

# **CAR EXPENSE TRACKER**

## **MICRO PROJECT REPORT**

Submitted by

**KAVIYA B**

**(23ITR083)**

**JANISHAA V**

**(23ITR067)**

**NAVEENA S**

**(23ITR110)**

*in partial fulfilment of the requirements*

*for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**



**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI ERODE – 638 060**

**MAY 2025**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI ERODE – 638060**

**MAY 2025**

**BONAFIED CERTIFICATE**

This is to certify that the Project report entitled **CAR EXPENSE TRACKER** is the Bonafide record of project work done by **KAVIYA B (23ITR083)**, **JANISHAA V (23ITR067)** and **NAVEENA S (23ITR110)** for **22ITT42 WEB TECHNOLOGY** during the year 2024–2025.

**COURSE IN CHARGE**

**HEAD OF THE DEPARTMENT**

**(Signature with seal)**

Date:

Submitted for the final viva voce examination held on \_\_\_\_\_ .

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI ERODE – 638060**

**MAY 2025**

**DECLARATION**

We affirm that the Project Report titled **CAR EXPENSE TRACKER** being submitted in partial fulfilment of the requirements for the award of Bachelor of Technology is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Date:**

**KAVIYA B**  
**(23ITR083)**

**JANISHAA V**  
**(23ITR067)**

**NAVEENA S**  
**(23ITR110)**

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Date:

Name and Signature of the course in charge

## **ABSTRACT**

The Car Expense Tracker is a smart web-based application developed to help vehicle owners efficiently monitor and manage all car-related expenses with ease and clarity. Many individuals struggle to keep a detailed record of ongoing costs such as fuel, maintenance, insurance, and repairs, often leading to budgeting issues or missed service deadlines. This project addresses those concerns by offering a centralized digital platform that streamlines the process of logging expenses, setting payment reminders, and analyzing financial data related to vehicle ownership.

Through a secure user authentication system, individuals can register, log in, and maintain a personalized dashboard where they can input their car expenses categorized by type and frequency. The application supports features like setting alerts for important due dates—such as insurance renewals or scheduled maintenance—and notifies users via email or in-app messages. Additionally, the system offers powerful analytical tools that generate comprehensive reports, charts, and graphs, enabling users to visualize their spending trends and make informed decisions.

The platform is built using modern technologies including HTML, CSS, and Bootstrap for an intuitive frontend design, with backend support from Node.js and Django for robust data handling. MySQL is used for efficient and secure database management. Designed with mobile responsiveness in mind, the interface ensures a seamless experience across devices. By offering a structured and digital approach to vehicle expense management, the Car Expense Tracker encourages financial discipline and ensures that owners are always informed about their car's upkeep. It minimizes the chances of missed payments, over-expenditure, or forgotten maintenance, ultimately extending the life of the vehicle and saving long-term costs. In conclusion, this application is not only a budgeting tool but also a digital companion that promotes responsible vehicle ownership and smarter financial planning.

## ACKNOWLEDGEMENT

First and foremost, we acknowledge the abundant grace and presence of Almighty throughout different phases of the project and its successful completion.

We wish to express our gratefulness to our beloved Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.,** and all the trust members of Kongu Vellalar Institute of Technology Trust for providing all the necessary facilities to complete the project successfully.

We express our deep sense of gratitude to our beloved Principal **Dr.V.BALUSAMY B.E.(Hons)., M.Tech., Ph.D.,** for providing us an opportunity to complete the project.

We express our gratitude to **Dr. S. ANANDAMURUGAN M.E., Ph.D.,** Head of the Department, Department of Information Technology for his valuable suggestions.

We are highly indebted to **Mrs. R. AARTHI B.E., M.E.,** Department of Information Technology for her valuable supervision and advice for the fruitful completion of the project.

We are thankful to the faculty members of the Department of Information Technology for their valuable guidance and support.

## TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE No
	<b>ABSTRACT</b>	<b>iv</b>
	<b>LIST OF FIGURES</b>	<b>viii</b>
	<b>LIST OF ABBREVIATIONS</b>	<b>ix</b>
<b>1.</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 INTRODUCTION	1
	1.2 OBJECTIVE	1
<b>2.</b>	<b>SYSTEM SPECIFICATION</b>	<b>2</b>
	2.1 HARDWARE REQUIREMENTS	2
	2.2 SOFTWARE REQUIREMENTS	2
	2.3 SOFTWARE DESCRIPTION	3
	2.3.1 Visual Studio Code	3
	2.3.2 MongoDB Compus	3
	2.3.3 Express.js	4
	2.3.4 Mongoose	4
	2.3.5 Body-parser	4
	2.3.6 Connect-Mongoose	5
	2.3.7 CORS	5
	2.3.8 Dotenv	5
	2.3.9 Bcrypt.js	6
	2.3.10 Angular	6

<b>3.</b>	<b>SYSTEM DESIGN</b>	<b>7</b>
	3.1 USE CASE DIAGRAM	7
	3.2 CLASS DIAGRAM	8
	3.3 SEQUENCE DIAGRAM	9
	3.4 ACTIVITY DIAGRAM	10
	3.5 DATABASE DESIGN	11
	3.5.1 Overview	11
	3.5.2 Sign Up Schema	11
	3.5.3 Login Schema	12
	3.6 MODULES DESCRIPTION	13
	3.6.1 Authentication Module	13
	3.6.2 Expense Module	13
	3.6.2 Donation Module	14
<b>4.</b>	<b>RESULTS</b>	<b>15</b>
<b>5.</b>	<b>CONCLUSION AND FUTURE WORK</b>	<b>16</b>
	<b>APPENDIX 1- CODING</b>	<b>18</b>
	<b>APPENDIX 2-SNAPSHOTS</b>	<b>29</b>
	<b>REFERENCES</b>	<b>33</b>

## LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
3.1	USE CASE DIAGRAM	7
3.2	CLASS DIAGRAM	8
3.3	SEQUENCE DIAGRAM	9
3.4	ACTIVITY DIAGRAM	10
A2.1	HOME PAGE	29
A2.2	LOGIN PAGE	30
A2.3	REGISTRATION PAGE	30
A2.4	MAINTENANCE PAGE	31
A2.5	EXPORT PAGE	31
A2.6	REFUEL PAGE	32
A2.7	EXPENSE TRACKER PAGE	32
A2.8	ADD CAR PAGE	33



## LIST OF ABBREVIATIONS

<b>CSS</b>	Cascading Style Sheets
<b>HTML</b>	HyperText Markup Language
<b>ID</b>	Identification
<b>JS</b>	JavaScript
<b>JSX</b>	JavaScript XML
<b>NFC</b>	Near-Field Communication
<b>QR</b>	Quick Response
<b>URL</b>	Uniform Resource Locator

## CHAPTER 1

### INTRODUCTION

#### 1.1 INTRODUCTION

Managing car-related expenses can become increasingly complex over time, especially with recurring costs such as fuel, servicing, insurance, and unexpected repairs. For many vehicle owners, keeping track of these expenses through manual records or basic spreadsheets often leads to missed payments, budgeting errors, or lack of clarity on overall spending. As personal and commercial transportation becomes a central part of modern life, there is a growing need for a digital solution that simplifies expense tracking while promoting financial awareness. Whether you're an individual managing a single car or a small business handling a fleet, this tool helps you stay on top of your spending, plan for upcoming costs, and make informed financial decisions. By replacing traditional methods with a centralized, digital approach, the tracker not only enhances accuracy and organization but also empowers users to maintain their vehicles more effectively and economically.

#### 1.2 OBJECTIVE

The main objective of the **Car Expense Tracker** is to offer a centralized and user-friendly digital platform that enables vehicle owners to efficiently record, manage, and analyze their car-related expenses. This system is designed to simplify the tracking of recurring and one-time costs such as fuel, maintenance, repairs, and insurance, ensuring that users maintain accurate and organized financial records. It aims to support timely vehicle upkeep by providing automated reminders and alerts for important events like service appointments and insurance renewals. Additionally, the tracker generates detailed reports and visual insights to help users understand their spending patterns and make informed budgeting decisions. With secure user authentication and a responsive design, the platform ensures that each user's data is protected and accessible across multiple devices. Overall, the Car Expense Tracker promotes better financial control, reduces the risk of missed payments, and encourages responsible vehicle ownership.

## CHAPTER 2

### SYSTEM SPECIFICATION

#### 2.1 HARDWARE SPECIFICATION

<b>Processor</b>	:	12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz
<b>Processor Speed</b>	:	1.30 GHz
<b>RAM</b>	:	8.00 GB
<b>Hard Disk</b>	:	512GB
<b>Keyboard</b>	:	Standard 104 enhanced
<b>Mouse</b>	:	Local PS/2

#### 2.2 SOFTWARE REQUIREMENTS

<b>Platform</b>	:	Visual Studio Code
<b>Language</b>	:	HTML, CSS, Bootstrap, Javascript, Node js
<b>Database</b>	:	Mongo DB
<b>Library</b>	:	Express.js, Mongoose, Body- Parser, Cors,Connect-Mongo,Dotenv,Bcrypt.js
<b>Framework</b>	:	Angular

## **2.3 SOFTWARE DESCRIPTION**

### **2.3.1 Visual Studio Code**

Visual Studio Code (VS Code) is a lightweight and powerful Integrated Development Environment (IDE) used for building the Car Expense Tracker application. It is an ideal choice for web development due to its support for multiple programming languages and rich features. VS Code provides intelligent code completion through IntelliSense, which helps developers write accurate code faster by suggesting variables, methods, and functions. Its integrated debugging tools enable developers to set breakpoints, inspect variables, and step through code, facilitating quick issue identification and resolution. Additionally, VS Code integrates seamlessly with Git, allowing for efficient version control management directly within the IDE. It also supports a variety of extensions such as ESLint, Prettier, and live-server, which enhance productivity by automating tasks like code formatting and running local servers. The built-in terminal further streamlines the workflow by allowing developers to execute commands, install dependencies, and test the application without leaving the IDE. With its clean, user-friendly interface and cross-platform compatibility (Windows, macOS, Linux), Visual Studio Code is a crucial tool that supports efficient development, debugging, and testing of the Car Expense Tracker application.

### **2.3.2 MongoDB Compass**

**MongoDB Compass** is a graphical user interface (GUI) for interacting with MongoDB, the NoSQL database used in the Car Expense Tracker project. MongoDB Compass allows developers and database administrators to visualize, manage, and optimize the database without needing to write complex queries in the command-line interface. It provides an intuitive and user-friendly interface to explore the database's collections, documents, and fields. Through Compass, users can view real-time data, filter, and query collections with ease, and gain insights into database performance through built-in analytics tools. MongoDB Compass also offers schema exploration features, enabling developers to understand the structure of stored data and make necessary adjustments to the schema when required. Additionally, it simplifies data

migration tasks and offers features like indexing, aggregation, and visual explain plans to improve query performance. This tool plays a crucial role in managing and optimizing the database for the Car Expense Tracker application, ensuring efficient data storage, retrieval, and maintenance.

### 2.3.3 Express.js

**Express.js** is a minimal and flexible web application framework for Node.js, designed to simplify the process of building web applications and APIs. In the Car Expense Tracker project, Express.js is used to handle HTTP requests, route incoming data, and manage server-side logic. Express makes it easy to set up middleware to process requests, handle errors, and ensure smooth communication between the client-side and server-side components. It is lightweight and allows for building RESTful APIs with ease, which is crucial for managing donation and expense records.

### 2.3.4 Mongoose

**Mongoose** is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward way to define the structure of your data, create schemas, and interact with the MongoDB database. In the Car Expense Tracker project, Mongoose is used to define models for various entities like donors, donations, and expenses. It also helps in querying the database efficiently and handling CRUD operations (Create, Read, Update, Delete) seamlessly. By using Mongoose, developers can ensure that the data is well-structured and validated before being stored in the database.

### 2.3.5 Body-Parser

**Body-Parser** is middleware for Express.js that simplifies the extraction of data from incoming HTTP requests. It allows for easy parsing of JSON and URL-encoded form data in request bodies, which is essential for processing form submissions and API requests. In the Car Expense Tracker, Body-Parser is used to handle POST requests containing donor information, donation data, and other form submissions.

This library ensures that the server can properly interpret the incoming data and use it for various operations, such as storing donations or updating expense records.

### 2.3.6 Cors

**Cors** (Cross-Origin Resource Sharing) is a Node.js package used to enable cross-origin requests. When building a web application that interacts with APIs hosted on a different domain, CORS is required to allow the frontend to access resources on the backend securely. In the Car Expense Tracker project, **Cors** ensures that the frontend (running on one domain) can safely send HTTP requests to the backend server (which may be running on a different domain or port). This prevents security issues while enabling communication between different parts of the application.

### 2.3.7 Connect-Mongo

**Connect-Mongo** is a MongoDB session store for **Express.js** and **Connect**, which allows session data to be stored in a MongoDB database instead of in memory or in cookies. This is particularly useful for applications with many users, as it allows session data to be persistent across different requests. In the Car Expense Tracker, **Connect-Mongo** is used to manage user sessions, such as keeping users logged in after they submit donations or log in. It ensures that session data is securely stored in MongoDB, allowing for easy session management and better scalability.

### 2.3.8 Dotenv

**Dotenv** is a zero-dependency module that loads environment variables from a `.env` file into `process.env`. This allows you to store sensitive information such as database credentials, API keys, and secret keys in a secure and easily accessible way. In the Car Expense Tracker project, **Dotenv** is used to manage environment-specific settings, such as the MongoDB connection string, JWT secrets, and other sensitive configuration values. By keeping these values in an `.env` file, the project ensures better security practices and easier management of different deployment environments (development, production, etc.).

### 2.3.9 Bcrypt.js

**Bcrypt.js** is a library for hashing passwords, providing security for user authentication. In the Car Expense Tracker project, **Bcrypt.js** is used to securely hash user passwords before storing them in the database. This prevents plain-text passwords from being exposed in case of a data breach. Bcrypt applies a salt to the password and hashes it multiple times, ensuring that even identical passwords have unique hashes. This adds an extra layer of security and helps protect user data from unauthorized access.

### 2.3.10 Angular

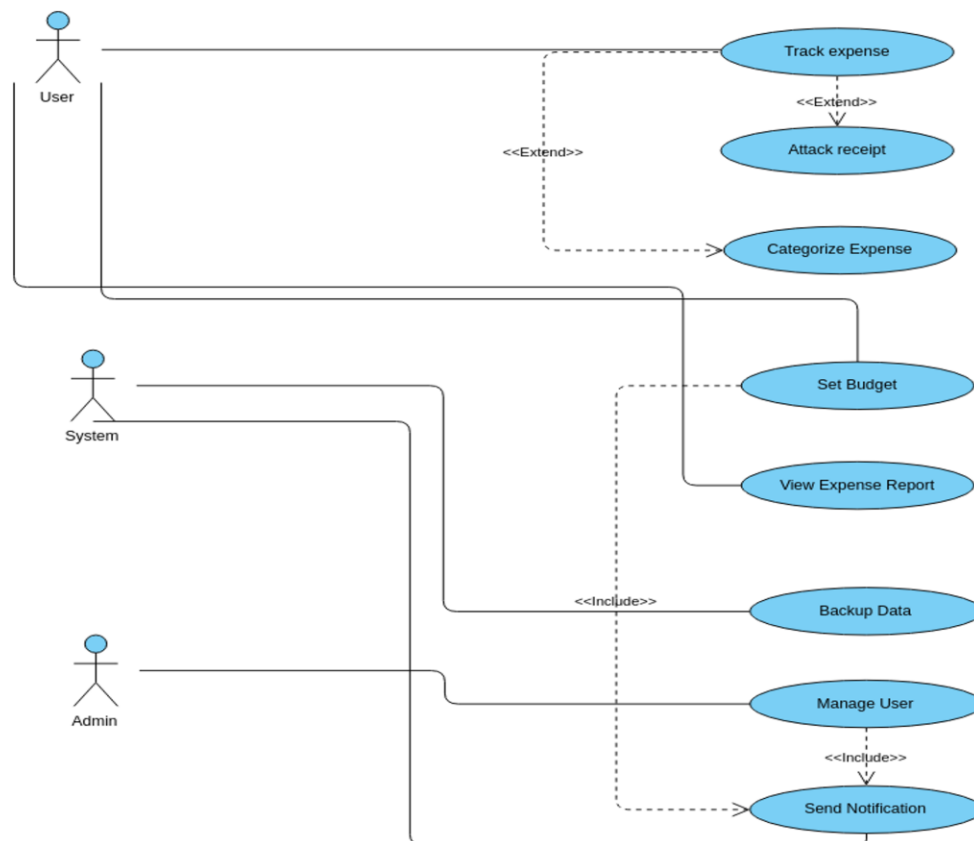
**Angular** is a robust, open-source front-end framework developed by Google, used to build dynamic, single-page web applications. In the Car Expense Tracker project, Angular is employed to develop the user interface, ensuring a responsive and interactive experience. With its component-based architecture, Angular allows the app to be modular and maintainable by breaking the UI into reusable components. It also features two-way data binding, which ensures that changes on the frontend, such as entering donation data, are automatically reflected in the model, creating a seamless experience. Angular's built-in routing allows for smooth navigation between pages, such as donation history, registration, and login. The framework's dependency injection (DI) system simplifies managing services, while RxJS and Observables handle asynchronous operations like making API calls to the backend, ensuring smooth communication with the Node.js and MongoDB backend. Angular CLI streamlines the development process by providing commands for generating components, services, and modules, making it easier to build and deploy the application. Through these features, Angular helps create a responsive, scalable, and maintainable front-end application that integrates smoothly with the backend to track donations, manage expenses, and provide an optimal user experience.

## CHAPTER 3

### SYSTEM DESIGN

#### 3.1 USE CASE DESIGN

The **Use Case Design** defines the interactions between the users (actors) and the system, illustrating how different users will engage with the Car Expense Tracker to accomplish specific tasks. These use cases provide a high-level understanding of the system's functionality and how various features support user actions.

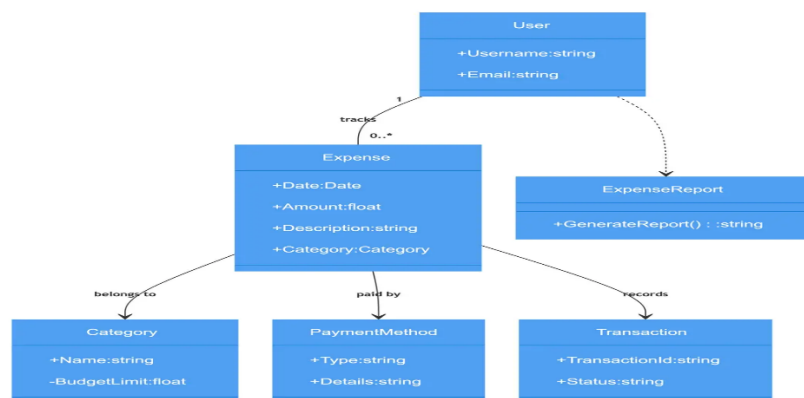


**Fig 3.1 Use Case Diagram**



## 3.2 CLASS DIAGRAM

In the class diagram for the Car Expense Tracker project, several essential classes represent the core entities and operations of the system. The user class contains attributes such as userID, name, email, password, and role to differentiate between vehicle owners and admins. It includes methods for user registration, login, updating profiles, and password management. The car class captures vehicle-specific data with attributes like carID, make, model, year, and registrationNumber, along with methods to add or update car details linked to the user. The expense class tracks all car-related expenses with attributes like expenseID, amount, category such as fuel, maintenance, or insurance, date, and description. It offers methods to add, edit, delete, and retrieve expenses. The reminder class handles service and payment reminders, including attributes such as reminderID, userID, reminderType, dueDate, and status, along with methods to create, update, disable reminders, and send notifications. The session class manages secure login sessions, with attributes such as sessionID, userID, loginTime, and logoutTime. It includes methods to create, validate, and terminate sessions. Relationships are established so that one user can own multiple cars and log multiple expenses. Each expense is associated with a car. Reminders and reports are tied to users, and the admin has authority over all user-related data and system configurations.



**Fig 3.2 Class Diagram**

### 3.3 SEQUENCE DIAGRAM

The Sequence Diagram of the Car Expense Tracker project illustrates the step-by-step interaction between system components and users during key operations, such as user login, expense logging, and report generation. One typical scenario is the Vehicle Owner logging in and recording a new car expense. In this sequence, the Vehicle Owner begins by entering login credentials through the frontend interface (developed using HTML, CSS, and Bootstrap). The frontend then sends a login request to the backend server (built using Node.js or Django), which communicates with the database (MySQL) to validate the provided credentials. Upon successful authentication, the server generates a secure session and sends a response back to the frontend, allowing the user to access their dashboard. From the dashboard, the Vehicle Owner can initiate the process of logging an expense by entering relevant details such as category, amount, date, and description. These inputs are sent to the backend, which processes and stores the information in the database. Once the data is successfully saved, a confirmation message is sent to the frontend, and the new expense entry is displayed in the user's expense list.

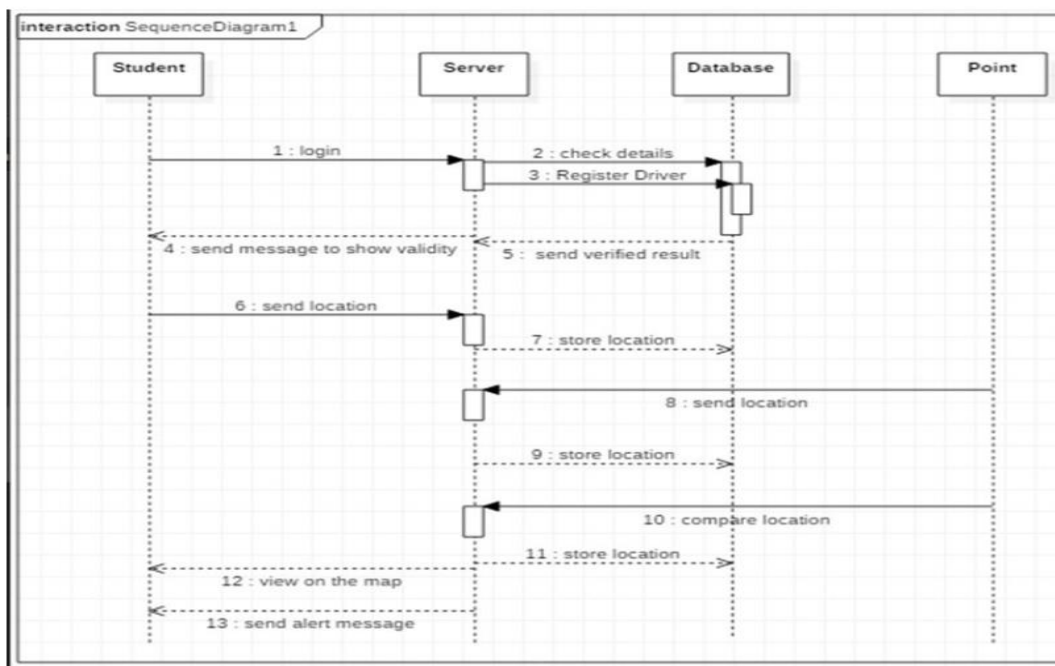


Fig 3.3 Sequence Diagram

### 3.4. ACTIVITY DIAGRAM

The Activity Diagram for the Car Expense Tracker illustrates the dynamic flow of control within the system during core operations such as user login, expense logging, reminder setting, and report generation. The process begins with the initial state where the user opens the application. The first decision point prompts the user to either register or log in. If the user is new, the system collects registration details, validates the inputs, and stores the user information in the database. If the user is already registered, they enter their credentials, which are authenticated through the backend server. Upon successful login, the system identifies the user role and directs them to the appropriate dashboard. A vehicle owner is given access to features like logging car expenses, setting service or insurance reminders, and viewing financial reports. The user can choose to log a new expense by entering details such as amount, category, and date, which are then validated and saved in the database. If the user opts to generate reports, the system fetches the relevant data and displays analytical summaries and visual charts. At any point, the user can choose to log out, ending the session and returning the system to its initial state.

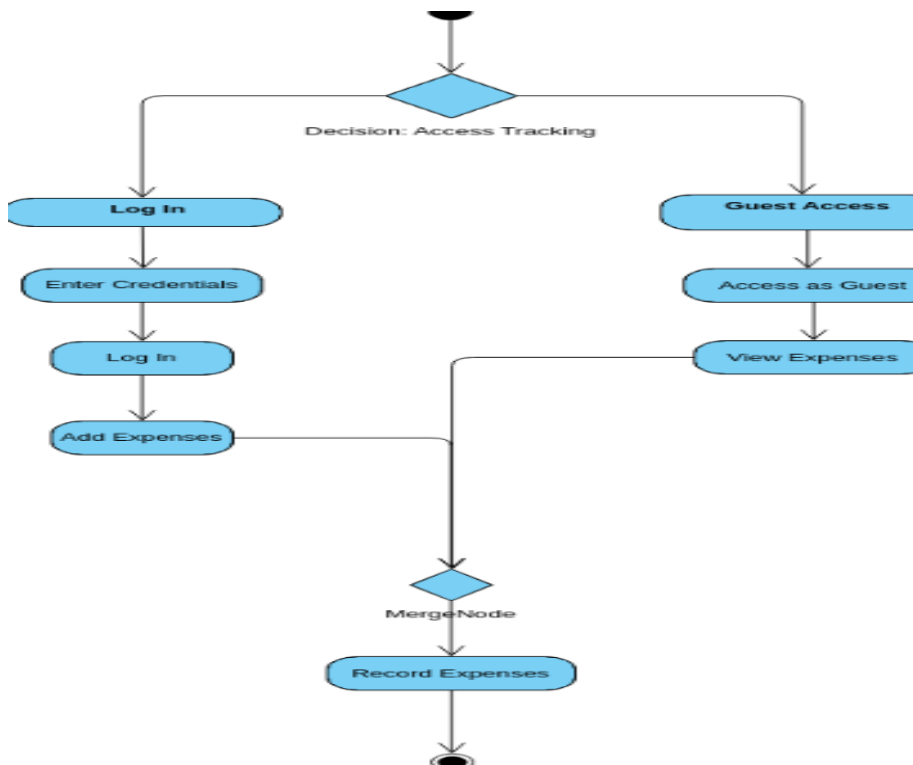


Fig3.3 Sequence Diagram

## 3.5 DATABASE DESIGN

### 3.5.1 OVERVIEW

The database design of the Car Expense Tracker plays a crucial role in efficiently managing, storing, and organizing all user and vehicle-related expense data. This project uses MySQL, a relational database management system, which ensures data consistency, integrity, and structured access through normalized tables. The design focuses on providing accurate tracking, easy scalability, and fast query performance. The core tables in the database include Users, Cars, Expenses, Reminders, and Reports. The **Users** table stores user information such as userID, name, email, password, and role. Each user can be linked to one or more entries in the **Cars** table, which holds data like carID, make, model, year, and registration number. The **Expenses** table records detailed entries with fields such as expenseID, userID, carID, category, amount, date, and description.

### 3.5.2 SIGN UP SCHEMA

The **Sign Up Schema** defines the structure of the user data that is collected and stored when a new user registers on the Car Expense Tracker platform. This schema is implemented in the backend using **Mongoose**, which allows defining the data model in a structured format for MongoDB. The schema ensures that all required user information is properly validated, securely stored, and easily accessible for authentication and role-based operations.

The key fields in the Sign Up Schema include:

**Name:** A string that stores the full name of the user. This is a required field used for identification and personalization.

**Email:** A unique string that serves as the primary identifier for the user account. It is validated for correct email formatting and used during login.

**Password:** A string that stores the user's password in a **hashed** format using **bcrypt.js** to ensure security and prevent plain-text storage.

**Role:** A string that specifies the user type, typically set as either `donor` or `admin`. This helps in implementing access control within the application.

**CreatedAt:** A date field that records the time of registration. This is useful for tracking user activity and registration trends.

### 3.5.3 LOGIN SCHEMA

The **Login Schema** in the Car Expense Tracker project is designed to authenticate registered users and grant them secure access to the system. Although login typically requires fewer fields than the sign-up process, it plays a critical role in validating user credentials and managing session flow. The login operation primarily depends on the **email** and **password** fields provided by the user, which are matched against existing records in the database using Mongoose and authentication middleware.

The key fields handled in the Login Schema include:

**Email:** A required string that must match an existing user's email in the database. It acts as the primary identifier during login and is validated for proper email formatting.

**Password:** A required string entered by the user. During the login process, this password is **compared with the hashed password** stored in the database using **bcrypt.js** to verify authenticity.

## 3.6 MODULES DESCRIPTION

### 3.6.1 AUTHENTICATION MODULE

The **Authentication Module** is one of the core components of the Car Expense Tracker project. It is responsible for verifying user identities and managing secure access to the platform. This module ensures that only authorized users, such as donors or administrators, can access and perform specific actions within the system. Built using **Node.js**, **Express.js**, **Mongoose**, and **bcrypt.js**, the module provides secure mechanisms for **sign-up**, **login**, and **session management**. During sign-up, the module validates the provided user data (such as name, email, and password), ensures the uniqueness of the email, and securely stores the password by hashing it using bcrypt before saving it in MongoDB. Upon login, the module retrieves the user's data based on the entered email, compares the hashed password, and upon a successful match, generates a **JWT (JSON Web Token)** or session token. This token is then sent back to the frontend (Angular), where it is stored and used for authenticating protected API requests.

### 3.6.2 Expense Logging Module

This core module allows users to add, view, edit, and delete car-related expenses such as fuel, maintenance, insurance, and repairs. Expenses are categorized and linked to individual users and vehicles to provide detailed records for tracking and analysis.

### 3.6.3 Reminder & Notification Module

Users can set reminders for important activities like insurance renewals, vehicle servicing, and emission checks. The system sends timely notifications via email or in-app alerts to help users avoid missing critical deadlines.

#### **3.6.4 Analytics & Report Module**

This module generates detailed reports and visual representations (charts and graphs) of the user's expenses. Users can filter data by category, date range, or vehicle, and export reports to support budgeting and decision-making.

#### **3.6.5 Vehicle Management Module**

This module allows users to register one or more vehicles with relevant details such as make, model, year, and registration number. Each vehicle is linked to its specific set of expenses, making it easier to manage costs on a per-vehicle basis.

#### **3.6.6 Admin Management Module**

Accessible only by users with admin roles, this module allows monitoring of user activity, managing system settings such as predefined categories and reminder types, and generating system-wide analytics reports for analysis and optimization.

#### **3.6.7 Database Management Module**

This backend module ensures that all user data, vehicle records, expenses, and reminders are stored securely in the database. It supports data integrity, efficient retrieval, and backup operations, making the system reliable and scalable.

## **CHAPTER 4**

### **RESULTS**

The Car Expense Tracker project was successfully developed, implemented, and tested, delivering reliable and efficient results in managing and monitoring vehicle-related expenses. The system enabled vehicle owners to register, log in, and seamlessly log various car expenses such as fuel, servicing, insurance, and repairs. Through the secure authentication module, user access was managed effectively, with role-based permissions ensuring that administrative features were only accessible to authorized users.

The application's frontend, built using HTML, CSS, and Bootstrap, provided an intuitive and mobile-responsive user experience, while the backend, powered by Node.js or Django, ensured smooth interaction with the MySQL database. Users were able to easily input and retrieve expense records, generate categorized reports, and set reminders for upcoming maintenance or insurance renewals. Admin users benefited from enhanced features such as system settings management, user activity monitoring, and access to aggregate data analytics.

The system demonstrated robust error handling, form validation, and secure password encryption using standard hashing techniques, ensuring data integrity and user security. The reminder and notification features helped users avoid missed deadlines, contributing to better vehicle maintenance and financial planning.

Overall, the Car Expense Tracker achieved its objectives by streamlining the tracking and reporting of car expenses, enhancing user awareness of spending patterns, and reducing the reliance on manual methods such as spreadsheets. The project also laid the groundwork for future enhancements including GPS-based fuel tracking, integration with real-time traffic data, multi-vehicle support, and third-party service provider APIs for insurance or servicing. The system proved to be a practical and scalable solution for individual car owners and small fleet managers alike.



## **CHAPTER 5**

### **CONCLUSION AND FUTURE WORK**

#### **Conclusion**

The Car Expense Tracker project effectively addresses the everyday challenges faced by vehicle owners in tracking and managing their car-related expenses. By integrating a clean and user-friendly interface with secure authentication and structured data handling, the system allows users to seamlessly log expenses, set reminders, and generate analytical reports. Whether tracking fuel costs, maintenance bills, or insurance renewals, the platform enables users to maintain accurate and organized records, reducing the dependence on manual methods such as paper logs or spreadsheets.

Developed using modern web technologies like HTML, CSS, Bootstrap, Node.js or Django for the backend, and MySQL for data storage, the application ensures robust performance, fast data retrieval, and secure information handling. The role-based access control further enhances data privacy and platform integrity by restricting administrative features to authorized users. The system meets its core objectives by improving financial awareness, supporting timely vehicle upkeep, and helping users plan their budgets more effectively. With its modular structure and scalable design, the Car Expense Tracker stands as a reliable and practical solution for both individual users and small-scale fleet managers seeking a smarter way to manage car expenses.

#### **Future Scope**

The Car Expense Tracker project holds significant potential for future enhancements aimed at enriching the user experience and expanding platform capabilities. A key improvement would be the integration of a mobile application version, allowing users to log and manage expenses directly from their smartphones, providing greater convenience and accessibility.

Adding support for SMS or email notifications could help alert users about upcoming service reminders, insurance renewals, and spending limits, ensuring timely action and reducing the risk of oversight.

Incorporating GPS-based mileage tracking and fuel consumption analysis would offer a deeper level of insight into vehicle usage patterns, while integration with external APIs—such as service center databases or insurance providers—could automate data entry and streamline record management.

A dynamic dashboard featuring visual representations like pie charts and trend lines would enable users to better understand their spending habits and optimize future vehicle-related decisions.

Multi-vehicle support would be beneficial for users managing more than one car, allowing them to switch between vehicle profiles and track expenses individually. Additionally, implementing AI-driven features like automated expense categorization or predictive alerts for recurring expenses could further enhance the system's intelligence and efficiency.

With these advancements, the Car Expense Tracker could evolve into a comprehensive, data-driven tool for modern vehicle management, offering long-term value for both individual users and commercial operators.

# APPENDIX 1

## CODING

### Index.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Car Expense Tracker</title>
<link rel="stylesheet" href="styleless.css">

</head>
<body>

<script src="script.js"></script>
<section class="hero">
<video autoplay muted loop class="bg-video">
<source src="2229697-uhd_3840_2160_30fps.mp4" type="video/mp4">
Your browser does not support the video tag.
</video>

<div class="overlay"></div>
<div class="hero-content">
<h1>CAR EXPENSES</h1>
<p>Manage Your Vehicle Expenses And Get Back On The Track!</p>
<div class="buttons">
<a href="about.html" class="btn find-out">Find Out More</a>
<a href="signup.html" class="btn sign-up">Sign Up Now</a>
</div>
</div>
</section>

</body>
<script>
window.onload = function () {
const video = document.getElementById('bgVideo');
video.playbackRate = 2.0; // Change this to whatever speed you want
};
</script>

</html>
```

## Server.js:

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const dotenv = require('dotenv');

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

mongoose.connect(process.env.MONGODB_URI || 'mongodb://localhost:27017/car-
management', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log('Connected to MongoDB'))
.catch(err => console.error('MongoDB connection error:', err));

const expenseSchema = new mongoose.Schema({
  date: { type: Date, required: true },
  category: { type: String, required: true },
  amount: { type: Number, required: true },
  description: String,
  receiptName: String
});

const Expense = mongoose.model('Expense', expenseSchema);

app.post('/api/expenses', async (req, res) => {
  try {
    const { date, category, amount, description, receiptName } = req.body;
    const expense = new Expense({
      date,
      category,
      amount,
      description,
      receiptName
    });
    await expense.save();
    res.status(201).json(expense);
  } catch (error) {
    res.status(500).json({ message: 'Error creating expense', error: error.message });
  }
});

app.get('/api/expenses', async (req, res) => {
  try {
    const expenses = await Expense.find().sort({ date: -1 });
```

```

res.json(expenses);
} catch (error) {
res.status(500).json({ message: 'Error fetching expenses', error: error.message });
}
});

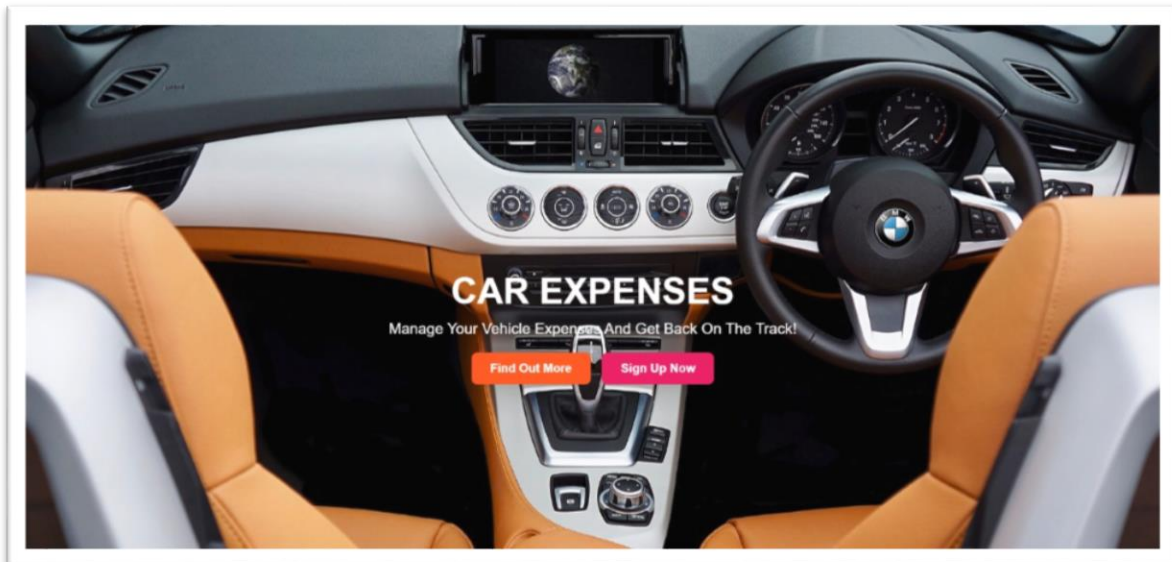
app.delete('/api/expenses/:id', async (req, res) => {
try {
const expense = await Expense.findByIdAndDelete(req.params.id);
if (!expense) {
return res.status(404).json({ message: 'Expense not found' });
}
res.json({ message: 'Expense deleted successfully' });
} catch (error) {
res.status(500).json({ message: 'Error deleting expense', error: error.message });
}
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
console.log(`Server running on port ${PORT}`);
});

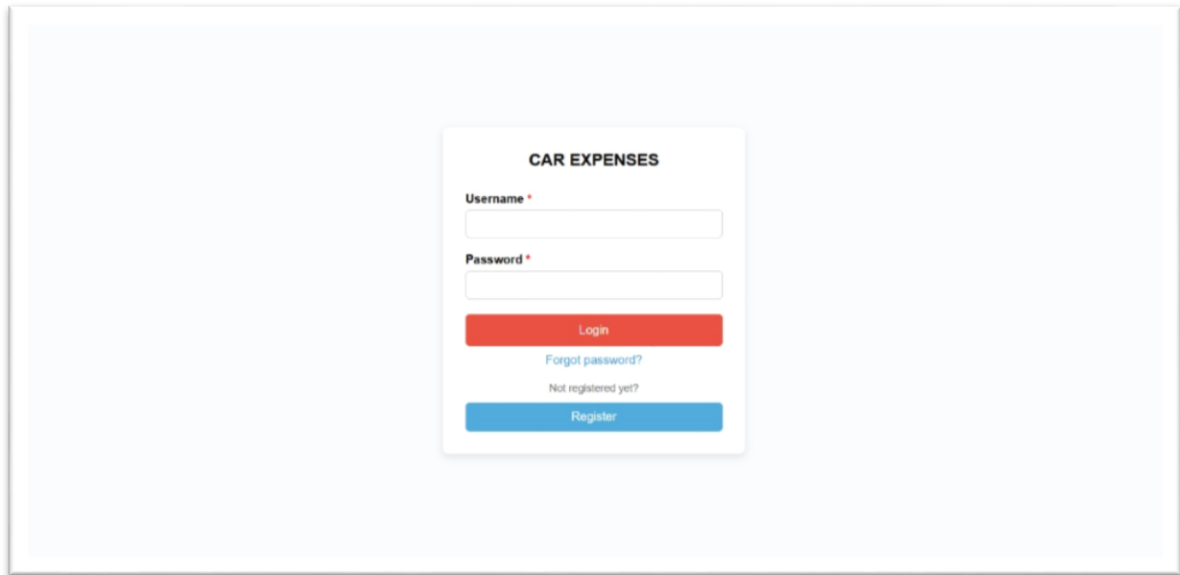
```

## APPENDIX 2

### SNAPSHOTS



**Figure A2.1 HOME PAGE**



The login page features a central white card with a light gray border. At the top of the card is the title "CAR EXPENSES" in bold black text. Below the title are two input fields: "Username \*" and "Password \*", both with red asterisks indicating required fields. Under the password field is a red "Login" button. Below the button are two links: "Forgot password?" and "Not registered yet?". At the bottom of the card is a blue "Register" button. The entire card is centered on a light blue background.

**CAR EXPENSES**

**Username \***

**Password \***

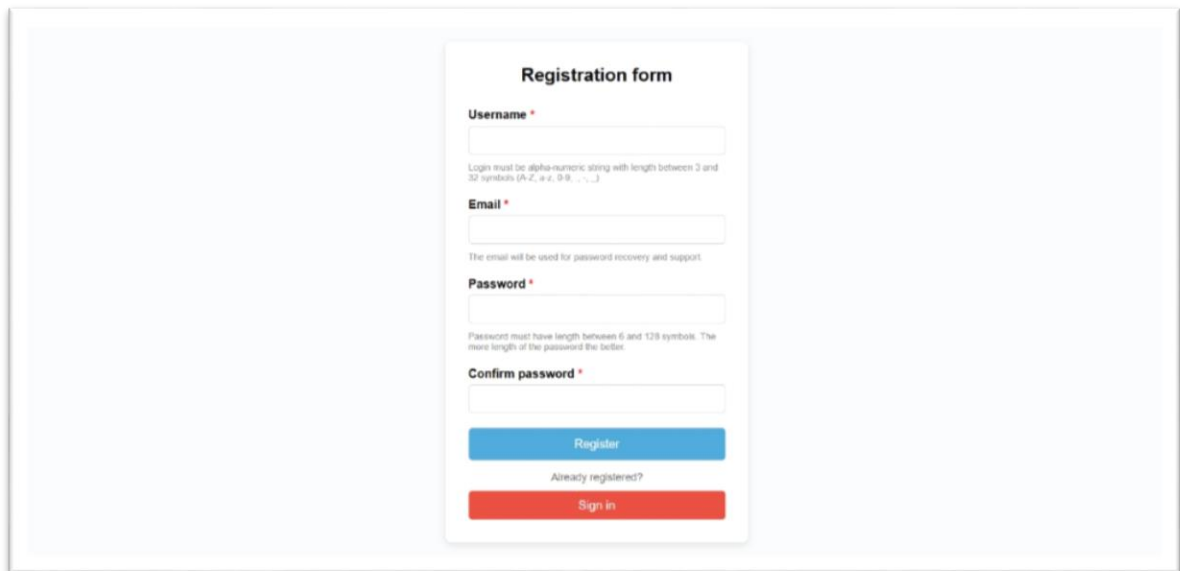
Login

[Forgot password?](#)

[Not registered yet?](#)

Register

**Figure A2.2 LOGIN PAGE**



The registration form is a white card with a light gray border, centered on a light blue background. It has the title "Registration form" in bold black text. The form contains four input fields: "Username \*" with a red asterisk, "Email \*" with a red asterisk, "Password \*" with a red asterisk, and "Confirm password \*" with a red asterisk. Below the "Username \*" field is a small gray text block: "Login must be alpha-numeric string with length between 3 and 30 symbols (A-Z, a-z, 0-9, ., -, \_)". Below the "Email \*" field is another small gray text block: "The email will be used for password recovery and support." Below the "Password \*" field is a third small gray text block: "Password must have length between 6 and 128 symbols. The more length of the password the better." Below the "Confirm password \*" field is a blue "Register" button. Below the button is a link "Already registered?". At the bottom of the card is a red "Sign in" button.

**Registration form**

**Username \***

Login must be alpha-numeric string with length between 3 and 30 symbols (A-Z, a-z, 0-9, ., -, \_)

**Email \***

The email will be used for password recovery and support.

**Password \***

Password must have length between 6 and 128 symbols. The more length of the password the better.

**Confirm password \***

Register

[Already registered?](#)

Sign in

**Figure A2.3 REGISTRATION PAGE**

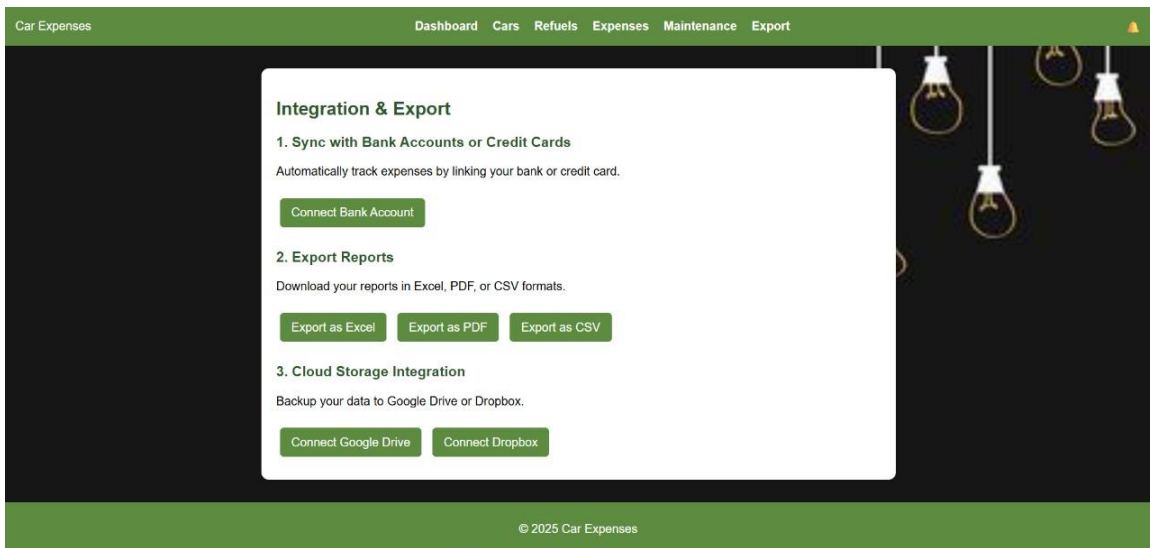


Figure A2. 4 EXPORT PAGE

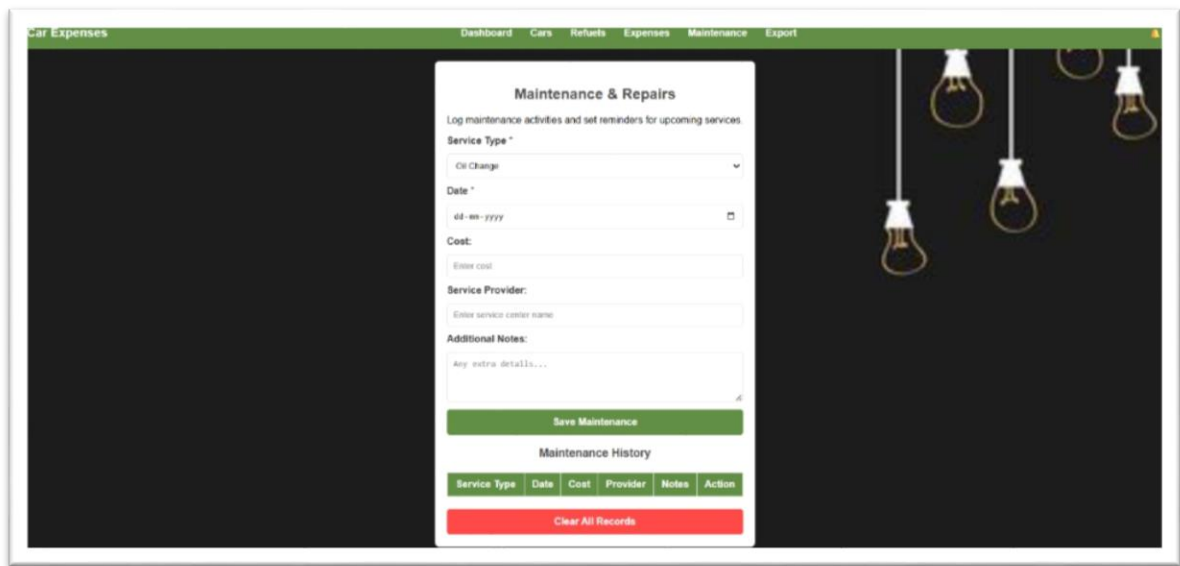


Figure A2.5 MAINTENANCE PAGE

Car Expenses

DashboardCarsRefuelsExpensesMaintenanceExport

Log Fuel Purchase

Date \*  
dd-mm-yyyy

Fuel Amount (liters/gallons) \*  
Enter fuel amount

Price per Unit \*  
Enter price per liter/gallon

Mileage (km/miles) \*  
Enter mileage since last refuel

Add Refuel

Fuel Log

Date	Fuel (L/G)	Price per Unit	Mileage	Efficiency (Km/L or MPG)	Actions
Clear All Data					

Fuel Efficiency Graph

Fuel Efficiency (Km/L or MPG)

1.0  
0.8  
0.6  
0.4  
0.2  
0

© 2016 Car Expenses v 1.0.0.0

FigureA2.6 REFUEL PAGE

Expense Tracker

Date:  
dd-mm-yyyy

Category:  
Fuel

Amount:  
Enter cost

Description:

Upload Receipt:  
Choose File No file chosen

Add Expense

Expense Records

Date	Category	Amount	Description	Receipt	Action
------	----------	--------	-------------	---------	--------

Figure A2.7 EXPENSE TRACKER PAGE



Car Expenses

Dashboard

Cars

Refuels

Expenses

Maintenance

Travels

Add Car

All fields marked with \* are mandatory.

Main Information

Vehicle Name \*

Enter vehicle name

Initial Mileage \*

Enter initial mileage

Mileage Unit \*

Miles

Tank Volume \*

Enter tank volume in liters

Make

Enter car make (e.g., Toyota)

Model

Enter car model (e.g., Corolla)

Color

Enter car color

Manufactured Year

Enter year of manufacture (e.g., 2022)

Save Car

Registered Cars

Figure A2.8 ADD CAR PAGE

## REFERENCES

1. MongoDB Official Documentation – <https://www.mongodb.com/docs>
2. Node.js Documentation – <https://nodejs.org/en/docs>
3. Express.js Guide – <https://expressjs.com/en/starter/installing.html>
4. Angular Framework Documentation – <https://angular.io/docs>
5. Mongoose ODM Documentation – <https://mongoosejs.com/docs>
6. Bcrypt.js GitHub Repository – <https://github.com/kelektiv/node.bcrypt.js/>
7. Visual Studio Code User Guide – <https://code.visualstudio.com/docs>
8. CORS Middleware for Node.js – <https://www.npmjs.com/package/cors>
9. Body-Parser Middleware – <https://www.npmjs.com/package/body-parser>
10. Dotenv Package Documentation – <https://www.npmjs.com/package/dotenv>
11. W3Schools – <https://www.w3schools.com>
12. GeeksforGeeks – <https://www.geeksforgeeks.org>
13. TutorialsPoint – <https://www.tutorialspoint.com>
14. Stack Overflow – <https://stackoverflow.com>

