.

3.3.3.5 Applications:

You will find all the android applications at the top layer and you will write your application and install on this layer. Examples of such applications are contacts, books, browsers, services etc. Each application performs a different role in the overall applications.

Advantages:

- Android is Linux based open source operating system , it can be developed by any one
- Easy access to the android apps
- You can replace the battery and mass storage, disk drive and UDB option
- Its supports all Google services
- The operating system is able to inform you of a new SMS and Emails or latest updates.
- It supports Multitasking
- Android phone can also function as a router to share internet
- Its free to customize
- Can install a modified ROM
- Its supports 2D and 3D graphics

CHAPTER 4

4.1 Design and Implementation Constraints

4.1.1 Constraints in Analysis

- ◆ Constraints as Informal Text
- ◆ Constraints as Operational Restrictions
- ◆ Constraints Integrated in Existing Model Concepts
- ◆ Constraints as a Separate Concept
- ◆ Constraints Implied by the Model Structure

4.1.2 Constraints in Design

- ◆ Determination of the Involved Classes
- ◆ Determination of the Involved Objects
- ◆ Determination of the Involved Actions
- ◆ Determination of the Require Clauses
- ◆ Global actions and Constraint Realization

4.1.3 Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation.

A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

4.2 Other Nonfunctional Requirements

4.2.1 Performance Requirements

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

4.2.2 Safety Requirements

1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.

5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

CHAPTER 5

5.1 Architecture Diagram:

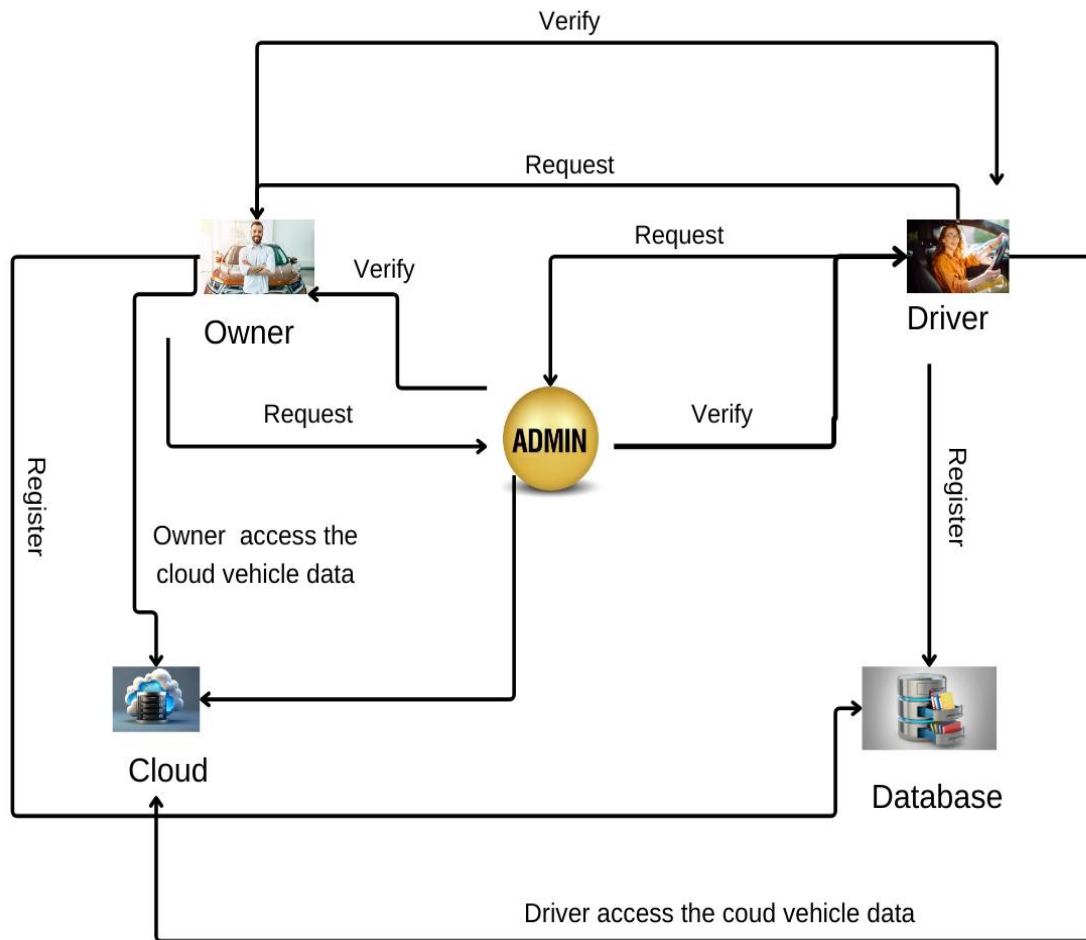


Fig: 5.1

5.2 Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

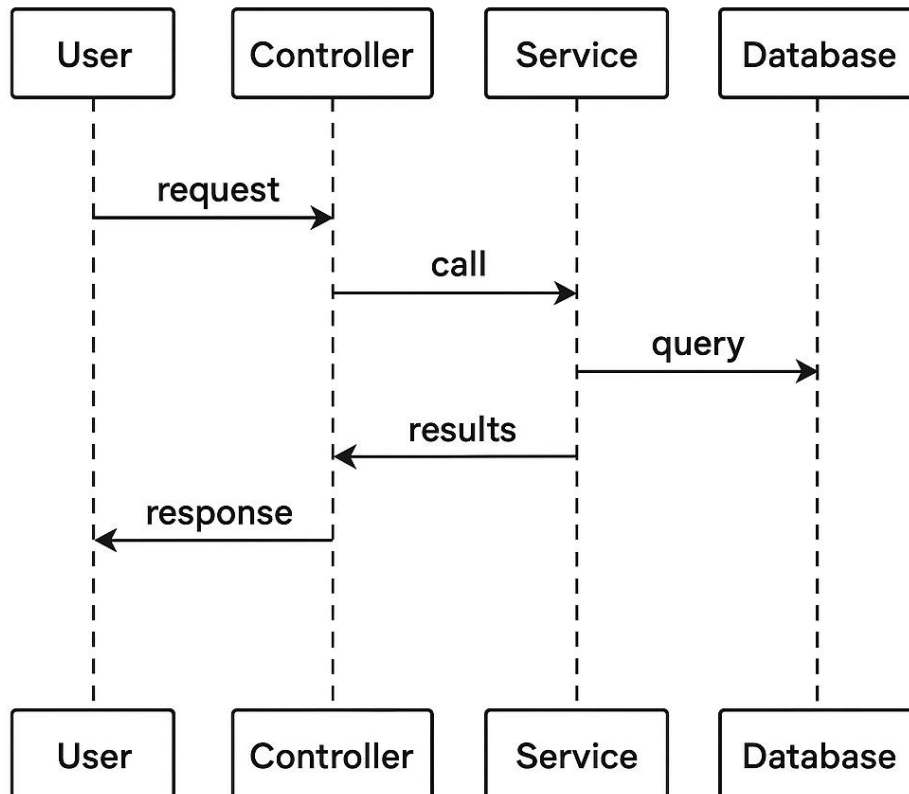


Fig 5.2

5.3 Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

5.3.1. USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.

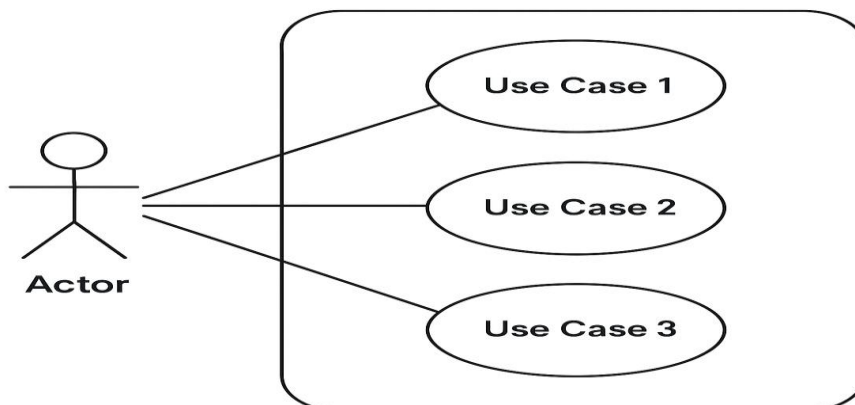


Fig 5.3

5.4 Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.

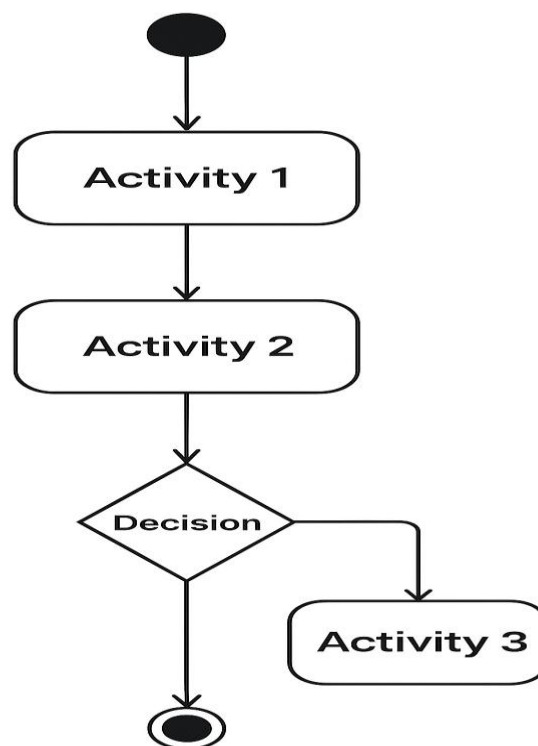


Fig 5.3

5.5 Collaboration Diagram:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.

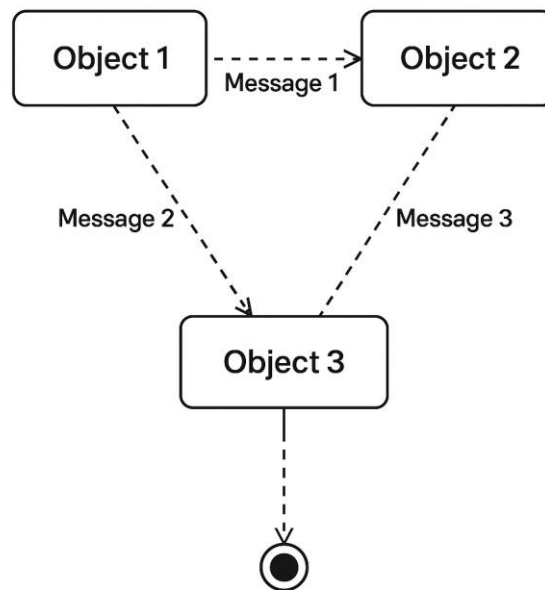


Fig 5.4

5.6 DATA FLOW DIAGRAM:

A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system, modeling its aspects. It is a preliminary step used to create an overview of the system which can later be elaborated DFDs can also be used for visualization of data processing.

Level 0:

Level 1:

Level 2:

Level 3:

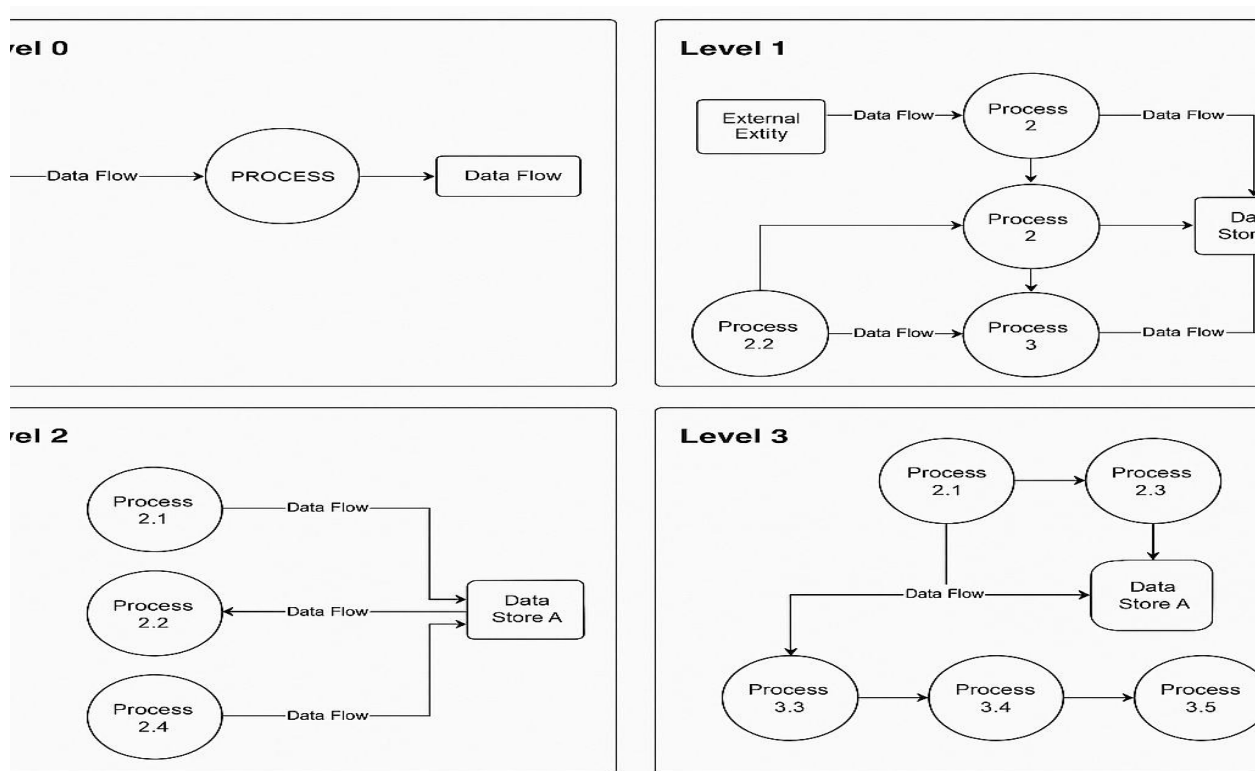


Fig 5.5

5.7 Class Diagram

A Class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

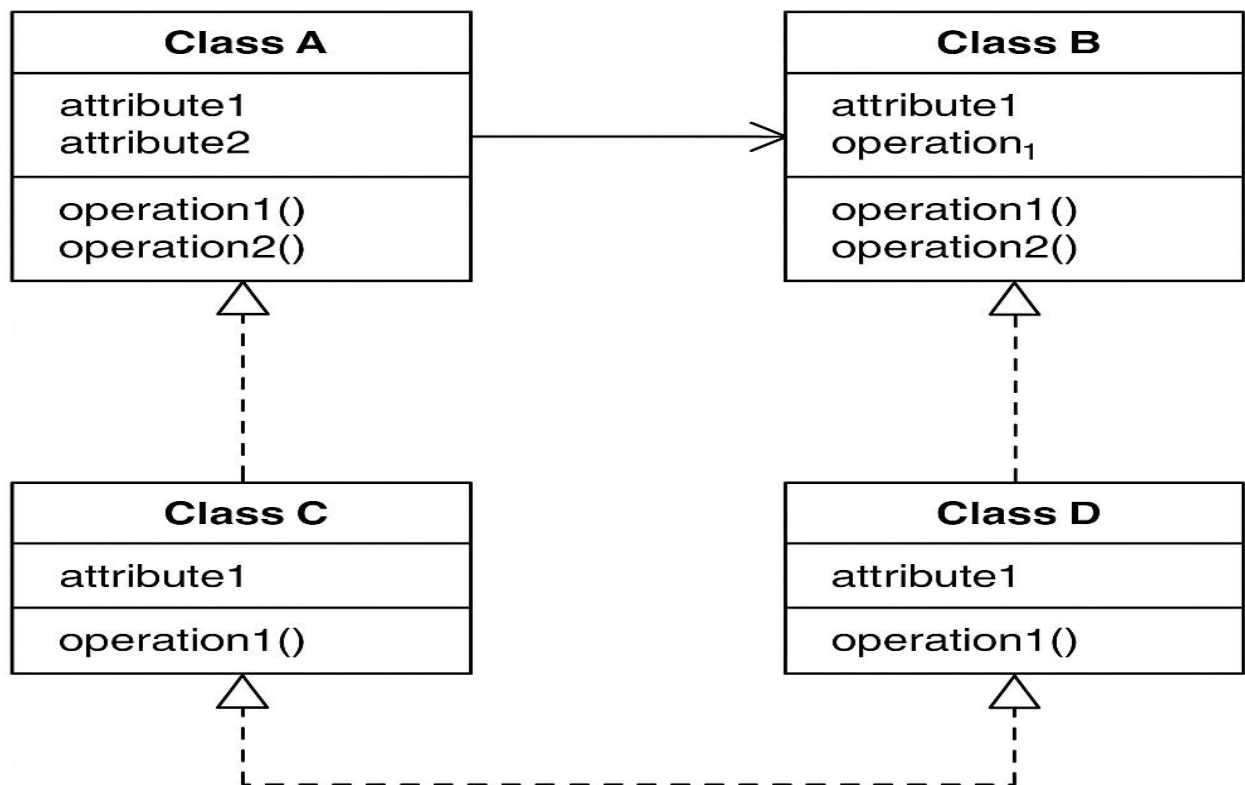


Fig 5.7

CHAPTER 6

SYSTEM DESIGN

Our ultrasound modem can be implemented on any device that has a speaker (for sending) or a microphone (for receiving) and supports a sample rate of 44.1kHz. Devices then communicate using ultrasound signals in an ad-hoc manner. Our target hardware platform is smartphones, in which these hardware requirements are ubiquitous. However, our modem can also be implemented on mobile payment stations, smart watches, IoT devices, smart home appliances, etc. The user inputs some information; the Hush software converts that information to an audio signal, and transmits it using the device speaker. A second device receives the signal and the Hush software decodes it. The information is then displayed to the user in the appropriate way.

6.1 MODULES

- **Admin Module**
- **Owner Module**
- **Driver Module**

6.2 MODULE EXPLANATION:

6.2.1 User Registration and login:

In this module user first register the Offloading Framework application and they provide user details (Name, password, email, mobile, dob) .And then login the user credential details like

username, password. Once user name and password is valid open the user profile screen will be display-

User Task Request:

In this module, user has to create a new task like Blur, crop of image operations. Those task has not been executed in the user system itself because no availability of performing that task or they willing to offloading this task. At the same time user will upload multiple task. So user will send that task to gateway (Mobile cloudlet). Gateway is the intermediate between the users. Mobility of user system and neighbors mobile has been monitored by gateway. Once task has been created by client they send it to gateway, gateway will choose cloudlets for performing that task based on the mobile user coverage. In the cloudlet to choose best mobile services based on energy, memory and performance.

Network Formation with Task Performed:

In this module Cloudlet gathering task from user, and then choose the best resource mobile users using with neighbour selection process. And rating based performance to the mobile user once got the best node cloudlet assign this task to particular node with say incentive. Intermediate node gathering task from cloudlet and start the operation like blur, crop. It takes some time to complete this task. Once finish the task to provide cloudlet .At tge same tine cloudlet will retransmit to user, and then task will reached task initiator with credit amount in your wallet.

Flop Coin Transactions:

In this final module once finish the user task and user will provide task amount to cloudlet after amount will be transfer to task performer based on performance like that incentive scheme. And then user will rating to particular mobile user based on operation. It will useful to neighbor nodes because cloudlet selection based on performance and rating system. Finally user will share our flop coins to transfer another user.

CHAPTER 7

CODING AND TESTING

7.1 CODING

Once the design aspect of the system is finalized the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screws into the system.

7.2 CODING STANDARDS

Coding standards are guidelines to programming that focus on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standards needed to achieve the above-mentioned objectives are as follows:

Program should be simple, clear and easy to understand.

Naming conventions

Value conventions

Script and comment procedure

Message box format

Exception and error handling

7.2.1 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be **self-descriptive**. One should even get the meaning and scope of the variable by its name. The conventions are adopted for **easy understanding** of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

7.2.2 VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

- Proper default values for the variables.

- Proper validation of values in the field.
- Proper documentation of flag values.

7.2.3 SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

7.2.4 MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X – User has performed illegal operation.
- ! – Information to the user.

7.3 TEST PROCEDURE

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design

itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough. Testing is one of the important steps in the software development phase.

Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

7.4 TEST DATA AND OUTPUT

7.4.1 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

7.4.2 FUNCTIONAL TESTS

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

7.4.3 PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

7.4.4 STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

7.4.5 STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.

- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

7.4.6 INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

7.5 TESTING TECHNIQUES / TESTING STRATEGIES

7.5.1 TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

7.5.1.1 WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

7.5.1.2 BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning

- Boundary value analysis
- Comparison testing

7.5.2 SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

7.5.2.1 INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on

another's sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

7.5.2.2 PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

7.5.2.3 SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear

attack. During security, the tester places the role of individual who desires to penetrate system.

7.5.2.4 VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

7.5.2.5 USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

- Input screen design.
- Output screen design.

APPENDIES

Main Source Code:

```
package com.ev.ElectricVechicle.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.client.RestTemplate;

import com.ev.ElectricVechicle.model.Admin;
import com.ev.ElectricVechicle.model.ElectricBike;
import com.ev.ElectricVechicle.model.ElectricCar;
import com.ev.ElectricVechicle.model.ElectricVechicleOwner;
import com.ev.ElectricVechicle.service.AdminService;

@Controller("/admin")
public class MainController {
```

```

@Autowired
private RestTemplate restTemplate;

@Autowired
private AdminService adminService;

@GetMapping("/")
public String homePage() {
    return "index";
}

@GetMapping("/admin")
public String adminPage() {
    return "admin";
}

@PostMapping("/admin/login")
public String login(Admin a, Model mod) {
    System.out.println(" admin "+a );
    //adminService.checking(a)

    if(a.getUsername().equalsIgnoreCase("admin")&&
a.getPassword().equalsIgnoreCase("admin")) {
        return "AdminHomePage";
    };
    mod.addAttribute("error", "Login fail");
    return "admin";
}

@GetMapping("/admin/car")
public String addCar() {

```

```

return "CarHome";
}

@PostMapping("/admin/car/add")

public String Car(ElectricCar car,Model mod) {
    System.out.println("carsss " +car);
    String receiverUrl = "http://localhost:9998/receiveData"; // URL of
Application B
    // Set headers
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
//
//      // Create an HttpEntity containing the headers and the body
    HttpEntity<ElectricCar> requestEntity = new HttpEntity<> (car,headers);
//
//      // Send POST request
    Boolean response = restTemplate.postForObject(receiverUrl, requestEntity,
Boolean.class);
    if(response) {
        mod.addAttribute("message", "message");
        mod.addAttribute("me", 1);
        return "CarHome";
    }

else {
        mod.addAttribute("err", "err");

```

```

        mod.addAttribute("me", 1);
        return "CarHome";
    }

}

@GetMapping("/admin/bike")
public String bikeHomePage() {
    return "BikeHome";
}

@GetMapping("/admin/update")
public String UpdatePage() {
    return "CarUpdatePage";
}

@PostMapping("/admin/updateSearching")
public String updateSearching(ElectricCar car, Model mod) {

    System.out.println(" car dfdsf vv "+car);

    String receiverUrl = "http://localhost:9998/updateSearching"; // URL of
Application B

// Set headers

    HttpHeaders headers = new HttpHeaders();

```

```

headers.setContentType(MediaType.APPLICATION_JSON);

// Create an HttpEntity containing the headers and the body
HttpEntity<ElectricCar> requestEntity = new HttpEntity<>(car, headers);

// Send POST request and get the response
ElectricCar response = restTemplate.postForObject(receiverUrl,
requestEntity, ElectricCar.class);

System.out.println(" null "+response);

// Check if the response is not null before adding it to the model
if (response != null) {
    System.out.println(" dskjldskjldsfkjldf ");
    mod.addAttribute("electricCar", response);

} else {
    // Optionally handle the case where response is null
    mod.addAttribute("hiddenForm", "1");
    mod.addAttribute("found", "found"); // Example of handling error
}

return "CarUpdatePage"; // This should return the name of your view
}

@PostMapping("/admin/car/update")

```

```

public String update(ElectricCar car,Model mod ) {

String receiverUrl = "http://localhost:9998/update"; // URL of Application B


    // Set headers
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

    // Create an HttpEntity containing the headers and the body
    HttpEntity<ElectricCar> requestEntity = new HttpEntity<>(car, headers);

    // Send POST request and get the response
    String response = restTemplate.postForObject(receiverUrl, requestEntity,
String.class);

    System.out.println(" response of the "+response);

    ElectricCar emptyCar=new ElectricCar();

    // Check if the response is not null before adding it to the model
//    if (response != null) {
//        mod.addAttribute("electricCar", response);
//    }

    // Optionally handle the case where response is null
//        mod.addAttribute("errorMessage", "Failed to update car information."); //
Example of handling error

```

```

//      }

// This should return the name of your view
if(response !=null) {

    if(response.equals("updated")) {
        System.out.println("enter the if blocked");
        mod.addAttribute("electricCar",emptyCar);
        mod.addAttribute("successOfUpdation", "Updated Successfully");
        mod.addAttribute("hiddenForm", "1");
    }
    else {
        mod.addAttribute("successOfUpdation", "Updated Failure");
    }
}
else {
    mod.addAttribute("hiddenForm", "1");
    mod.addAttribute("successOfUpdation", "Updated Failure");
}

return "CarUpdatePage";
}

@PostMapping("/admin/delete")
public String deletePage() {

```

```

        return "CarDelete";
    }

    @PostMapping("/admin/deleteSearching")
    public String deleteSearching(ElectricCar car, Model mod) {

        System.out.println(" car dfdsf vv "+car);

        String receiverUrl = "http://localhost:9998/updateSearching"; // URL of
Application B

        // Set headers
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);

        // Create an HttpEntity containing the headers and the body
        HttpEntity<ElectricCar> requestEntity = new HttpEntity<>(car, headers);

        // Send POST request and get the response
        ElectricCar response = restTemplate.postForObject(receiverUrl,
requestEntity, ElectricCar.class);

        System.out.println(" null "+response);

        // Check if the response is not null before adding it to the model

        if (response != null) {
            System.out.println(" dskjldskjldsfkjldf ");
            mod.addAttribute("electricCar", response);
        }
    }
}

```



```

    } else {
        // Optionally handle the case where response is null
        mod.addAttribute("hiddenForm", "1");

        mod.addAttribute("NotFoundData", "Not Found Vehicle"); // Example of
handling error
    }

    return "CarDelete"; // This should return the name of your view

}
@PostMapping("/admin/car/delete")

public String delete( ElectricCar car,Model mod ) {

    System.out.println(" car dfdsf vv "+car);
    String receiverUrl = "http://localhost:9998/delete";
// URL of Application B

    // Set headers

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

    // Create an HttpEntity containing the headers and the body

```

```

HttpEntity<ElectricCar> requestEntity = new HttpEntity<>(car, headers);

// Send POST request and get the response

String response = restTemplate.postForObject(receiverUrl, requestEntity,
String.class);

if(response.equals("deleted")) {
    System.out.println("yeah ");
    //mod.addAttribute(receiverUrl, response);

    mod.addAttribute("hiddenForm", "1");
    mod.addAttribute("successOfDeletion","Deleted Successfully");
}
else {
    mod.addAttribute("successOfDeletion","Deleted UnSuccessfully");

    System.out.println("no ");
    mod.addAttribute("hiddenForm", "1");
}

return "CarDelete";

}

```

```

@PostMapping("/getallOwnersdetails")

```

```

    public ResponseEntity<List<ElectricVechicleOwner>>
    getAllOwnersDetais(Model mod) {

```

```

        System.out.println("hello list");

        List<ElectricVechicleOwner>
ls=adminService.getAllOwnersDetais();

        mod.addAttribute("owners", ls);

        System.out.println("size "+ls.size());

        mod.addAttribute("count", "hello");

        return ResponseEntity.ok(ls);

    }

    @PostMapping("/get")

    public ResponseEntity<ElectricVechicleOwner> get(@RequestBody
ElectricVechicleOwner own,Model mod) throws Exception{

        System.out.println(" own "+own.getEmail());

        adminService.MatchOwnerandVin(own);

        return ResponseEntity.ok(own);

    }


//    @PostMapping("/get")
//
//    public String get(@RequestBody ElectricVechicleOwner own) {
//        System.out.println(" own "+own);
//
//
//
//        adminService.MatchOwnerandVin(own);
//
//        return "own";
//    }

    /* Bike functions */

```

```

    @PostMapping("/admin/bike/add")
    public String bikeAdd(ElectricBike bike, Model mod) {

        String receiverUrl = "http://localhost:9998/addBikeDetails"; // URL of
Application B

        // Set headers

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        //

//        // Create an HttpEntity containing the headers and the body
        HttpEntity<ElectricBike> requestEntity = new HttpEntity<>(bike,
headers);

        //

//        // Send POST request

        Boolean response = restTemplate.postForObject(receiverUrl,
requestEntity, Boolean.class);

        if(response) {
            mod.addAttribute("hiddenmsg", "1");

return "BikeHome";
        }
        else {
            return "BikeHome";
        }
    }

```

```

    }

    @GetMapping("/admin/bike/update")
    public String bikeUpdatePage(ElectricBike bike) {
        return "BikeUpadatePage";
    }

    @PostMapping("/admin/bike/updateSearching")
    public String bikeUpdateSearching( ElectricBike bike,Model mod) {
        System.out.println(" bike "+bike);

        String receiverUrl = "http://localhost:9998/bike/updateSearching";
// URL of Application B

// Set headers

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);


        // Create an HttpEntity containing the headers and the body
        HttpEntity<ElectricBike> requestEntity = new HttpEntity<>(bike,
headers);

        ElectricBike response = restTemplate.postForObject(receiverUrl,
requestEntity, ElectricBike.class);


try {
    if(response!=null) {
        mod.addAttribute("ElectricBike", response);
        return "BikeUpadatePage";
    }
}

```

```

        else {
            mod.addAttribute("successOfUpdation", "Not Found Vehicle");
            return "BikeUpdatePage";
        }
    } catch (Exception e) {
        mod.addAttribute("successOfUpdation", "Not Found Vehicle");
        return "BikeUpdatePage";
    }
}

@PostMapping("/admin/bike/update")
public String bikeUpdate(ElectricBike bike, Model mod) {
String receiverUrl = "http://localhost:9998/bike/update"; // URL of Application B
// Set headers

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

    // Create an HttpEntity containing the headers and the body

    HttpEntity<ElectricBike> requestEntity = new HttpEntity<>(bike, headers);

    Boolean response = restTemplate.postForObject(receiverUrl, requestEntity,
Boolean.class);

    if(response) {
        mod.addAttribute("successOfUpdation", "Successfully Updated");
        return "BikeUpdatePage";
    }
}

```

```

    }
    else {
        mod.addAttribute("successOfUpdation", "Fail to Update");
        return "BikeUpdatePage";
    }
}

@GetMapping("/bikeUpdate/close")
public String bikeHome() {
    System.out.println("Hello close page is called ");
    return "BikeHome";
}

@GetMapping("/admin/bike/delete")
public String bikeDeletePage() {
    return "BikeDeletePage";
}

@PostMapping("/admin/bike/DeleteSearching")

public String DeleteSearch(ElectricBike bike, Model mod) {
    String receiverUrl = "http://localhost:9998/bike/updateSearching";
    // URL of Application B
    // Set headers
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

    // Create an HttpEntity containing the headers and the body
    HttpEntity<ElectricBike> requestEntity = new HttpEntity<>(bike,
headers);

```

```

        ElectricBike response = restTemplate.postForObject(receiverUrl,
requestEntity, ElectricBike.class);

        System.out.println(" response "+response);

        if(response!=null) {

            mod.addAttribute("ElectricBike", response);

            return "BikeDeletePage";

        }
        else {

            //.addAttribute("successOfUpdation", "Fail to Update");

            return "BikeDeletePage";

        }

    }
}

```

```

@PostMapping("/admin/bike/delete")

```

```

public String deletionProcess(ElectricBike bike,Model mod) {
System.out.println(" hello guys ");
String receiverUrl = "http://localhost:9998/bike/delete"; // URL of Application B
// Set headers

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);

    // Create an HttpEntity containing the headers and the body
    HttpEntity<ElectricBike> requestEntity = new HttpEntity<>(bike,
headers);
}

```



```

        Boolean response = restTemplate.postForObject(receiverUrl, requestEntity,
Boolean.class);

        if(response) {

            System.out.println("success deletion ");

            mod.addAttribute("successOfDeletion", "Deleted Successfully");

            return "BikeDeletePage";

        }

        mod.addAttribute("successOfDeletion", "Deleted UnSuccessfully");

        return "BikeDeletePage";

    }

    @PostMapping("/admin/home")
    public String adminhomePage() {

        return "AdminHomePage";

    }

```

```

@PostMapping("/admin/logout")

    public String allHomePage() {

        return "index";

    }

}

```

Implementation Of Eelectrical vechicle application:

```

package com.ev.ElectricVechicle;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

```

```

import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class ElectricVechicleApplication {

    public static void main(String[] args) {
        SpringApplication.run(ElectricVechicleApplication.class, args);
        System.out.println("Application running sucessfully");
    }

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}

```

Implementation Of Eelectrical vechicle Cloud:

```

package com.ev.cloud.EvCloud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EvCloudApplication {

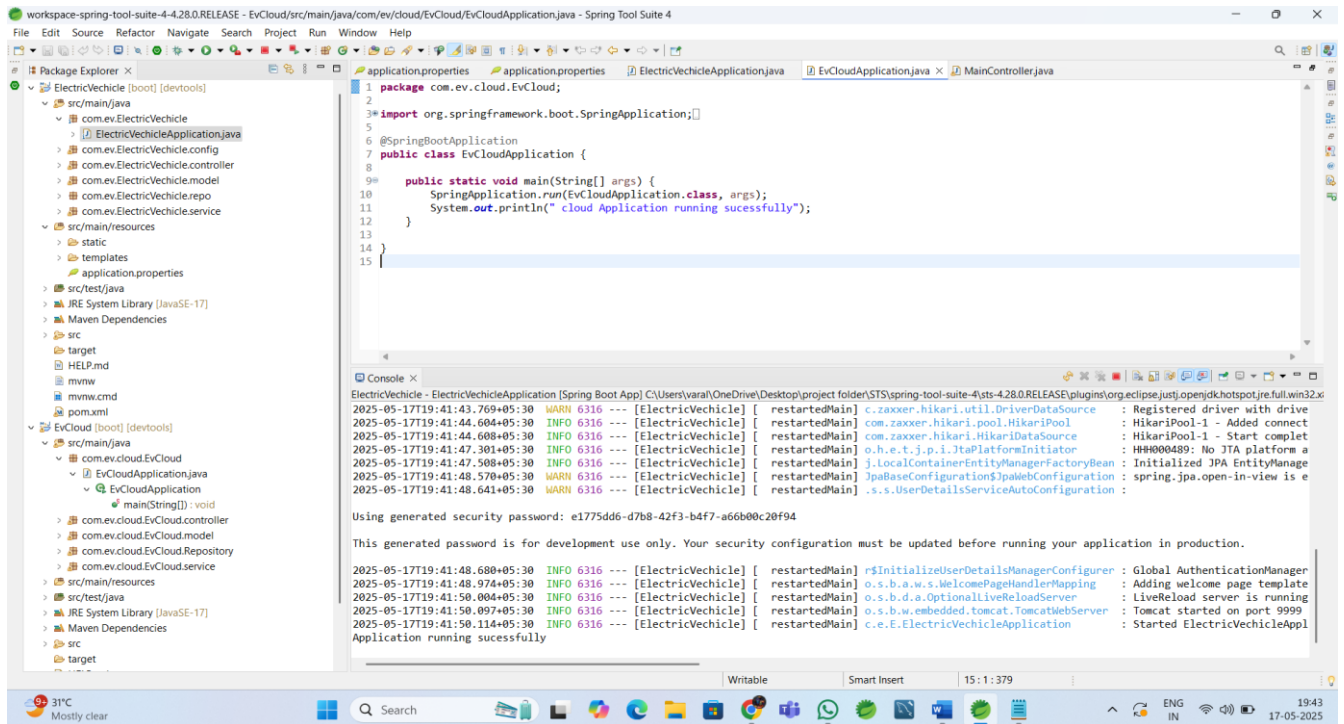
    public static void main(String[] args) {

```

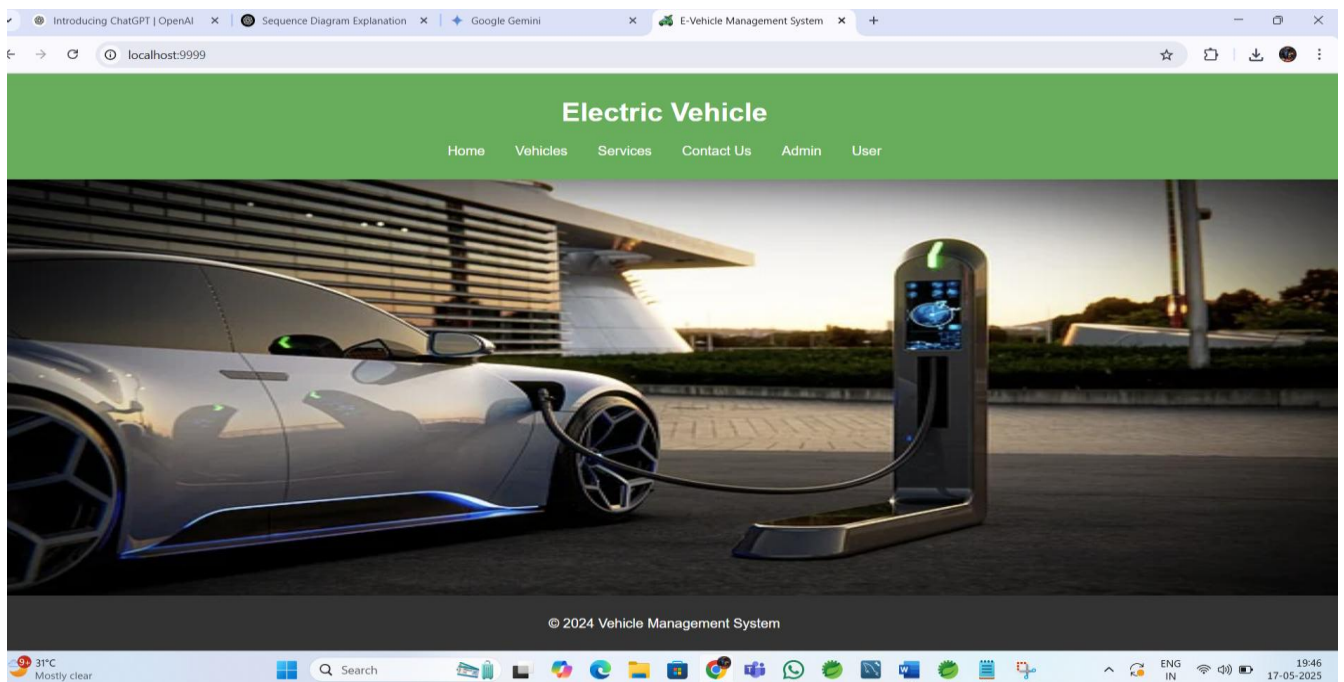
```
SpringApplication.run(EvCloudApplication.class, args);  
  
System.out.println(" cloud Application running sucessfully");  
  
    }  
  
}
```

Screenshots:

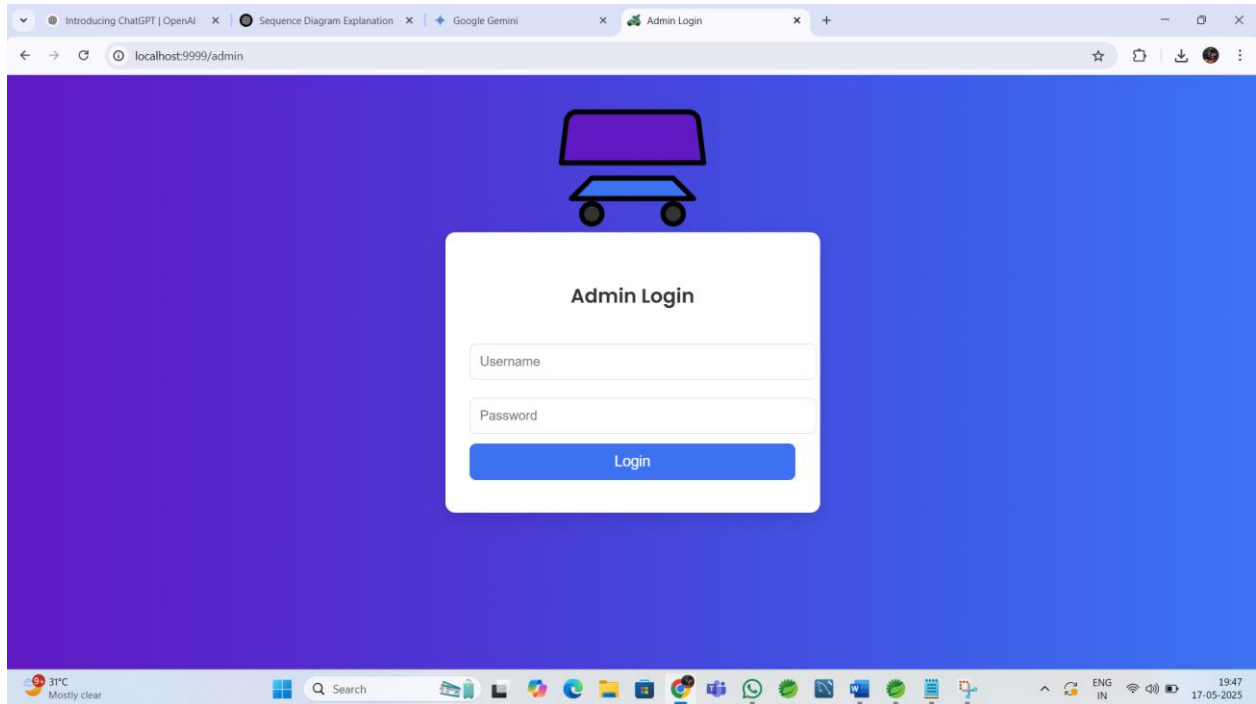
Application Running Process:



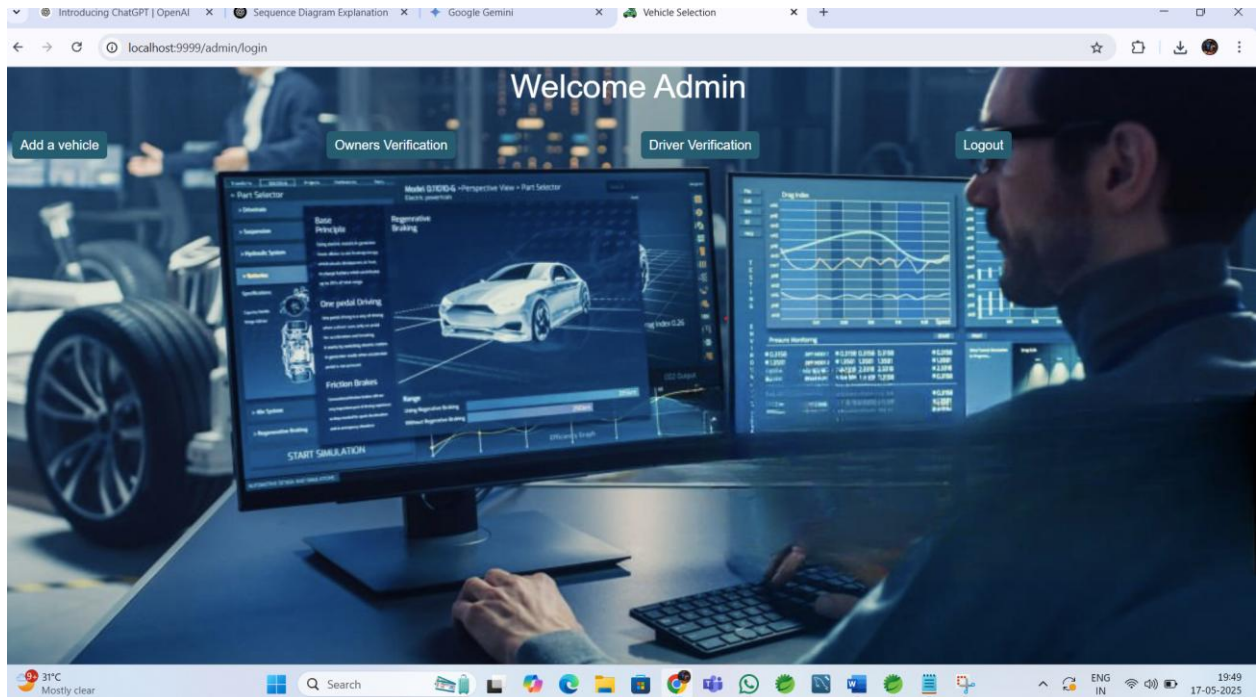
Home Page:



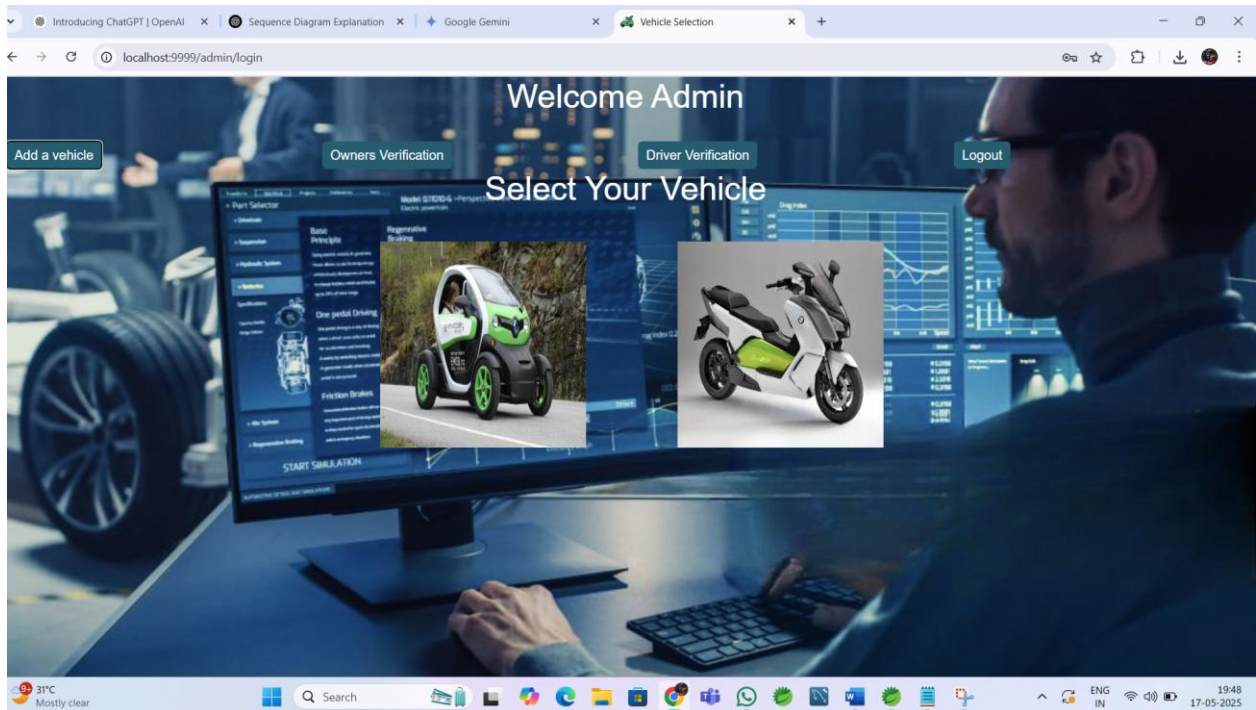
Admin Page:



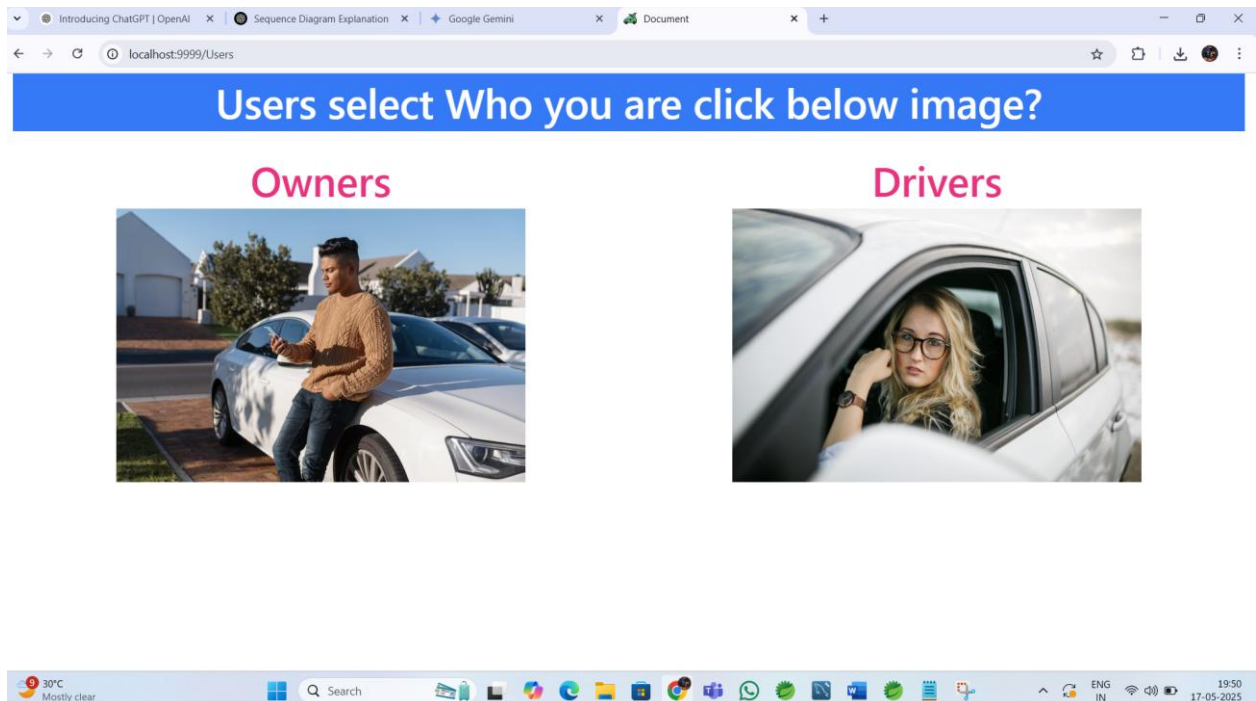
Welcome Admin page:



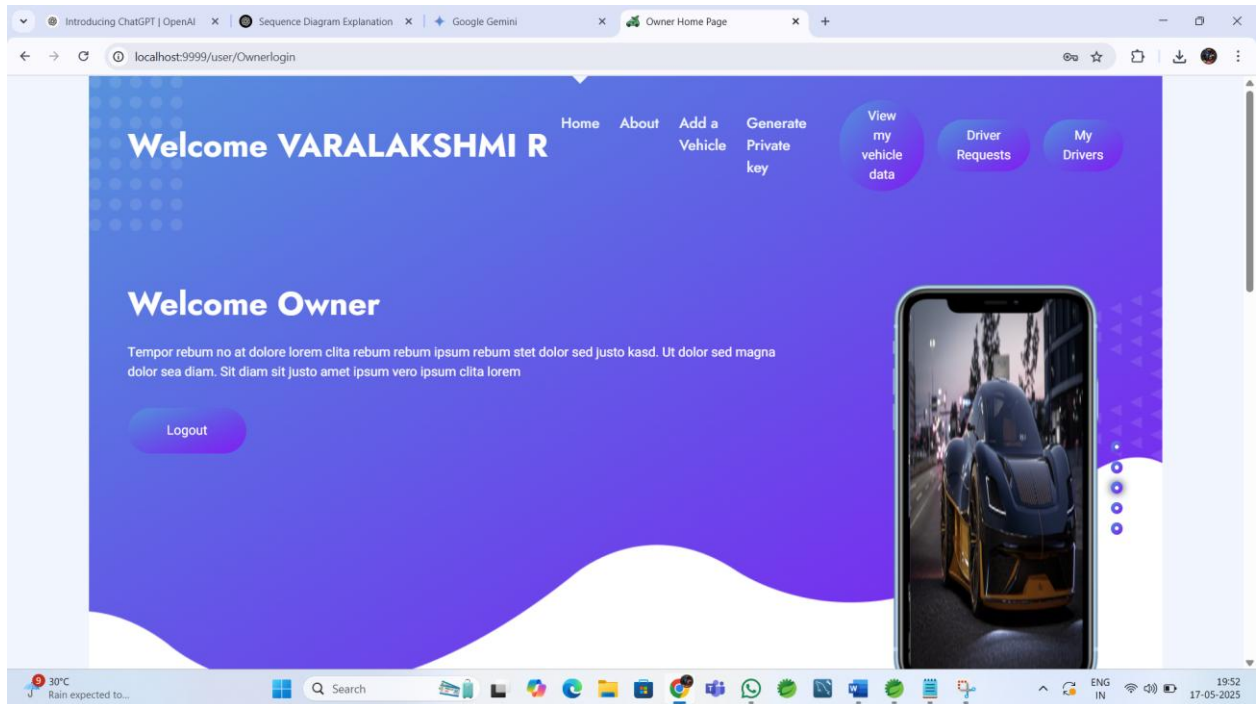
Vehicle Selection view:



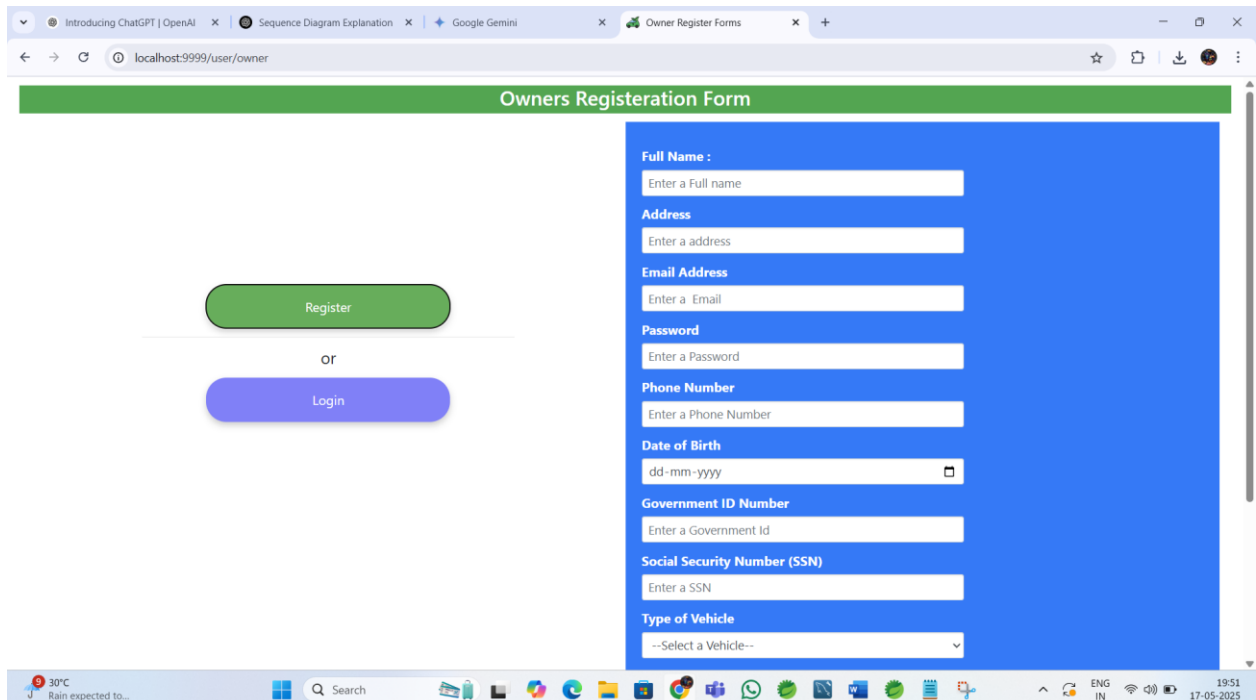
User Page:



Owner page:



Owner Registration Page:



Owner Login page:

Login

Email
varalakshmi02052003@gmail.com

Private Key

OTP: Check your email for otp
19294

VIEW DATA

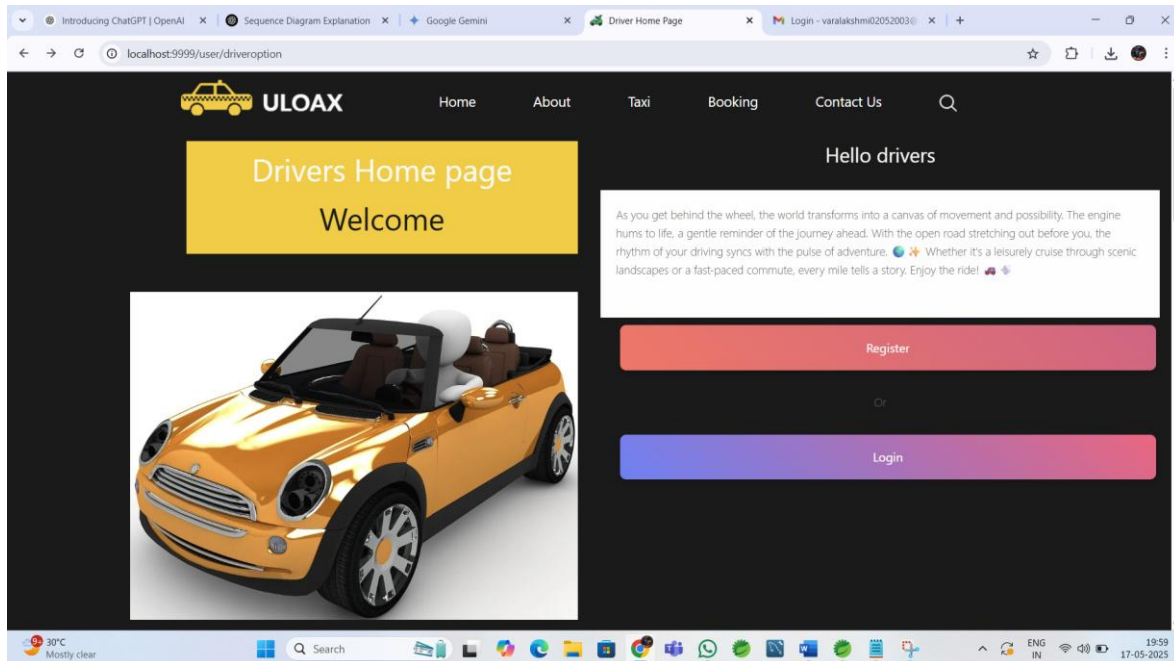
RESEND OTP

Car Details page:

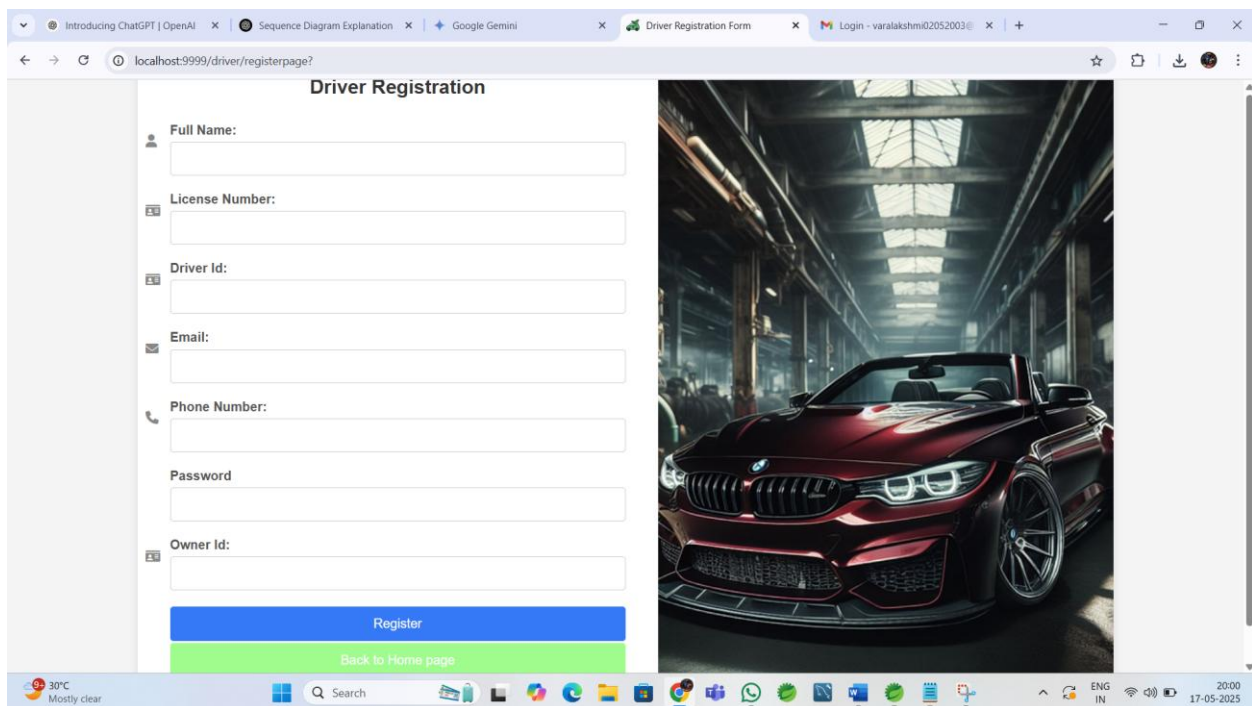
Car Details

Make: bmw	Model: bmw123	Year 2025	Battery Type lithiumion
Battery Capacity (kWh) 23	Range (miles/km) 33	Efficiency (miles/kWh) 17.0	Has Regenerative Braking Yes
Supports Fast Charging Yes	Charging Time (hours) 55.0	0 to 60 mph (seconds) 55.0	Color blue

Driver Home Page:



Driver Registration page:



Driver can access the Electrical vehicle:

The screenshot shows a web browser window with the address bar displaying 'localhost:9999/owner/driverlist'. The browser has several tabs open, including 'Introducing ChatGPT | OpenAI', 'Sequence Diagram Explanation', 'Google Gemini', 'Driver List', and 'Login - varalakshmi02052003@'. The 'Driver List' tab is active, showing a table with the following data:

Full Name	Driver ID	Email	License No	Owner ID
Prasanna	1789	prasanna@gmail.com	963852741	22

Below the table is a large image of a futuristic city at night with glowing lines and buildings. The Windows taskbar at the bottom shows the date as 17-05-2025 and the time as 19:56.

REFERENCES

PKI (X.509). Accessed: Jan. 10, 2024. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc5280> [23] A. Shamir, “Identity-based cryptosystems and signature schemes,” in Proc. Workshop theory Appl. Cryptograph. Techn., 1984, pp. 47–53

D. Boneh and M. Franklin, “Identity based encryption from the Weil pairing,” in Proc. Advance Crypto’01, 2001, pp. 213–229.