RV COLLEGE OF ENGINEERING®, BENGALURU-560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Next Gen Car Showroom Management System

Project Report

Hardik Hiraman Pawar	**1RV21CS046**
Karan Sathish	**1RV21CS058**
Harshit Dhoot	**1RV21CS049**

in partial fulfillment for the requirement of 6h Semester

Software Engineering Laboratory Mini Project (21IS62)

Under the Guidance of

Prof. Smriti Srivastava
Assistant Professor
Dept of CSE
RV College of Engineering

Academic Year 2023- 2024

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **'Next gen car showrrom management system'** is carried out by **Hardik Hiraman Pawar (1RV21CS046) , Harshit Dhoot (1RV21CS049), Karan Sathish(1RV21CS058)** , who are bonafide students of RV College of Engineering®, Bengaluru, in partial fulfillment of the curriculum requirement of 6th Semester Software Engineering (21IS62) Laboratory Mini Project during the academic year **2023-2024**. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements in all respect laboratory mini-project work prescribed by the institution.

Signature of Faculty In-charge

Head of the Department
Dept. of CSE, RVCE

External Examination

Name of Examiners **Signature with date**

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this project work. I would like to take this opportunity to thank them all.

I deeply express my sincere gratitude to my guide **Prof. Smriti Srivastava, Assistant Professor,** Department of CSE, RVCE, Bengaluru, for his able guidance, regular source of encouragement and assistance throughout this project

I would like to thank Dr.Ramakanth Kumar P, Head of Department, Computer Science & Engineering, R.V.C.E, Bengaluru, for his valuable suggestions and expert advice.

First and foremost I would like to thank **Dr. Subramanya. K. N**, Principal, R.V.C.E, Bengaluru, for his moral support towards completing my project work.

I thank my Parents, and all the Faculty members of Department of Computer Science & Engineering for their constant support and encouragement.

Last, but not the least, I would like to thank my peers and friends who provided me with valuable suggestions to improve my project.

Abstract

Traditional car showroom management systems often rely on manual processes and disconnected databases, leading to inefficiencies in data management, customer interaction, and payment processing. This project addresses these shortcomings by developing a webbased car showroom management system that utilizes a combination of Flask, MySQL, and NoSQL databases.

The system offers a comprehensive suite of functionalities, including car listing management, customer interaction tools, test drive booking, sales tracking, and employee incentive tracking. A user-friendly web interface and unique QR code generation for customer identification prioritize accessibility and streamline both customer and employee interactions. Additionally, the integration of two secure payment gateways, Stripe and a blockchain-based option, provides flexibility and caters to diverse customer preferences. A customer review system empowers customers to provide feedback, fostering a customercentric approach.

The project successfully demonstrates the potential of technology to enhance car showroom management. The system streamlines operations, improves customer experience with a user-friendly interface and diverse payment options, and motivates employees through incentive tracking features. This foundation can be further developed to include advanced inventory management, in-depth financial reporting, and direct car service scheduling functionalities. By continuously improving upon the core functionalities and incorporating evolving technologies, this system has the potential to significantly improve efficiency, customer satisfaction, and employee engagement within the car showroom industry.

Table of Contents

1. Introduction 7

2. Software Requirement Specification 8
   2.1 Introduction
   2.2 Functional Requirements
   2.3 Software Requirements
   2.4 Interface Requirements

3. Estimation of Project Metrics 12
   3.1 Introduction
   3.2 Preliminary Analysis
   3.3 Analysis using COCOMO

4. UML use case Diagrams and Use case scenarios 16
   4.1 Overview
   4.2 Actors
   4.3 Major use cases and their descriptions
   4.4 Relationships between use cases
   4.5 Summary

5. Domain classes 20

6. Activity modeling 22
   6.1 Register a new customer
   6.2 Book a test drive/purchase
   6.3 Whatsapp and Email
   6.4 Review and Sentiment Analysis

7. UML class and Sequence Diagrams 26
   7.1 UML Class diagram
   7.2 Sequence diagram

8. Data Flow Diagrams 30
   8.1 Level0 DFD
   8.2 Level1 DFD

9. Designing Test Suites 35

10. Conclusion & Future Enhancement 41

References 42

Appendix: Snapshots 43

GLOSSARY

- **DFD**: Data Flow Diagram - A graphical representation of data movement within a system.
- **Flask**: A Python web framework used for developing web applications.
- **MySQL**: A relational database management system used to manage structured data.
- **MongoDB**: A NoSQL database used for storing unstructured data, such as images and JSON-like documents.
- **QR Codje**: A machine-readable code used for storing information, such as customer IDs, which can be scanned for quick access.
- **Stripe**: A payment processing platform used for handling online payments.
- **NoSQL**: Non-relational databases that provide flexible schemas and scalability, used for managing unstructured data.
- **Appointment**: A scheduled meeting between a customer and showroom staff for activities like test drives.
- **Test Drive**: An activity where a potential customer drives a car before deciding to purchase it.
- **API**: Application Programming Interface - A set of rules that allows different software entities to communicate with each other.
- **Review**: Feedback provided by customers about their experience, typically including a star rating and written comments.
- **Entity Relationship Diagram (ERD)**: A diagram that shows the relationships between different entities in a database.
- **Normalization**: The process of organizing a database to reduce redundancy and improve data integrity.
- **SRS**: Software Requirement Specification - A detailed description of the software's functionalities and constraints.

Chapter 1

Introduction

The ever-evolving automotive industry demands efficient management systems that streamline operations and enhance customer experience. This project addresses this need by developing a webbased car showroom management system that utilizes a combination of Flask, MySQL, and NoSQL databases.

Project Objectives:

- **Streamline Showroom Operations:** Develop a comprehensive system to manage car listings, customers, test drive bookings, and sales data, facilitating efficient workflow within the showroom.

- **Enhance User Experience:** Design a user-friendly web interface that is accessible and simplifies interactions for both customers and showroom employees.

- **Offer Diverse Payment Options:** Integrate two secure payment gateways: Stripe for traditional transactions and a blockchain-based option for added security and cater to evolving customer preferences.

- **Empower Customers:** Implement features like unique QR code generation for identification and a review system to provide feedback on employee performance, fostering a customercentric approach.

- **Motivate Employees:** Integrate functionalities for employees to manage sales data and track their associated incentives, promoting a results-oriented environment.

Project Scope:

This project focuses on developing a web-based application using Flask for the user interface and a combination of MySQL and NoSQL databases for data storage. The system will encompass functionalities relevant to car showroom management, customer interaction, employee performance tracking, and secure payment processing with two integrated gateways.

While the core functionalities cater to core showroom needs, the scope excludes functionalities like advanced inventory management, in-depth financial reporting, or direct car service scheduling. These functionalities can be considered for future enhancements based on evolving requirements.

This report will delve deeper into the system's architecture, functionalities, and implementation details, demonstrating how it achieves the outlined objectives within the defined scope. It will highlight the project's contribution to improved car showroom management, enhanced customer experience, and employee engagement.

Chapter 2

Software Requirement specification

2.1.. Introduction

The purpose of this document is to define and describe the requirements of the project and to spell out the system's functionality and its constraints

2.1.1. Scope of this Document

The document details the various requirements for the Development of our project title "Car Showroom Web Application". The functional requirements and the software requirements are envisioned after carefully analysing the needs and requirements of all stakeholders including customers and employees of the physical car showroom

2.1.2 Overview

The product is a web application with features to enable seamless booking and sale of cars listed in the showroom while maintaining user and employee statistics to calculate sales benefits for the users. We take it a step further by integrating features such as sentiment analysis to consolidate feedback and provide an efficient review of products and services to the admins of the web application. Some other important features include the integration of multiple modes of payment including credit/debit card and crpto payment gateways to support user convenience regardless of the mode of payment. An innovative feature here is QR based identification of users to enable ease of login or authentication

2.1.3 Business Context

This web application can help showroom owners boost sales while encompassing a holistic view of trends in customer satisfaction to make informed decisions .

2.2. Functional Requirements

2.2.1. Employee Management:

- User registration and management
- Role-based access control
- Performance evaluation

2.2.2. Customer Management:

- Customer registration with OCR integration for ID card scanning.
- Appointment scheduling and reminder system
- Test drive management
- Purchase history tracking

- Communication channels (email, SMS)

2.2.3. Car Management:

- Detailed car information including specifications, features, images, and videos.
- Inventory tracking and management
- Maintenance scheduling and service history
- Sales analysis and reporting

2.2.4. Other Features:

- Dashboard for real-time data visualization
- Reporting and analytics

2.3. Software Requirmeents:

2.3.1. Relational Database Management System (RDBMS): MySQL

- **Purpose**:
  - Stores structured data, mainly for employee, customer, and car information.
  - Maintains relationships between different data entities.
  - Ensures data integrity, consistency, and security.
- **Features:**
  - Widely used, open-source, and community-supported.
  - Handles complex queries efficiently.
  - Supports ACID (Atomicity, Consistency, Isolation, Durability) properties for reliable transactions.

2.3.2. NoSQL Database: MongoDB

- **Purpose:**
  - Stores unstructured and semi-structured data, such as images, OCR-processed text, and customer feedback.
  - Provides flexibility in data schema and scalability for large datasets.
- **Features:**
  - Document-oriented structure for storing data in JSON-like documents.
  - High performance and scalability.
  - Supports flexible querying and indexing.

2.3..3. OCR Software: EasyOCR

- **Purpose:**

  - Extracts text from images and documents for features like employee ID card scanning and customer registration.

- **Features:**

  - Lightweight and easy to integrate with Python applications.
  - Supports multiple languages and text detection.
  - Can handle various image formats and document types.

2.3.4. Web Development Framework: Flask

- **Purpose:**

  - Creates the server-side logic and routes for the web application.
  - Handles user interactions, database connections, and API endpoints.

- **Features:**

  - Lightweight and flexible microframework.
  - Easy to learn and use for small to medium-sized projects.
  - Offers good performance and scalability.

2.3.5. Frontend Framework: Tailwind CSS

- **Purpose:**

  - Styles the user interface with utility classes for rapid development.
  - Provides a consistent and customizable look and feel.

- **Features:**

  - No pre-designed components, allowing complete control over the UI.
  - Highly customizable with a large collection of utility classes.
  - Promotes clean and maintainable code.

2.4. Interface Requirements:

2.4.1. User Interfaces

- **Web GUI**
  The user interface for this program is the interface rendered by our flask application as an interactive web based GUI
- **4.1.2 CLI**
  There is no command line interface
- **4.1.3 API**
  There is no API for the product

2.4.2. Hardware Interfaces

The web application requires access to a database which needs to be hosted at a high speed server.

2.4.3 Communications Interfaces

If we decide to implement an Ad Hoc network for a shared database, the operating system will handle those connections.

Chapter 3

Estimation of Project Metrics

3.1. Introduction:

Our project titled "Car Showroom Management Website" provides a simple and easy to use interface to sell cars, book test drives, manage employee incentives and keep track of the overall firm's business statistics while introducing innovative features such as sentiment analysis and crypto payment methods.

This section is concerned with ascertaining the project metrics in order to have a good estimate of the project timeline and feasibility before beginning to work on the project and prevent any pitfalls. For this, we have employed two popular techniques, the COCOMO(cost constructive model) and the Halstead's complexity metrics.

3.2. Preliminary Analysis:

- The first step to evaluating the project metrics involves identifying the project type. There exist three main project types which are organic, semi-detached and embedded projects. Our project is more of a stand-alone application and hence, falls into the category of organic projects.
- The development team in our case is a small team of four members which is another characteristic of an organic project.
- The tech stack used in this project has been used by all members of our team numerous times before making it more suited too be an organic project.

3.3. Analysis as per COCOMO:

Now that we have identified the type of project, it is now time to proceed with the estimation as per the COCOMO model. This has three main stages:

3.3.1.Basic COCOMO model:

In this stage, the total effort required for the software development(in person months) and the time required to develop the software are estimated using KDSI, i.e, the estimated size of the project in Kilo Delivered Source instructions. Below is the formula for the same

$$Effort = a * (KDSI)^b \ PM$$

$$Tdev = 2.5 * (Effort)^c \ Months$$

Here, a, b, and c are constants (2.4, 1.05, 0.38) for organic projects.

In the case of our project, the number of source instructions is composed of approximately:

- HTML: 3500
- CSS: 2500
- JS: 500
- FLASK: 1000
- Solidity: 40
- React(for payment gateway): 800
- Sentiment Analysis, email and Whatsapp integration: 900

Total number of instructions = 9240 = 9.24 = KDSI

Now, by the formula discussed earlier,

Effort = 24.78

Tdev = 8.4 months

Hence, from the first analysis stage, we conclude that it would take approximately 8.4 months to complete the project. Although this estimate is on the higher side, since many of the components we plan to include have aldready been implemented by us in old projects and only the project integration is a hassle. Realistically, it would take around 4 months.

3.3.2. Intermediate COCOMO model:

In this stage, the concept of effort multipliers and Effort Adjustment Factor are used in order to make the estimation more precise to the real world scenario

In order to estimate EAF, the appropriate value from each row is chosen and multiplied together to obtain the value.

In our scenario, the computed value of EAF turned out to be 0.4432.

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Using this EAF, we get the corrected estimates as per the formulae below:

$$Effort|_{corrected} = Effort * EAF$$

$$Tdev|_{corrected} = 2.5 * (Effort|_{corrected})^{c}$$

In our case, the calculation is as follows:

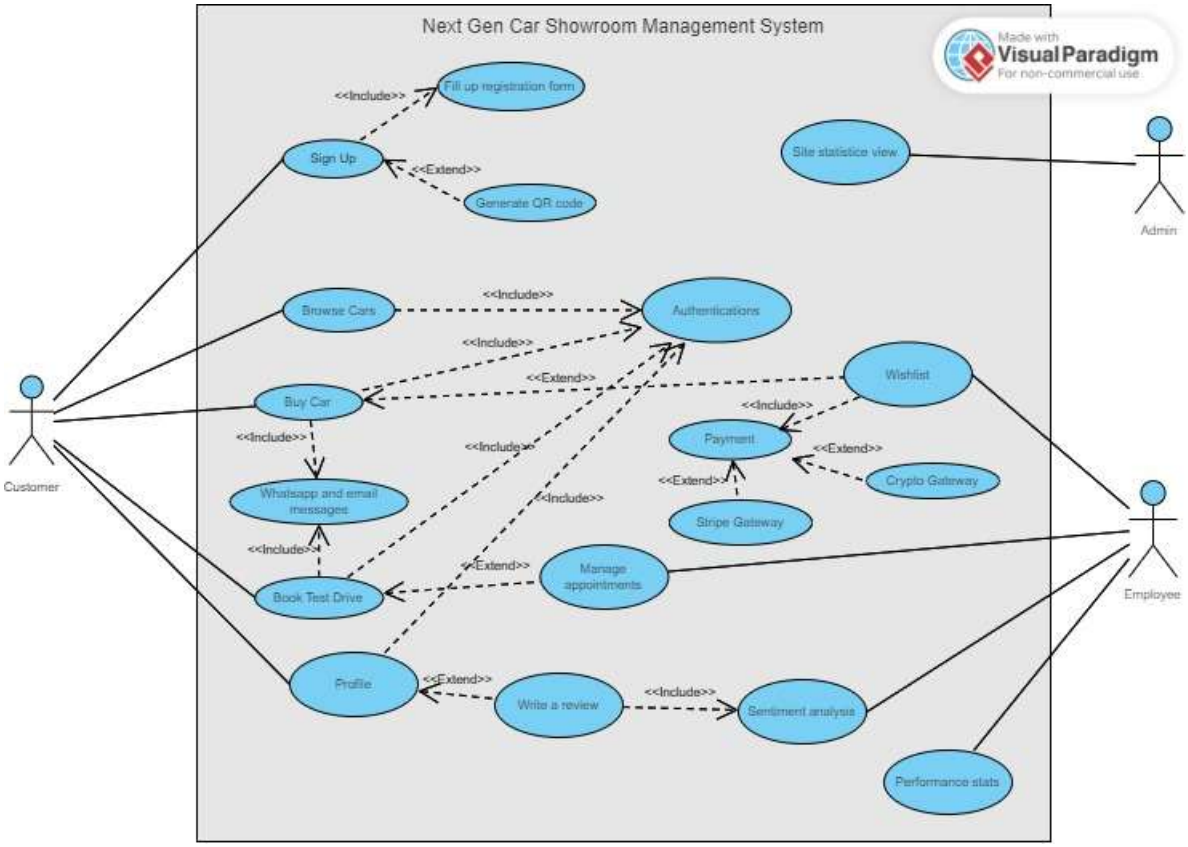
$$\text{Effort} = 24.78 * 0.4432 = 10.98\text{PM}$$

$$\text{Tdev} = 2.5 * 2.485 = 6.21\text{months}$$

Considering the calculations described above, the expected duration of the project would be around 6 months if the estimated number of lines of code turns out to be accurate.

Chapter 4

UML Use Case Diagram



4.1 Overview

The use case diagram offers a high-level overview of the system, showcasing the various functionalities available to different actors. The actors identified include the Customer, Admin, and Employee. Each actor interacts with different use cases, highlighting the relationships and dependencies within the system.

4.2 Actors

The actors in the system represent the different user roles interacting with the system's functionalities.

4.2.1 Customer

- Interacts with various use cases such as Sign Up, Browse Cars, Buy Car, Book Test Drive, Profile, and Manage Appointments.

4.2.2 Admin

- Has access to specialized functionalities like the Site Statistics View.

4.2.3 Employee

- Engages with functionalities including Wishlist, Payment, Manage Appointments, and Performance Stats.

4.3 Major Use Cases and Their Descriptions

The following sections provide detailed descriptions of the key use cases identified in the system.

4.3.1 Sign Up

- Description: This use case allows a new customer to register within the system.
- Sub-cases:
    - 4.3.1.1 Fill up Registration Form: A required step where the customer provides necessary details to sign up.
    - 4.3.1.2 Generate QR Code (<<Extend>>): After successful registration, a QR code may be generated for the customer.

4.3.2 Browse Cars

- Description: Enables customers to browse through the available cars in the showroom.
- Sub-cases:
    - 4.3.2.1 Authentications (<<Include>>): Ensures that only authenticated users can browse the cars.

4.3.3 Buy Car

- Description: Facilitates the car purchasing process for customers.
- Sub-cases:
    - 4.3.3.1 Authentications (<<Include>>): Authentication is required for the purchase process.
    - 4.3.3.2 Whatsapp and Email Messages (<<Include>>): Notifications regarding the purchase are sent through these channels.
    - 4.3.3.3 Payment (<<Include>>): The payment process is integral to completing the purchase.
    - 4.3.3.4 Crypto Gateway (<<Extend>>): Provides an optional payment method using cryptocurrency.
    - 4.3.3.5 Stripe Gateway (<<Extend>>): Another optional payment method using the Stripe payment gateway.

4.3.4 Book Test Drive

- Description: Allows customers to book a test drive.
- Sub-cases:
    - 4.3.4.1 Whatsapp and Email Messages (<<Include>>): Notifications are sent to confirm the booking.
    - 4.3.4.2 Manage Appointments (<<Extend>>): Extends to managing appointments related to test drives.

4.3.5 Profile

- Description: Manages the customer's profile within the system.
- Sub-cases:
    - 4.3.5.1 Write a Review (<<Extend>>): Allows customers to write reviews about cars or the showroom.

4.3.6 Wishlist

- Description: Allows both customers and employees to manage a wishlist of cars.
- Sub-cases:
    - 4.3.6.1 Authentications (<<Include>>): Ensures that wishlist management is only accessible to authenticated users.

4.3.7 Payment

- Description: Manages the payment transactions within the system.
- Sub-cases:
    - 4.3.7.1 Stripe Gateway (<<Include>>): Uses Stripe for processing payments.
    - 4.3.7.2 Crypto Gateway (<<Extend>>): Provides an alternative payment method using cryptocurrency.

4.3.8 Manage Appointments

- Description: Enables employees to manage customer appointments, including test drives.

4.3.9 Site Statistics View

- Description: Provides the admin with access to various site statistics for monitoring and decision-making purposes.

4.3.10 Sentiment Analysis

- Description: Analyzes customer sentiments from reviews or feedback to gain insights.
- Sub-cases:
    - 4.3.10.1 Performance Stats (<<Extend>>): The results of the sentiment analysis can influence performance statistics.

4.3.11 Performance Stats

- Description: Provides statistical data regarding employee performance and overall system efficiency.

4.4 Relationships Between Use Cases

This section details the relationships between various use cases.

4.4.1 Include Relationships (<<Include>>)

- Indicates that the base use case mandates the behavior of the included use case.

4.4.2 Extend Relationships (<<Extend>>)

- Indicates that the base use case optionally includes the behavior of the extending use case.

4.5 Summary

The use case diagram provides a comprehensive view of the "Next Gen Car Showroom Management System." It helps in understanding the scope of the system, the various

actors involved, and the relationships between different use cases. This lays a solid foundation for further detailed analysis and system design.

Chapter 5

Domain Classes

From the problem statement, we cab identify the following nouns and noun phrases:

- Car
- Showroom
- Admins
- Managers
- Employees
- Web Application
- Customer
- Sales
- Bookings
- Sales
- Payment
- WhatsApp
- E-mail
- Sentiment Analysis
- Review
- QR-Code

We get a set of 16 potential classes from this but many of these may not be relevant or may be redundant. After filtering out the classes based on out understanding of the problem statement keeping in mind the context, the final set of classes are:

- Customer
- Employee
- Admin
- Car
- Inventory
- Appointments
- Payment

- Invoice
- WhatsApp
- E-mail
- Review
- Sentiment Analysis

The above identified classes pave the way for modelling of design and implementation classes.

Chapter 6

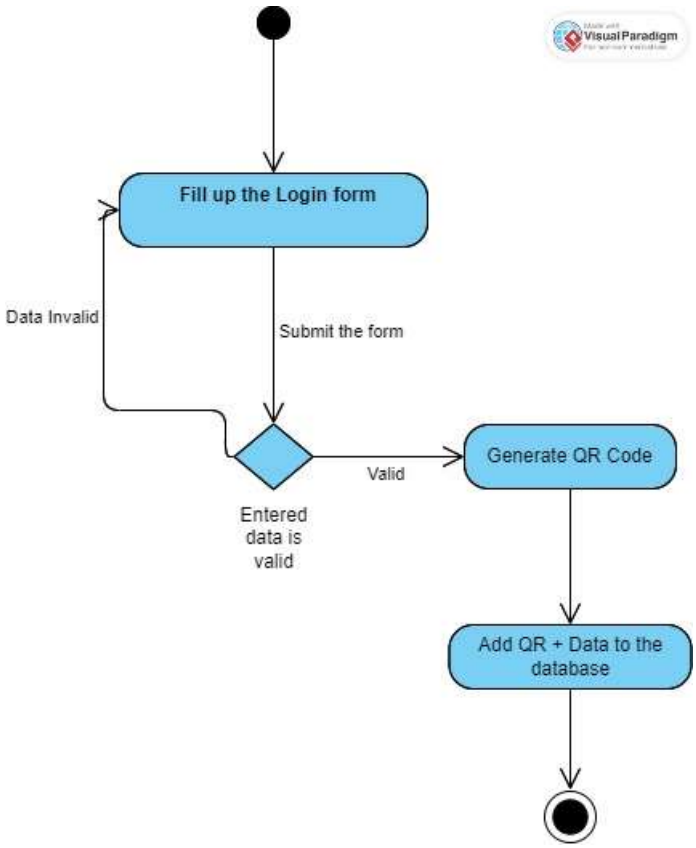
Acitivity Modelling

6.1. Activity Diagram

From the given problem, we can identify the following functionalities offered by the system:

- Register a new Customer
- Add a car to the wish-list
- Purchase a car
- Write a Review

6.1.1. Register a new Customer:

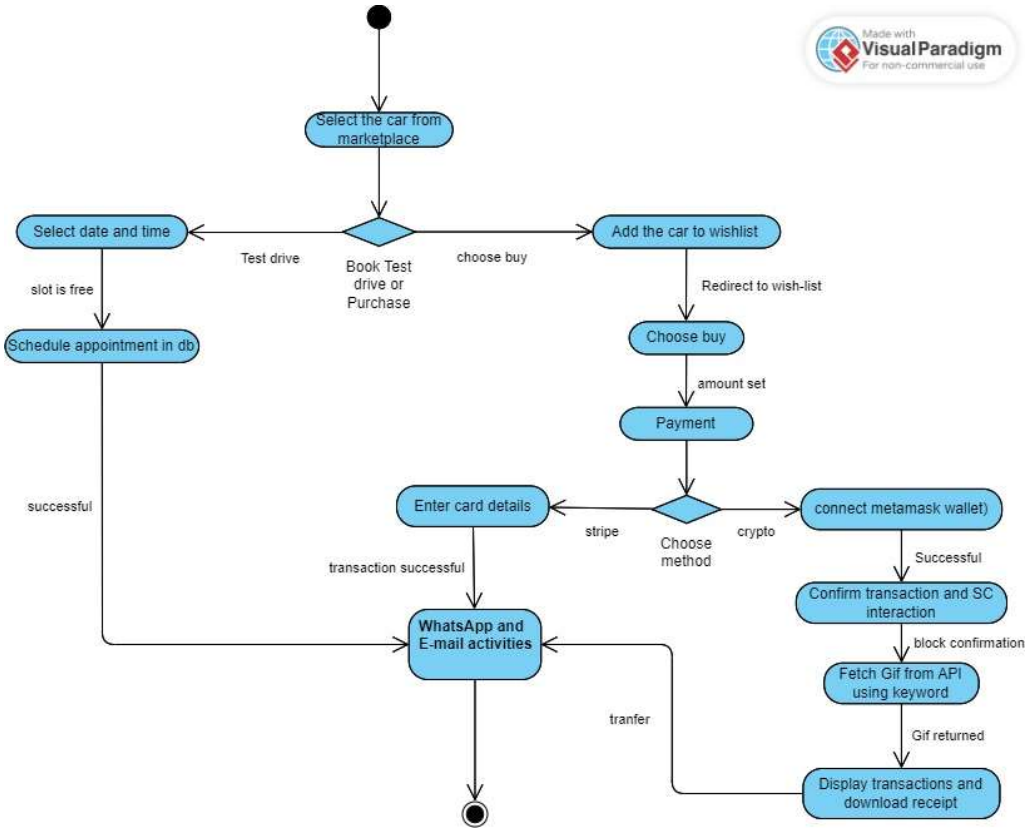


- **Start:** The process begins with the user filling out a login form.
- **Submit the Form:** Once the form is filled, the user submits it.
- **Validation Check:** The system checks if the entered data is valid.

- **If the data is invalid:** The user is prompted to fill in the form again.
- **If the data is valid:** The process continues to the next step.

- **Generate QR Code:** A QR code is generated based on the valid data.
- **Store Data:** The generated QR code and the corresponding data are added to the database.
- **End:** The process concludes.

6.1.2. Book a Test Drive/ Purchase:



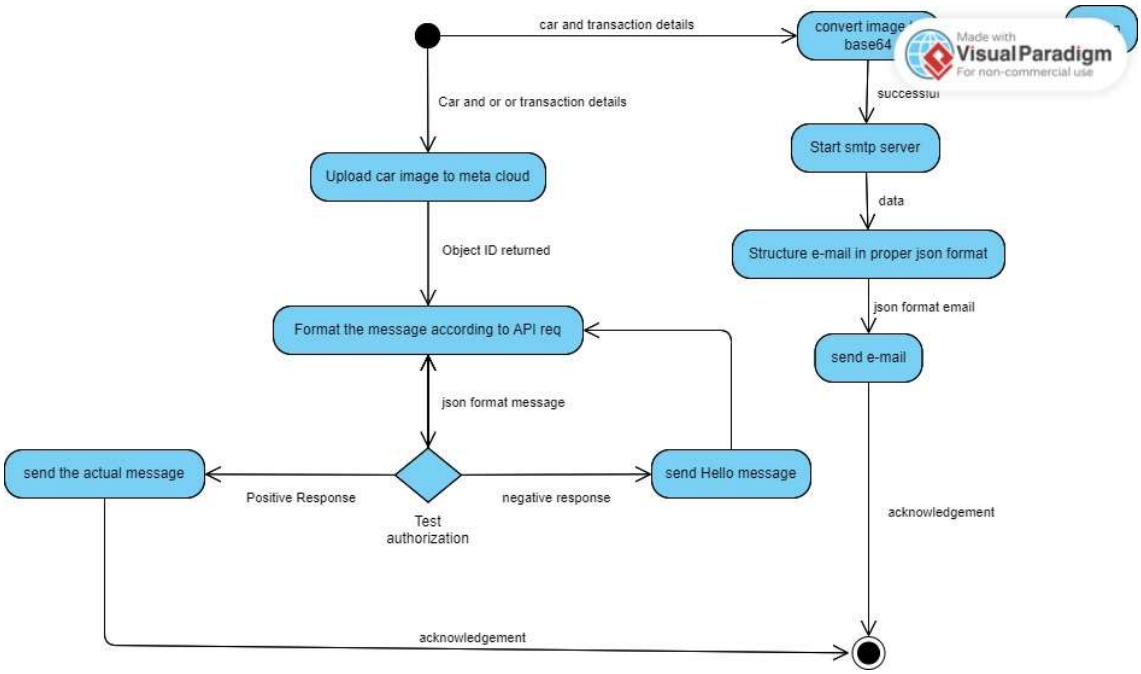
- **Start:** The process begins with the user selecting a car from the marketplace.
- **Decision Point:** The user decides whether to book a test drive or make a purchase.

- **Test Drive:**
  - The user selects a date and time for the test drive.
  - The system checks if the slot is free.
  - If available, the appointment is scheduled in the database.
- **Purchase:**
  - The user adds the car to the wishlist and then chooses to buy it.
  - The system redirects to the wishlist where the user confirms the amount.
  - **Payment Method:**
    - **Card Payment:**
      - The user enters card details.
      - If the transaction is successful, WhatsApp and email activities are triggered.
    - **Crypto Payment:**
      - The user connects a Metamask wallet.

- If successful, the transaction is confirmed, and smart contract interaction occurs.
- The system fetches a GIF from an API using a keyword, displays the transaction, and provides a download receipt.

- **End:** The process concludes with the user receiving confirmation and transaction details.

6.1.3. WhatsApp and E-mail:



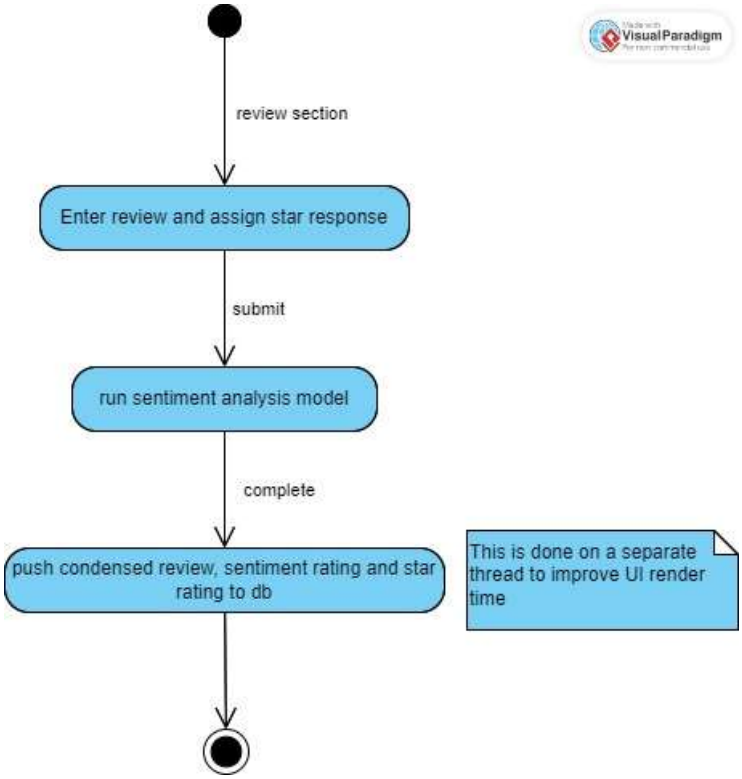
- **Start:** The process begins with car and transaction details.
- **Image Conversion:** The car image is converted into a base64 format.

- **Successful Conversion:** The process proceeds if the conversion is successful.

- **Path 1:**
  - The car image is uploaded to a meta cloud.
  - An Object ID is returned.
  - The message is then formatted according to the API requirements using the returned Object ID.
  - **Test Authorization:**
    - **Positive Response:** If authorization is successful, the actual message is sent.
    - **Negative Response:** The process ends without sending the message.
- **Path 2:**
  - An SMTP server is started.
  - The e-mail is structured in a proper JSON format.
  - The e-mail is sent, followed by an acknowledgement.
  - A "Hello" message is sent if needed.

- **End:** The process concludes with either sending the actual message (if authorized) or terminating if the test authorization fails.

6.1.3. Review and Sentiment Analysis:



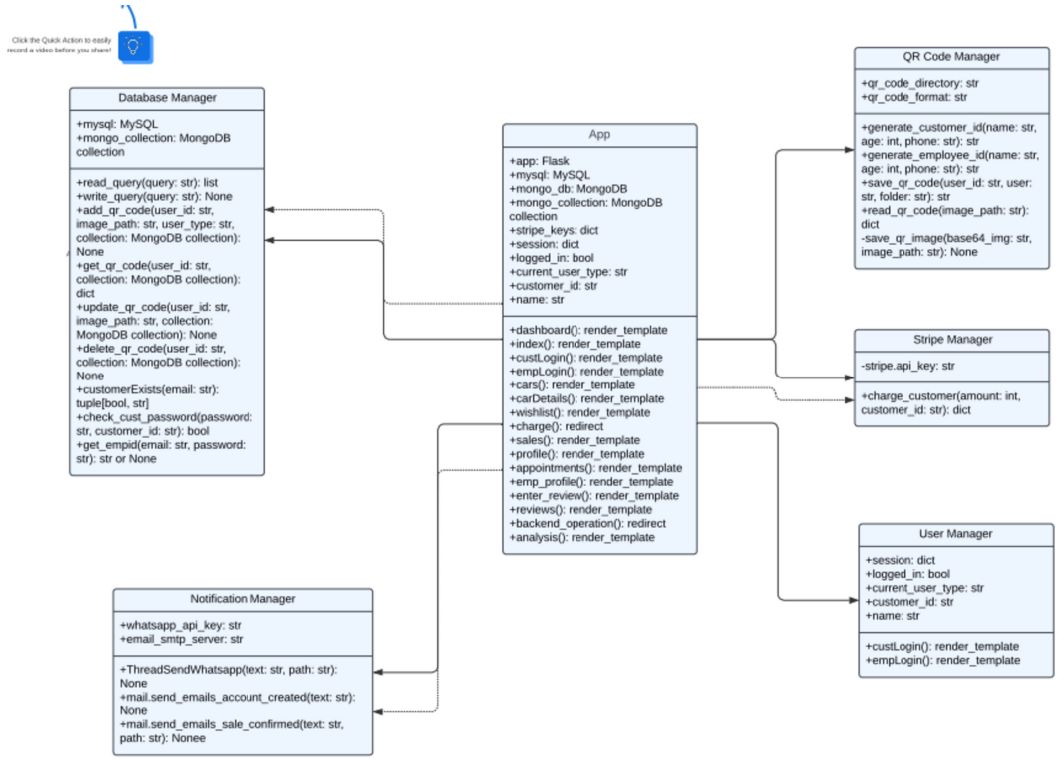
- **Start:** The process begins with the review section.
- **Enter Review:** The user enters a review and assigns a star rating.
- **Submit:** The review and star rating are submitted.
- **Run Sentiment Analysis:** A sentiment analysis model is run on the submitted review.
- **Push Data to Database:**After the sentiment analysis is complete, the condensed review, sentiment rating, and star rating are pushed to the database. This operation is performed on a separate thread to improve UI render time.

- **End:** The process concludes with the data successfully stored in the database.

Chapter 7

UML Class and Sequence Diagrams

7.1. UML Class Diagram:



Classes and their Responsibilities:

1. **Database Manager**

   - Manages the database connections for MySQL and MongoDB.
   - **Key Methods:**
     - **read_query(query: str):** Reads data from the database.
     - **write_query(query: str):** Writes data to the database.
     - **add_qr_code(user_id: str, image_path: str, user_type: str, collection: MongoDB collection):** Adds a QR code entry.
     - **get_qr_code(user_id: str, collection: MongoDB collection):** Retrieves a QR code.
     - **update_qr_code(user_id: str, image_path: str, collection: MongoDB collection):** Updates a QR code.
     - **delete_qr_code(user_id: str, collection: MongoDB collection):** Deletes a QR code.
     - **customerExists(email: str):** Checks if a customer exists.
     - **check_cust_password(password: str, customer_id: str):** Validates customer password.
     - **get_empid(email: str, password: str):** Retrieves employee ID based on credentials.

2. **App**

- Main application class using Flask framework.
- Handles user sessions, stripe keys, and routes for different functionalities.
- **Key Methods:**
    - **dashboard():** Renders the dashboard.
    - **custLogin(), empLogin():** Handles customer and employee login.
    - **cars(), carDetails(), wishlist(), sales(), appointments(), profile(), enter_review():** Renders different sections of the app.
    - **backend_operation(), analysis():** Redirects for backend operations.

3. **QR Code Manager**

- Manages QR code creation and handling.
- **Key Methods:**
    - **generate_customer_id(name: str, age: int, phone: str):** Generates a customer ID.
    - **generate_employee_id(name: str, age: int, phone: str):** Generates an employee ID.
    - **save_qr_code(user_id: str, user: str, folder: str):** Saves a QR code.
    - **read_qr_code(image_path: str):** Reads a QR code.
    - **save_qr_image(base64_img: str, image_path: str):** Saves a QR image.

4. **Stripe Manager**

- Manages payment processing via Stripe.
- **Key Methods:**
    - **charge_customer(amount: int, customer_id: str):** Charges a customer using Stripe.

5. **User Manager**

- Manages user-related data and sessions.
- **Key Properties:**
    - session, logged_in, current_user_type, customer_id, name.
- **Key Methods:**
    - **custLogin(), empLogin():** Handles user logins.

6. **Notification Manager**

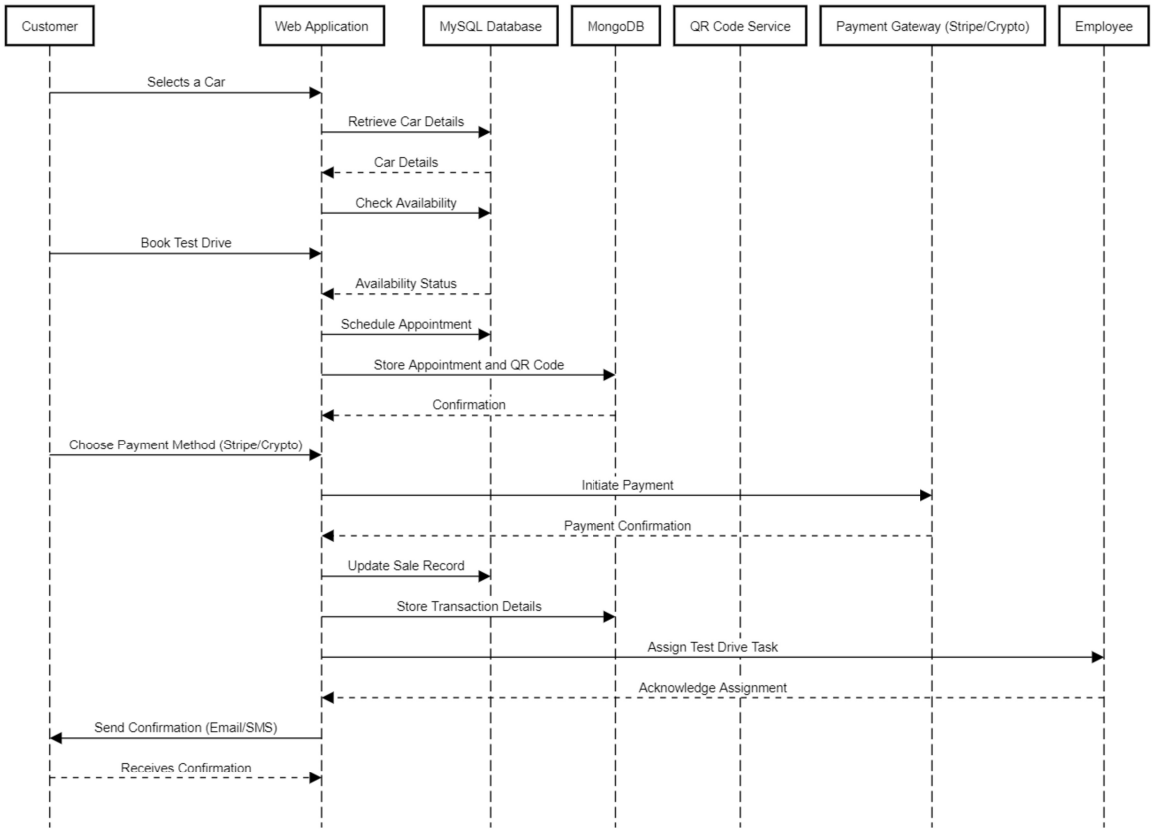
- Manages sending notifications via WhatsApp and email.
- **Key Methods:**
    - **ThreadSendWhatsApp(text: str, path: str):** Sends a WhatsApp message.
    - **email_smtp_server:** Handles email server operations.
    - **mail_send_emails_account_created(text: str):** Sends account creation emails.
    - **mail_send_emails_sale_confirmed(text: str, path: str):** Sends sales confirmation emails.

Relationships:

- The App class interacts with Database Manager, QR Code Manager, Stripe Manager, and User Manager to perform various operations like user management, payment processing, and data handling.
- The Notification Manager is used for sending out communications through WhatsApp or email, likely triggered by actions in the App class.

This UML class diagram provides an overview of the key components and their interactions in a system managing users, QR codes, payments, and notifications.

7.2. Sequence Diagram:



- **Customer Interaction:** The customer interacts with the web application by selecting a car and booking a test drive.

- **Backend Processes:**

  - The web application retrieves car details from the MySQL database.
  - It checks availability and schedules the test drive in the database.
  - QR codes and related data are stored in MongoDB.

- **Payment:**

  - The customer selects a payment method (Stripe or Crypto).

- The web application processes the payment through the chosen gateway and updates the databases accordingly.

- **Employee Assignment:** An employee is assigned to manage the test drive, and the web application confirms the task assignment.

- **Confirmation:** The customer receives confirmation of the test drive and payment through email or SMS.

Chapter 8

Data Flow Diagram

Fig 8.1 & 8.2 presents a series of Data Flow Diagrams (DFDs) designed to illustrate the information flow within the showroom management system. These DFDs provide a visual representation of how data is captured, processed, and utilized within the system, ensuring efficient operations and a seamless customer experience.

The DFDs are structured into two key levels:

- **Level 0 DFD (Context Diagram):** This high-level overview depicts the entire showroom management system as a single process and identifies its interaction with external entities, such as customers and suppliers. It focuses on the overall data exchange between the system and the outside world.

- **Level 1 DFD:** This level delves deeper into the system, breaking down the single process from the Level 0 DFD into more granular sub-processes. It illustrates how these sub-processes work together to manage various aspects of the showroom, including customer interaction, car inventory, appointments, sales transactions, and customer reviews. It details the data flow between these sub-processes, ensuring a clear understanding of how data is utilized within the system.

By analyzing these DFDs, stakeholders can gain valuable insights into the data flow within the showroom management system. This analysis can be used to:

- **Identify potential bottlenecks:** Detect areas where data processing might be slow or inefficient.

- **Optimize internal processes:** Streamline data exchange between sub-processes for improved performance.

- **Enhance data security:** Understand how data is accessed and stored to implement appropriate security measures.

- **Improve information accessibility:** Ensure relevant data is readily available to the appropriate personnel for informed decision-making.

These DFDs provide a foundational understanding of the data flow within the showroom management system, paving the way for further system development, optimization, and ultimately, a wellfunctioning and efficient environment for both customers and showroom staff.

8.1 DFD Level 0

The Level 0 DFD(Fig 8.1), also known as the context diagram, provides a high-level overview of the entire showroom management system. It focuses on the system itself and its interaction with external entities.

- **Process:** Showroom Management

- **External Entities:**

  1)Customers: Interact with the showroom to browse cars, schedule appointments, make purchases, and leave reviews.

  2)Suppliers: Provide cars for the showroom to sell.

- **Data Flows:**

  1)Inbound:

    a)Customer inquiries (e.g., car details, appointment requests)

    b)Car information from suppliers o Outbound:

  2)Appointment confirmations

    a)Sales information (car details, purchase details)

    b)Reviews (written feedback)

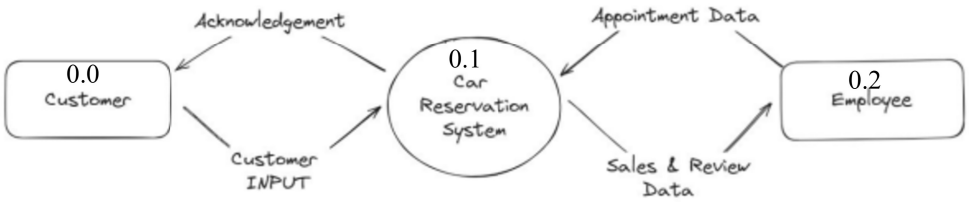


Fig 8.1

8.2. DFD Level 1:

The Level 1 DFD breaks down the single process from the Level 0 DFD into more detailed subprocesses. It shows how the showroom management system achieves its overall function.

- Process: Showroom Management o Sub-processes:
  - Customer Management:
    - Handles customer registration and profile updates.
    - Tracks customer appointments and purchase history.
  - Car Management:
    - Tracks car inventory, details, and specifications.

        - Maintains car maintenance history.
        - Updates car status (available, sold, etc.)
    - Appointment Management:
        - Schedules appointments for customers to test drive cars.
        - Links customers, employees, and cars for appointments.
        - Tracks appointment status (pending, confirmed, completed).
    - Sales Management:
        - Processes car sales transactions.
        - Records sales details (date, price, payment method, finance details).
        - Links sales with customers, employees, and cars involved.
    - Review Management:
        - Captures customer reviews for cars.
        - Stores review details (star rating, written feedback).
        - Links reviews with customers and cars.
- External Entities: Same as Level 0 DFD (Customers, Suppliers)
- Data Flows:
    - Between Sub-processes: (examples)
        - Customer Management -> Appointment Management: Customer information for scheduling appointments.
        - Car Management -> Sales Management: Car details for sales transactions.
        - Appointment Management -> Sales Management: Appointment completion triggers sales recording.
        - Customer Management -> Sales Management: Customer information for sales records.
        - Customer Management -> Review Management: Customer information for linking reviews.
        - Car Management -> Review Management: Car information for linking reviews.

- Between System and External Entities: (examples)
  - Customer inquiries -> Customer Management (Inbound)
  - Appointment confirmations -> Customers (Outbound)
  - Car information from Suppliers (Inbound) -> Car Management
  - Sales information -> Customers (Outbound) (e.g., invoices)
  - Reviews (written feedback) -> Customers (Outbound) (optional - confirmation or thank you)

This breakdown illustrates the internal functionalities of the showroom management system and how data flows between different sub-processes to achieve efficient customer service, car management, and sales operations. Remember, these are general examples, and the specific data flows and sub-processes can be further customized depending on the specific needs of the showroom.

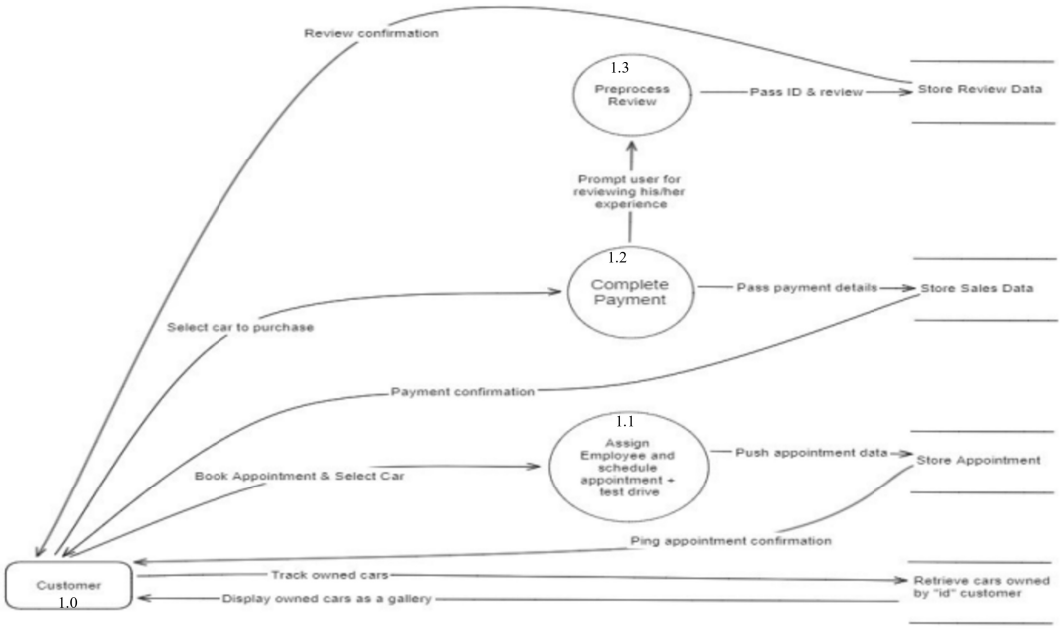


Fig 8.2

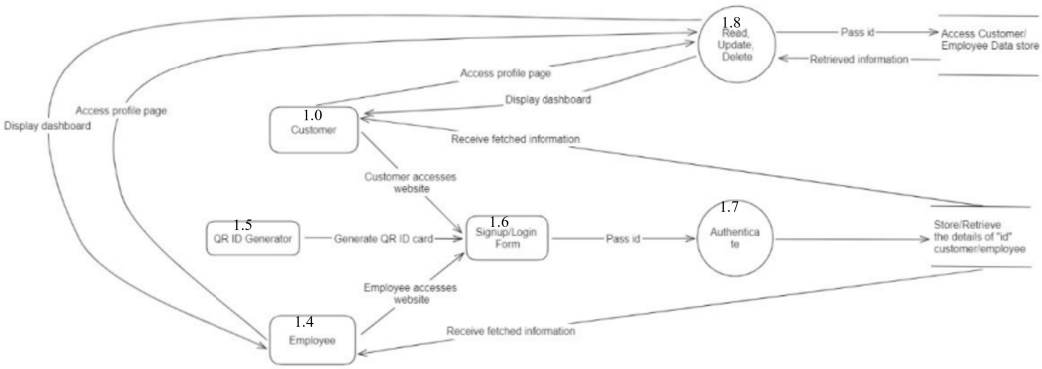

1.8
Read, Update, Delete
Pass id
Access Customer/ Employee Data store
Retrieved information
Access profile page
Display dashboard
Access profile page
Display dashboard
1.0
Customer
Receive fetched information
Customer accesses website
1.5
QR ID Generator
Generate QR ID card
1.6
Signup/Login Form
Pass id
1.7
Authenticate
Store/Retrieve the details of "id" customer/employee
Employee accesses website
1.4
Employee
Receive fetched information

Fig 8.3

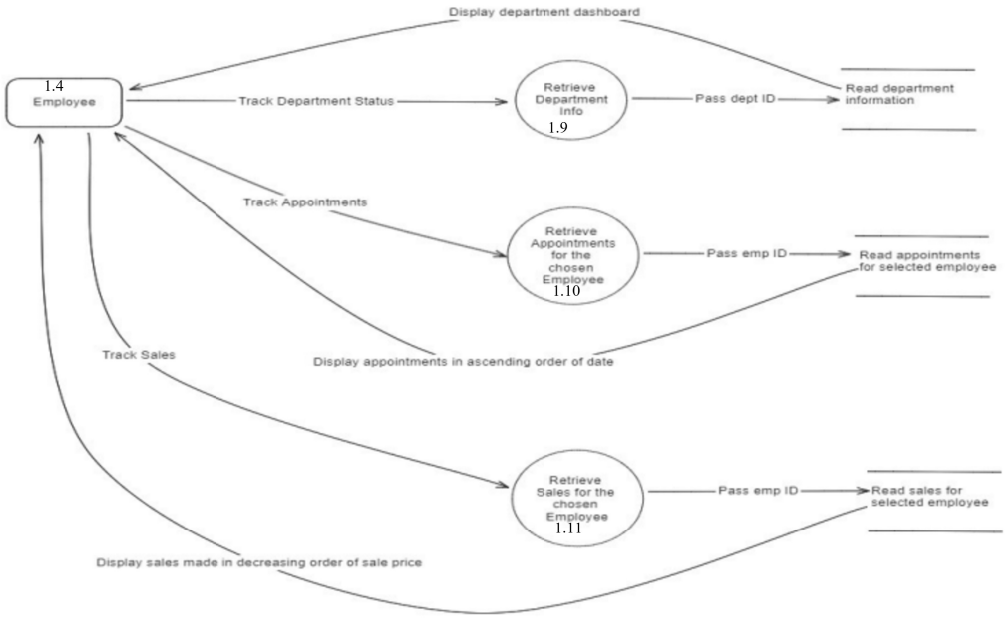

Display department dashboard
1.4
Employee
Track Department Status
Retrieve Department Info
1.9
Pass dept ID
Read department information
Track Appointments
Retrieve Appointments for the chosen Employee
1.10
Pass emp ID
Read appointments for selected employee
Track Sales
Display appointments in ascending order of date
Retrieve Sales for the chosen Employee
1.11
Pass emp ID
Read sales for selected employee
Display sales made in decreasing order of sale price

Fig 8.4

Chapter 9

Designing test suites

Test Password Hashing & Managing using bcrypt Utility:

Title	Description	Steps	Pre-condition	Post-condition	Expected Output
Test Hash Password Length	Verify that the hashed password length is 60 characters	1. Set password to "test_password". 2. Call hash_password(password). 3. Assert the length of hashed_password is 60.	hash_password function is implemented	None	Hashed password is 60 characters long.
Test Hash Password Type	Ensure that the hashed password is of type string	1. Set password to "test_password". 2. Call hash_password(password). 3. Assert the type of hashed_password is str.	hash_password function is implemented	None	Hashed password is a string.
Test Hash Password Consistency	Verify that the same password generates different hashes	1. Set password to "test_password". 2. Call hash_password(password) twice. 3. Assert that hashed_password1 is not equal to hashed_password2.	hash_password function uses salting	None	Different hashes are generated for the same password due to salting.
Test Check Password Correct	Confirm that the correct password matches the hashed password	1. Set password to "test_password". 2. Call hash_password(password). 3. Call check_password(password, hashed_password).	hash_password and check_password functions are implemented	None	Correct password matches the hashed password.

		4. Assert that it returns True.			
Test Check Password Incorrect	Confirm that an incorrect password does not match the hashed password	1. Set password to "test_password". 2. Set wrong_password to "wrong_password". 3. Call hash_password(password). 4. Call check_password(wrong_password, hashed_password). 5. Assert that it returns False.	hash_password and check_password functions are implemented	None	Incorrect password does not match the hashed password.

Test HuggingFace Sentiment Analysis & Text Summarization Utility:

Title	Description	Steps	Pre-condition	Post-condition	Expected Output
Test Sentiment Analysis Positive	Verify that the sentiment analysis correctly identifies positive sentiment	1. Set text to "I love this product.". 2. Call sentiment_analysis(text). 3. Assert that the label is 'POSITIVE'. 4. Assert that the score is greater than 0.9.	sentiment_analysis function is implemented	None	Sentiment is classified as POSITIVE with a confidence score > 0.9.
Test Sentiment Analysis Negative	Verify that the sentiment analysis correctly identifies negative sentiment	1. Set text to "I hate this product.". 2. Call sentiment_analysis(text). 3. Assert that the label is 'NEGATIVE'. 4. Assert that the score is greater than 0.9.	sentiment_analysis function is implemented	None	Sentiment is classified as NEGATIVE with a confidence score > 0.9.
Test Summarize Text Long Input	Ensure that the text summarization produces a shorter summary than the input	1. Set text to a long text input. 2. Call summarize_text(text). 3. Assert that the word count of result['summary_text'] is less than the word count of the original text.	summarize_text function is implemented	None	Summary is shorter than the original input text.

Test QR Generator Utility:

Title	Description	Steps	Pre-condition	Post-condition	Expected Output
Test Generate Employee ID Format	Verify that the employee ID format is correct	1. Set name to "Alice", age to 28, gender to "Female", and salary to 60000. 2. Call generate_employee_id(name, age, gender, salary). 3. Assert the format of employee_id using regex.	generate_employee_id function is implemented	None	Employee ID follows the format alice_F_28_60k_XXXX where XXXX is a 4-digit number.
Test Generate Customer ID Format	Verify that the customer ID format is correct	1. Set name to "Bob", age to 30, and phone to "9876543210". 2. Call generate_customer_id(name, age, phone). 3. Assert the format of customer_id using regex.	generate_customer_id function is implemented	None	Customer ID follows the format bob_3210_XXXX where XXXX is a 4-digit number.
Test Save QR Code Employee	Ensure that the QR code for an employee is saved correctly	1. Set employee_id to "alice_F_28_60k_1234". 2. Set folder to "test/QR_ID_Employee". 3. Call save_qr_code(employee_id, user='E', folder=folder). 4. Assert that the file exists. 5. Assert the image path. 6. Remove the file.	save_qr_code function is implemented and folder exists	None	QR code image is saved at the correct path test/QR_ID_Employee/alice_F_28_60k_1234.png.
Test Save QR Code Custo	Ensure that the QR code for a customer is saved	1. Set customer_id to "bob_3210_1234". 2. Set folder to "test/QR_ID_Custo	save_qr_code function is implemented and	None	QR code image is saved at the correct path test/QR_ID_Customer/bob_3210_1234.png.

mer	correctly	mer". 3. Call save_qr_code(customer_id, user='C', folder=folder). 4. Assert that the file exists. 5. Assert the image path. 6. Remove the file.	folder exists		

Test QR Reader Utility:

#	Title	Description	Steps	Pre-condition	Post-condition	Expected Output
1.	Test Read QR Code Employee	Verify that the QR reader correctly reads employee QR data	1. Set up the employee QR image path. 2. Call read_qr_code(self.employee_qr_image). 3. Assert that qr_data starts with "bob".	QR code for employee exists at self.employee_qr_path	None	QR data starts with the name "bob".
2.	Test Read QR Code Customer	Verify that the QR reader correctly reads customer QR data	1. Set up the customer QR image path. 2. Call read_qr_code(self.customer_qr_image). 3. Assert that qr_data starts with "charlie".	QR code for customer exists at self.customer_qr_path	None	QR data starts with the name "charlie".
3.	Test Read QR Code No Data	Verify that the QR reader returns empty data for a blank QR image	1. Create a blank QR image at empty_image_path. 2. Call read_qr_code(empty_image_path). 3. Assert that qr_data is an empty string. 4. Remove the image.	None	None	QR data is an empty string if no QR code is present in the image.
4.	Test Read QR Code Invalid Path	Ensure that the QR reader raises an error when given an invalid path	1. Call read_qr_code("invalid_path.png"). 2. Assert that a TypeError is raised with the message "cannot unpack non-iterable NoneType object".	None	None	A TypeError is raised when an invalid path is provided.

Test Save QR Image Utility:

#	Title	Description	Steps	Pre-condition	Post-condition	Expected Output
1.	Test Save QR Image	Verify that the QR image is saved correctly	1. Set up a base64 encoded image string. 2. Call save_qr_image(self.sample_base64_img, self.image_path). 3. Assert that the image file exists at self.image_path.	sample_base64_img is a valid base64 string	None	The image file exists at the specified path after saving.
2.	Test Save Multiple QR Images	Verify that multiple QR images can be saved correctly	1. Loop through 5 iterations. 2. In each iteration, set image_path to a unique path. 3. Call save_qr_image(self.sample_base64_img, image_path). 4. Assert that the image file exists at each path.	sample_base64_img is a valid base64 string	None	Each image file exists at the specified path after saving.
3.	Test Save QR Image No Path	Ensure that saving an image without a path raises an error	1. Call save_qr_image(self.sample_base64_img, None). 2. Assert that a TypeError is raised.	sample_base64_img is a valid base64 string	None	A TypeError is raised when the path is None.
4.	Test Image Content	Verify that the content of the saved image matches the base64 datam	1. Call save_qr_image(self.sample_base64_img, self.image_path). 2. Open the saved image file in binary mode. 3. Assert that the file content matches the decoded base64 string.	sample_base64_img is a valid base64 string	None	The saved image content matches the original base64 data.

Chapter 10

Conclusion & Future Enhancement

Project Conclusion: A Step Forward in Modern Car Showroom Management
This project successfully developed a web-based car showroom management system that leverages a combination of Flask, MySQL, and NoSQL databases. The system offers comprehensive functionalities for managing car listings, customers, test drives, sales, and employee incentives. Prioritizing user experience, the accessible web interface and unique QR code generation for customer identification streamline interactions. Additionally, the integration of two secure payment gateways, Stripe and a blockchain option, provides flexibility and caters to diverse customer preferences.

The implemented review system empowers customers to provide feedback, fostering a customer-centric approach. Furthermore, employee functionalities for managing sales data and tracking incentives promote a results-oriented work environment.

Key Achievements:

- Streamlined showroom operations through efficient data management.
- Enhanced customer experience with a user-friendly interface and diverse payment options.
- Empowered customers through feedback mechanisms.
- Motivated employees with incentive tracking features.
- Incorporated a crypto based payment gateway with gif-based transaction storage to gamify the payment experience on 5irechain and SHM.
- Incorporated a concept of downloadable payment receipts.
- Incorporated a QR code-based customer authentication system.

Future Enhancements:

This project serves as a foundation for a modern car showroom management system. Future enhancements could include:

- Integrating advanced inventory management functionalities.
- Developing in-depth financial reporting capabilities.
- Expanding the system to include direct car service scheduling.
- Expanding the payment options to incorporate more mainstream cryptocurrencies such as bitcoin, Ethereum and worldcoin.
- Including a admin login to provide complete control of the database from the website.

By continuously improving upon the core functionalities and incorporating evolving technologies, this system has the potential to significantly improve efficiency, customer satisfaction, and employee engagement within the car showroom industry.

References

[1] Flask, "Flask Documentation," 3.0.x, [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/.

[2] phpMyAdmin, "phpMyAdmin Documentation," [Online]. Available: https://www.phpmyadmin.net/docs/.

[3] MongoDB, "MongoDB Documentation," [Online]. Available: https://www.mongodb.com/docs/.

[4] PyPI, "qrcode Documentation," [Online]. Available: https://pypi.org/project/qrcode/.

[5] Solidity, "Solidity Documentation," v0.8.24, [Online]. Available: https://docs.soliditylang.org/en/v0.8.24/.

[6] 5ire, "5ire Documentation," [Online]. Available: https://www.5ire.org/documentation.

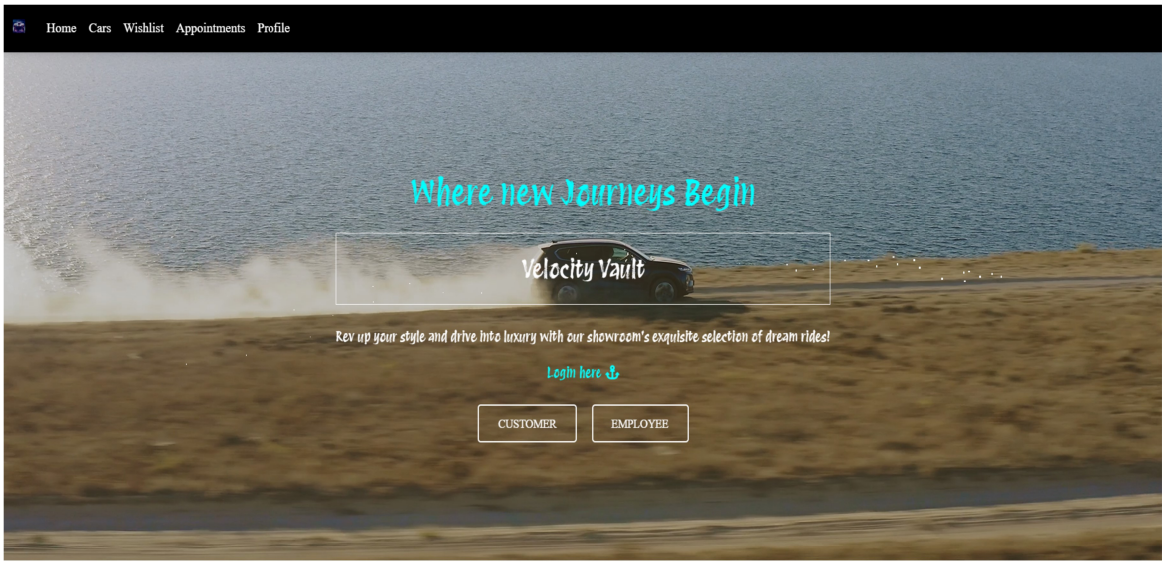
[7] "Car Showroom Management System," [Online]. Available: https://www.scribd.com/document/337853350/car-Showroom-Management-System-doc.

[8] Hardvan, "DBMS-Lab-EL," GitHub. [Online]. Available: https://github.com/Hardvan/DBMS-Lab-EL.

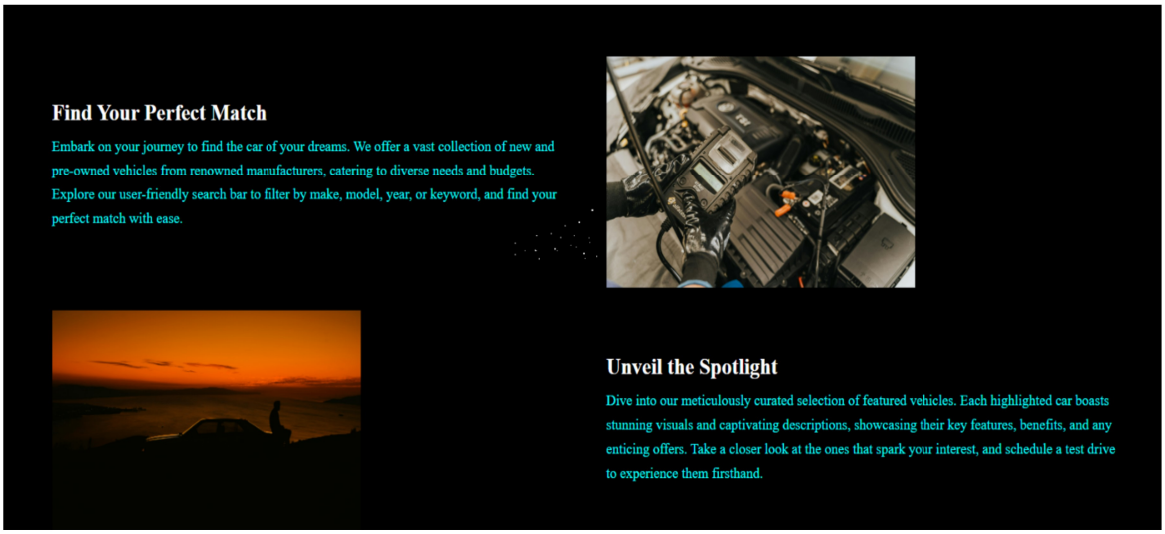
Appendix

Screenshots with descriptions

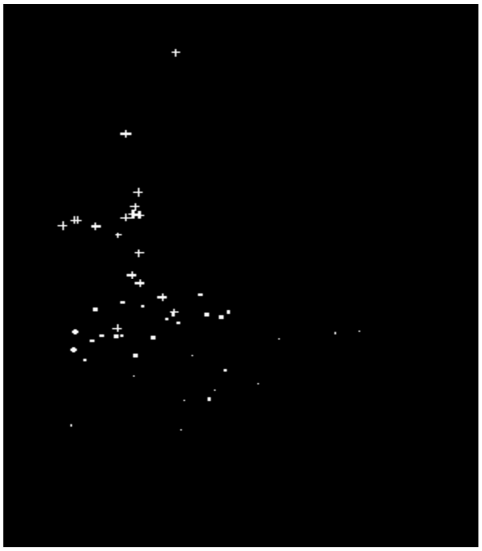
Dashboard Section: This section represents the landing page of the website. The user can choose to either sign-up/login as a customer or employee and the navbar changes accordingly.

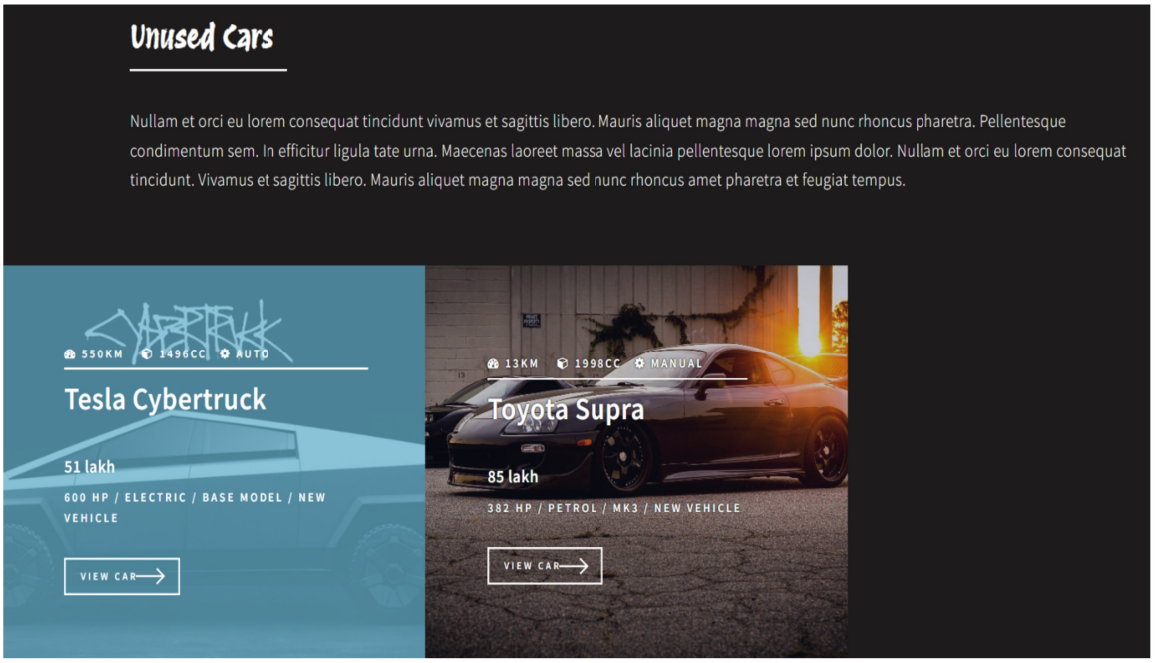


The lower half of the dashboard gives a description of our vision and highlights.



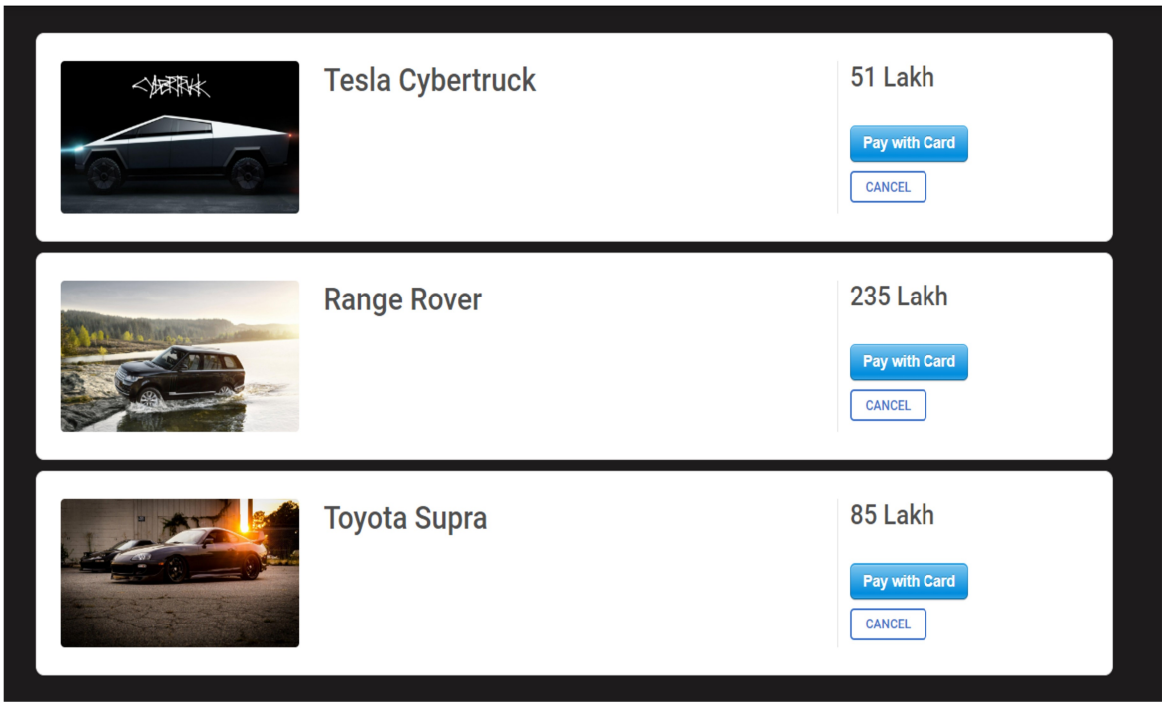
Innovative Frontend Features: The website uses a JavaScript based mouse graphic feature that drops glittery stars whenever the mouse is moved providing an aesthetic appearance and fun experience to users. In addition, login checks are implemented in the backend which display alerts whenever an unauthorized access is attempted.



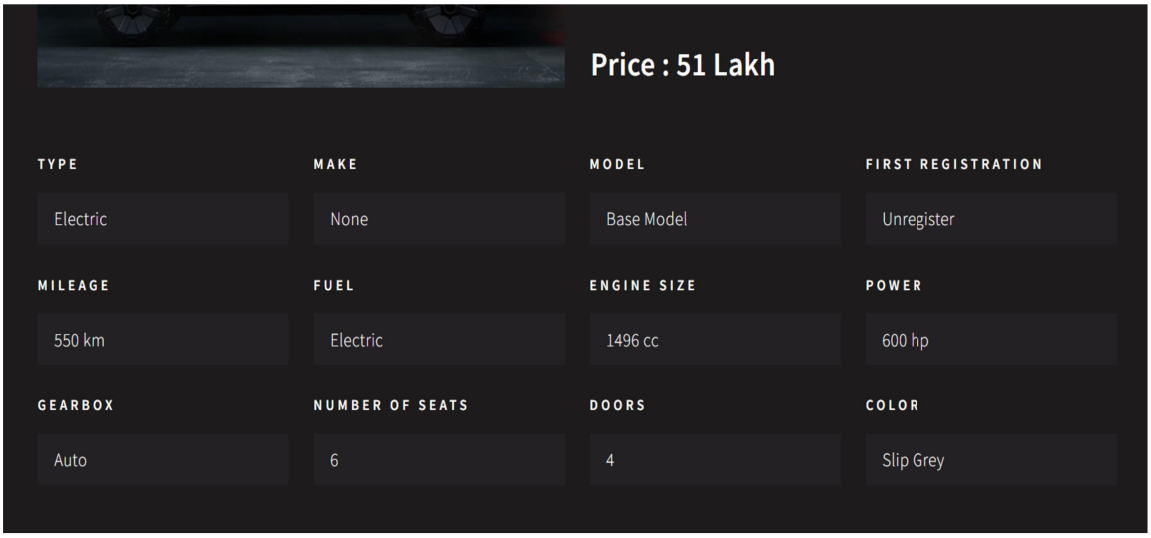


Cars Section: This section contains well designed card representations of the cars available in the inventory in the form shown below. More information about a specific car can be accessed by clicking the view car button.

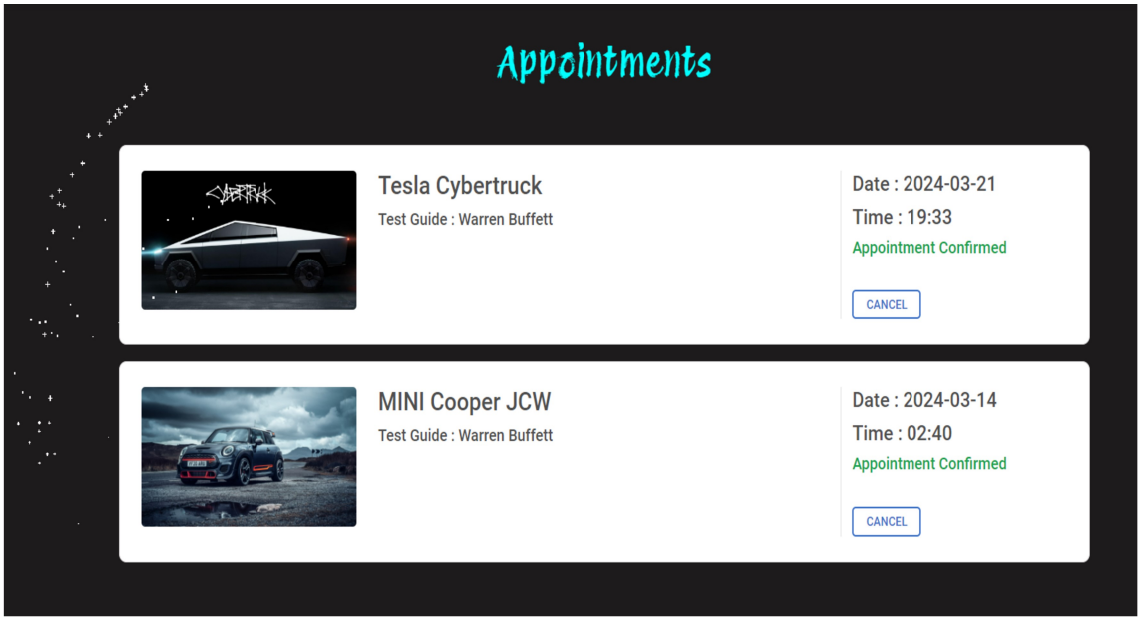
Wishlist Section: If a user likes a car, he/she can add it to the Wishlist section. The user can later access his Wishlist from the navbar to choose to buy/cancel the car.



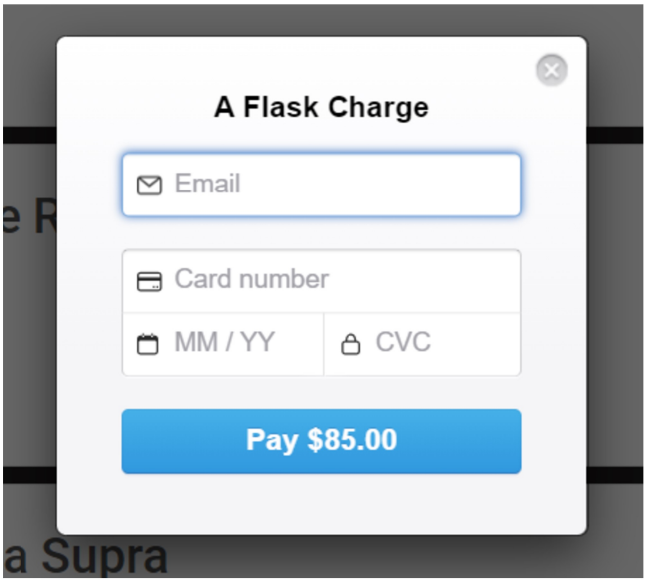
Car Details Section: This section displays the full details of the chosen car from the engine type, mileage, fuel type and so on. It also allows a customer to book an appointment at a chosen date.



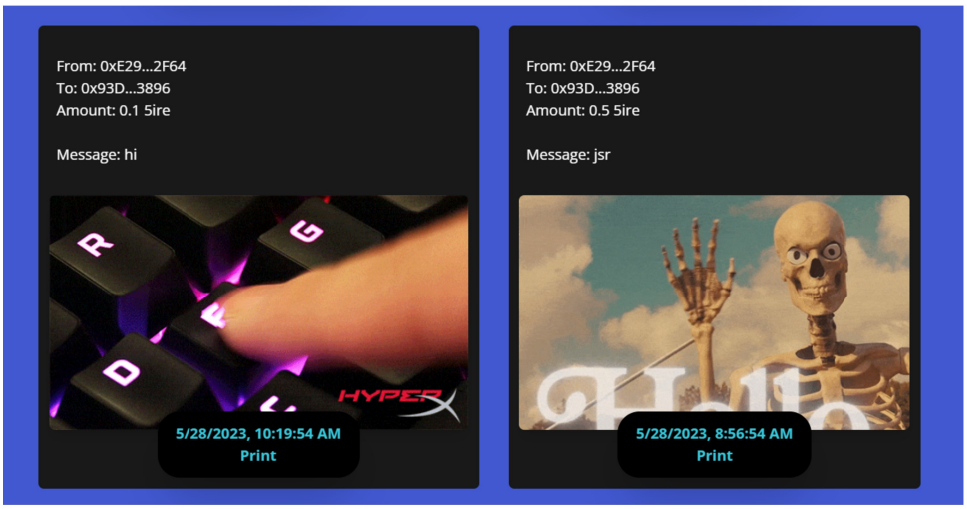
Once an appointment is booked, it can be viewed by the customer in the appointments section with a employee being allotted to the customer based on our randomizing function in the backend.

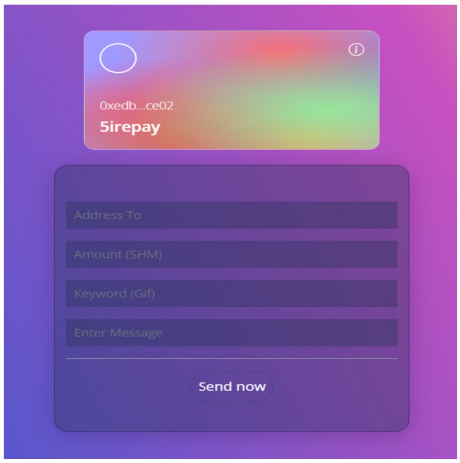


Payment by Stripe: Stripe API is used to accept card payments for any product on the website.



Payment in crypto: A gamified crypto payment gateway with immutable receipt generation for each transaction is integrated for an innovative feature in the project.


From: 0xE29...2F64
To: 0x93D...3896
Amount: 0.1 5ire
Message: hi
5/28/2023, 10:19:54 AM
Print
From: 0xE29...2F64
To: 0x93D...3896
Amount: 0.5 5ire
Message: jsr
5/28/2023, 8:56:54 AM
Print


0xedb...ce02
5irepay
Address To
Amount (SHM)
Keyword (Gif)
Enter Message
Send now