A Scalable MERN-Based Intelligent Vehicle Data Management and Analytics System

1) Mrs.Sujata suresh Vijapure- Student

2) Mrs.Nanda Kulakarni- Project Guide(HOD)

3) Mrs.Trupti Bhase- Project Guide

4) Mrs.Sujata Salunke- Project Guide

How to Cite this Article:
Vijapure, S. S., Bhase, T. & Salunke, S. (2026). A Scalable MERN-Based Intelligent Vehicle Data Management and Analytics System. International Journal of Creative and Open Research in Engineering and Management, 02(05). https://doi.org/10.55041/ijcope.v2i5.397

https://doi.org/10.55041/ijcope.v2i5.397

Abstarct :

In today's rapidly evolving digital world, effective management of vehicle-related data has become essential for both individuals and organizations handling fleets. Conventional approaches such as manual logbooks and spreadsheet-based tracking systems are often inefficient, error-prone, and do not provide real-time access or meaningful insights. To overcome these challenges, this project introduces **Drive Ledger**, a web-based vehicle management solution developed using the MERN stack, including MongoDB, Express.js, React.js, and Node.js.

The system acts as a centralized platform for storing and managing key vehicle information such as trip records, fuel usage, maintenance history, and driver details. It integrates secure user authentication, role-based access control, and seamless communication through RESTful APIs. The frontend is designed to deliver a user-friendly and responsive experience, while the backend ensures efficient data handling and storage.

The proposed solution enhances operational productivity, increases data reliability, and assists in better decision-making through interactive dashboards and reports. System evaluation indicates strong performance, quick data access, and scalability. Additionally, the platform is designed to support future improvements such as predictive maintenance, IoT-based tracking, and mobile application integration.

Keyword :

Vehicle Management System, MERN Stack, MongoDB, Express.js, React.js, Node.js, Web Application, Fleet Management, Data Management, RESTful APIs, Role-Based Access Control, Data Analytics, Trip Tracking, Fuel Monitoring, Maintenance Management.

1. Introduction

The **Drive Ledger** project is a full-stack web application designed using the MERN stack—MongoDB, Express.js, React.js, and Node.js—with the objective of automating and digitalizing vehicle data management. The system enables organized collection, storage, and retrieval of important vehicle-related information such as trip records, fuel consumption, and maintenance details through an intuitive user interface and RESTful APIs.

The backend is developed using Node.js and Express.js, which manage API handling, business logic, and interaction with the database. MongoDB is used as the database, with Mongoose acting as the Object Data Modeling (ODM) tool to provide a flexible and scalable schema structure. On the frontend, React.js is utilized to create a responsive and dynamic user interface, enhanced with styling frameworks like Material UI or Tailwind CSS.

The overall system architecture supports essential operations such as Create, Read, Update, and Delete (CRUD), along with asynchronous data communication and modular component-based design. It ensures secure data handling while maintaining high performance and scalability. The application is suitable for both individual users and organizations that require centralized access to real-time vehicle data and operational control.

By replacing traditional manual methods, the system improves data accuracy, supports informed decision-making, and enhances overall efficiency through the use of modern web technologies.

2. Literature Review

Various research studies have examined the development of web-based applications and data management systems using modern technologies such as the MERN stack. Chamalla et al. (2024) designed a study support application utilizing MERN technologies, demonstrating the effectiveness of full-stack JavaScript frameworks in building scalable and efficient web solutions. Their research emphasizes the significance of modular system design and real-time data processing in modern applications.

Thokala (2021) analyzed issues related to data integrity and redundancy in distributed database systems. The study highlights the importance of proper data validation techniques and well-structured database design to maintain consistency and reliability. These aspects are especially crucial in systems like vehicle management, where accurate and consistent data handling is essential.

Raju et al. (2021) explored the development of applications using the MERN stack and discussed its benefits, including rapid development, code reusability, and efficient communication between client and server components. Their findings support the adoption of MERN as a powerful framework for building modern web applications.

Despite these advancements, most existing research focuses on general-purpose web applications rather than domain-specific solutions such as vehicle management systems. Furthermore, many current systems lack advanced features like real-time data analytics, role-based access control, and seamless integration of multiple operational modules.

Research Gap

- Limited availability of integrated vehicle management systems built using the MERN stack
- Insufficient implementation of real-time analytics in fleet management applications
- Lack of focus on secure user role management and access control mechanisms

The proposed **Drive Ledger** system aims to overcome these limitations by offering a unified, scalable, and secure platform specifically designed for vehicle data management.

In recent years, advancements in web technologies and cloud computing have enabled the creation of intelligent fleet management systems. Many of these systems incorporate IoT devices and GPS tracking to provide real-time monitoring and tracking capabilities. However, such solutions are often costly and complex to implement.

The Drive Ledger system offers a more affordable and scalable alternative by leveraging the MERN stack. It ensures efficient data processing, secure user access, and real-time updates without requiring expensive infrastructure, making it a practical solution for modern vehicle management needs.

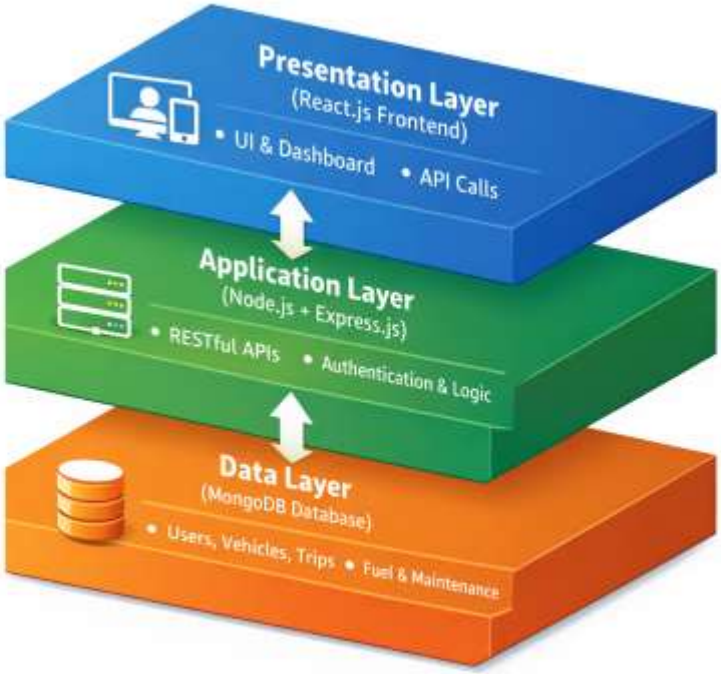

Figure 1 - 3-Tier Architecture Diagram of Drive Ledger System

3.Methodolgy

3.1 Introduction

The methodology of the **Drive Ledger system** defines the structured approach followed to design, develop, and implement the web-based vehicle management application. It ensures that the system is efficient, secure, scalable, and capable of handling real-time vehicle data. The development process is based on modern web technologies and follows a modular and layered architecture.

3.2 Development Approach

The project follows an **Agile-based incremental development model**, where the system is divided into smaller modules and developed step by step.

Phases of Development:

- **Requirement Analysis:**Identifying system needs such as trip tracking, fuel monitoring, maintenance logging, and user roles.
- **System Design:**Creating architecture diagrams, ER diagrams, and defining system modules.
- **Implementation:**Developing frontend, backend, and integrating APIs with the database.
- **Testing:**Performing unit testing and integration testing to ensure proper functionality.
- **Deployment (Optional):**Hosting the application on platforms like cloud servers.

3.3 Database Design Methodology

The Drive Ledger system uses **MongoDB**, a NoSQL database, to store and manage vehicle-related data in a flexible and scalable manner. Unlike traditional relational databases, MongoDB stores data in document format (JSON-like), which allows dynamic schema design.

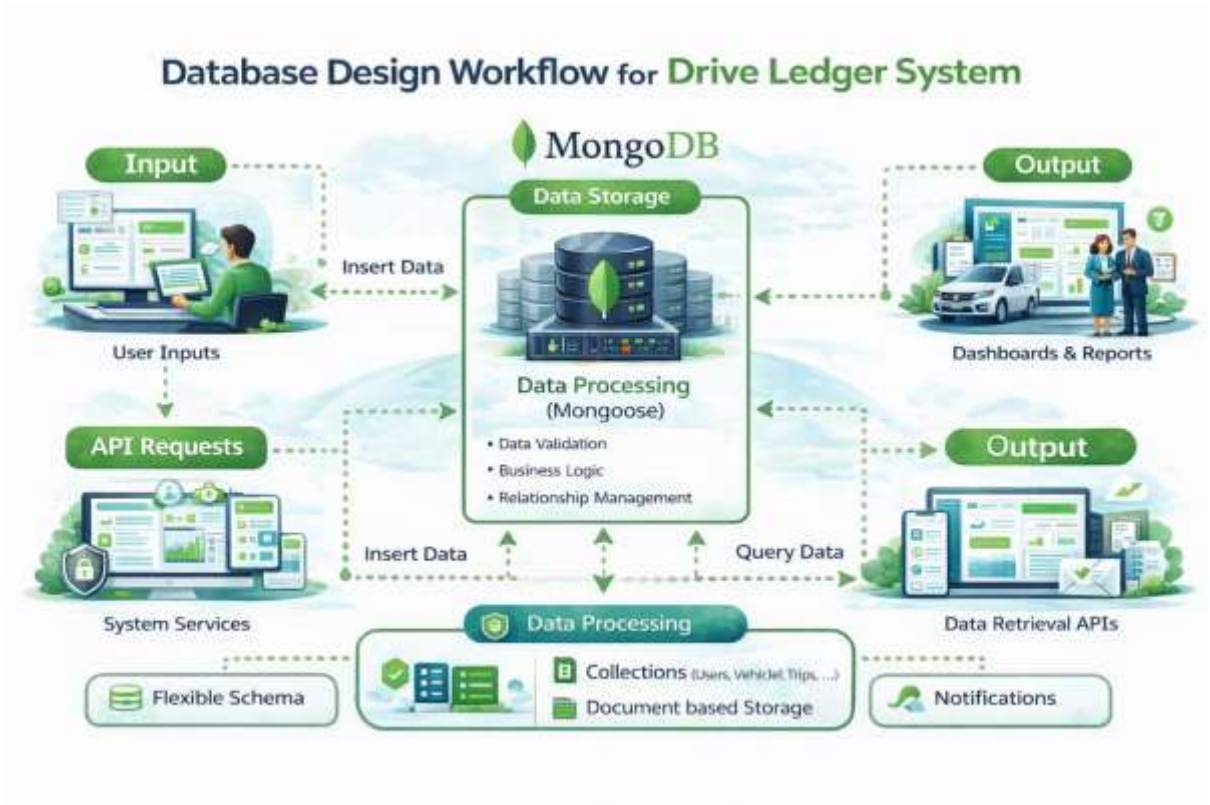

Figure2- Database Processing Workflow Diagram

Collections Used:

- **Users:** Stores user credentials and roles
- **Vehicles:** Contains vehicle details such as number, model, and type
- **Drivers:** Stores driver information
- **Trips:** Records trip details like distance, date, and vehicle used
- **Fuel Logs:** Maintains fuel consumption data
- **Maintenance Records:** Stores servicing and repair details

Design Features:

- Flexible schema allows easy modification
- Reduced data redundancy improves efficiency
- Faster query performance ensures quick data access

Relationships:

- One vehicle can have multiple trips
- One driver can handle multiple trips
- One trip can have multiple fuel logs
- One vehicle can have multiple maintenance records

This design ensures efficient data storage, consistency, and scalability.

3.4 Module-Based Implementation

The system is developed using a **modular approach**, where each module handles a specific functionality. This improves maintainability and scalability.

Modules:

- **Authentication Module:**Handles user login and registration. Uses JWT for secure authentication and role-based access control.
- **Vehicle Module:**Manages vehicle data including adding, updating, and deleting vehicle records.
- **Driver Module:**Stores driver details and assigns drivers to vehicles or trips.
- **Trip Module:**Records trip information such as distance, date, and assigned vehicle/driver.
- **Fuel Module:**Tracks fuel usage, quantity, and cost for each trip.
- **Maintenance Module:**Maintains records of servicing, repairs, and vehicle health history.

Each module interacts with backend APIs to perform CRUD operations efficiently.

3.5 API Design Methodology

The **Drive Ledger system** uses **RESTful APIs (Representational State Transfer)** to enable communication between the frontend and backend. APIs act as an interface that allows the user interface to send requests to the server and receive responses in a structured format. This approach ensures smooth interaction between different components of the system.

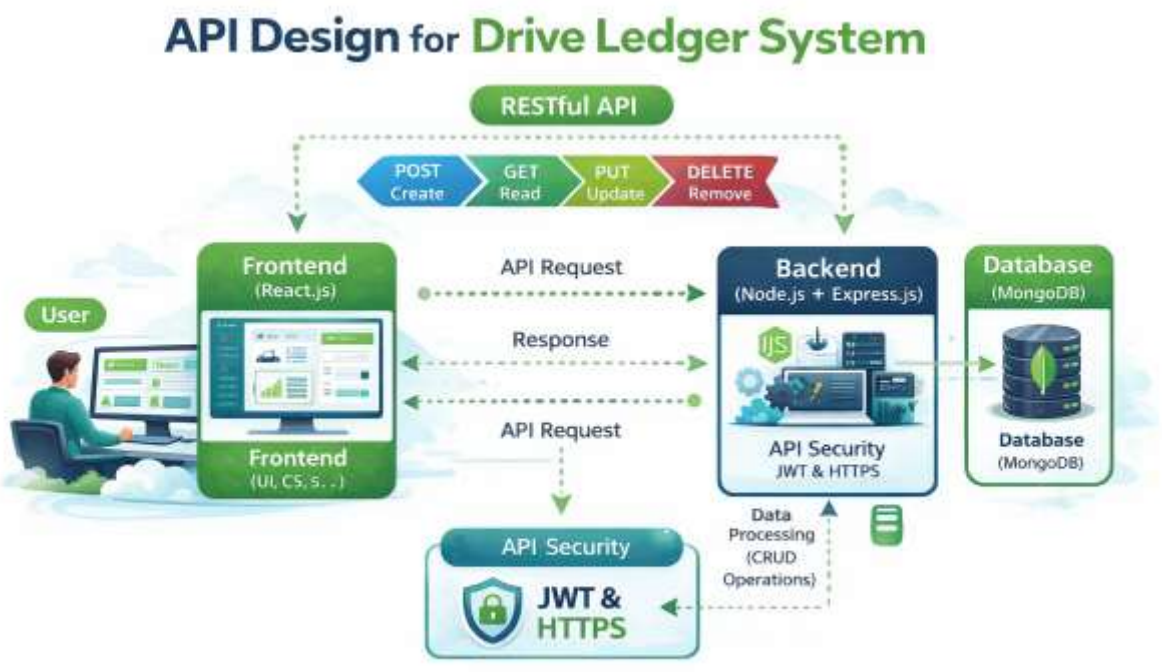

Figure 3-API Workflow of Vehicle Management System

3.5.1 Purpose of API Design

The main purpose of API design in the system is to:

- Enable efficient communication between frontend and backend
- Support data exchange in a structured format
- Perform CRUD operations (Create, Read, Update, Delete)
- Maintain separation between user interface and business logic
- Allow easy scalability and integration with future technologies

---

3.5.2 RESTful Architecture

The system follows a **RESTful architecture**, which is based on standard web communication principles.

Key Characteristics:

- **Stateless Communication:** Each request is independent and contains all required information
- **Client-Server Model:** Frontend and backend operate separately
- **Resource-Based Design:** Each data entity is treated as a resource
- **Standard Methods:** Uses predefined HTTP methods for operations

This architecture improves flexibility and scalability.

3.5.3 API Operations (CRUD)

APIs in the system are designed to perform the following operations:

- **Create:** Add new records such as vehicles, trips, or users
- **Read:** Retrieve existing data from the database
- **Update:** Modify existing records
- **Delete:** Remove unwanted data

These operations ensure complete control over system data.

3.5.4 API Endpoint Structure

Each module in the system is associated with specific API endpoints. These endpoints represent different resources such as users, vehicles, trips, fuel logs, and maintenance records.

- Authentication endpoints handle user login and registration
- Vehicle endpoints manage vehicle-related operations
- Trip endpoints handle trip data
- Fuel endpoints manage fuel records
- Maintenance endpoints store service information

This structured design makes the system organized and easy to maintain.

3.5.5 API Workflow

The working process of APIs in the system is as follows:

1. The user performs an action on the frontend interface
2. The frontend sends a request to the backend API
3. The backend processes the request using business logic
4. The system interacts with the database to store or retrieve data
5. The backend sends a response back to the frontend
6. The frontend displays the result to the user

This workflow ensures efficient and real-time data processing.

3.5.6 Security in API Design

Security is an important aspect of API design. The system implements:

- **Authentication mechanisms** to verify user identity
- **Authorization controls** to restrict access based on roles
- **Input validation** to prevent incorrect or harmful data
- **Secure communication protocols** to protect data during transmission

These measures ensure safe and reliable communication.

---

4. Three-Tier Architecture

4.1. Presentation Layer (The Frontend)

This is the "face" of the application—what the users (Admins, Drivers) interact with.

- **Role:** Captures user input (like entering a fuel log) and displays data (like a vehicle's maintenance history).
- **Technology:** Usually built with frameworks like **React, Angular, or a Mobile App (Flutter/Swift)**.
- **Communication:** It does not talk to the database directly. It sends HTTP requests (GET, POST, etc.) to the API layer and renders the JSON response it receives.

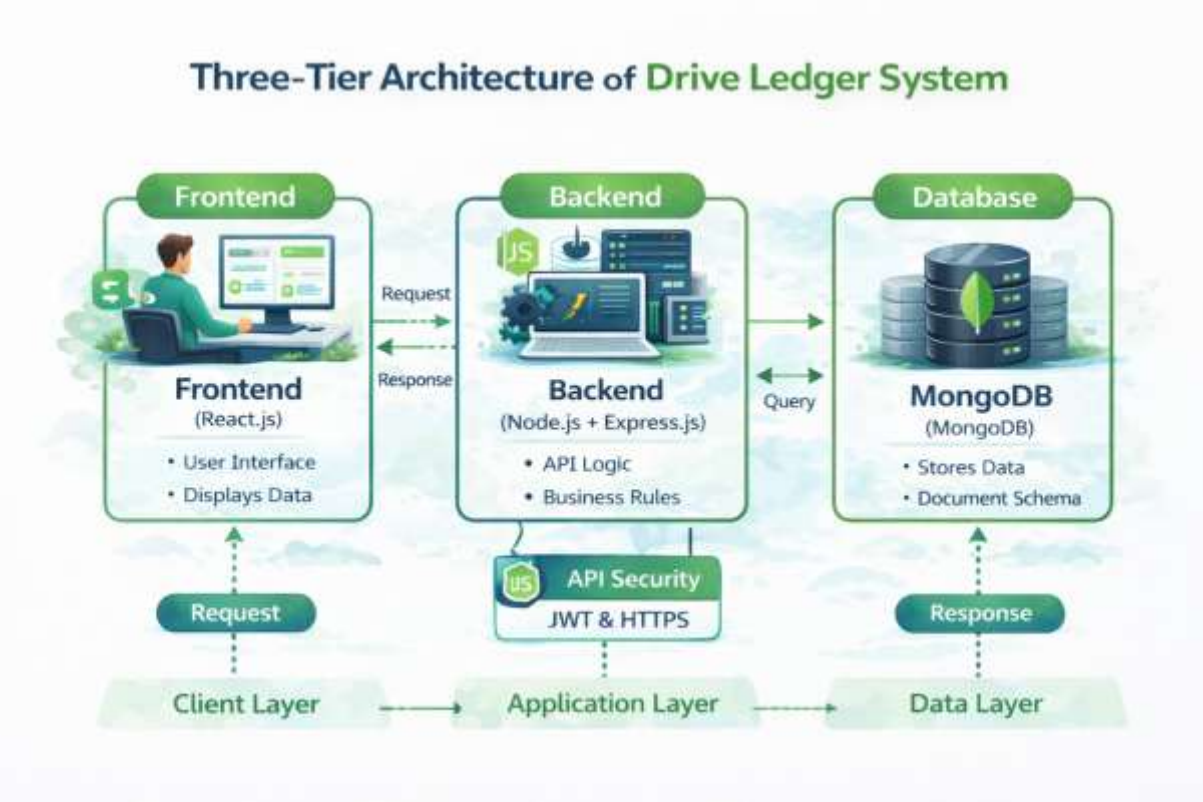

Figure 4 – Three tier Architecture

4.2. Application/Logic Layer (The Backend API)

This is the "brain" of the system where the **Module-Based Implementation (Section 3.6)** lives. It acts as the intermediary.

- **Role: * Authentication:** Validates JWT tokens to ensure the user is logged in.
  - **Validation:** Checks if the data makes sense (e.g., ensuring "Fuel Liters" isn't a negative number).
  - **Business Rules:** Calculates costs, triggers maintenance alerts, and handles the logic of assigning drivers to vehicles.
- **Structure:** This is where your **RESTful APIs** reside. It receives a request from the Presentation Layer, processes it, and decides what to do with the data.

4.3. Data Layer (The Database)

This is the "memory" of the system, using **MongoDB**.

- **Role:** Persistence. It stores the collections of Users, Vehicles, and Trips.
- **Flexibility:** Because it is a NoSQL layer, if you decide next month that you need to store "Tire Pressure" in the Vehicle collection, you can add that field without having to restructure the entire database.
- **Efficiency:** It handles the complex task of indexing and retrieving thousands of logs quickly when the Logic Layer requests them.

5. Conclusion

The **Drive Ledger system** successfully demonstrates the development of a modern, web-based vehicle management solution using the MERN stack. The project effectively replaces traditional manual methods such as logbooks and spreadsheets with a centralized and automated system that ensures accuracy, efficiency, and real-time accessibility of vehicle data.

The system integrates key functionalities including trip management, fuel tracking, maintenance records, and user management into a single platform. With the implementation of RESTful APIs, modular architecture, and a three-tier design, the application achieves smooth data flow, scalability, and maintainability. Security mechanisms such as authentication and role-based access control further enhance the reliability of the system.

The use of MongoDB enables flexible and efficient data storage, while React.js provides a dynamic and user-friendly interface. The system has been tested to ensure proper performance, fast data retrieval, and reliable operation under different conditions.

Overall, the Drive Ledger system provides a cost-effective, scalable, and efficient solution for vehicle data management. It lays a strong foundation for future enhancements such as mobile application development, IoT integration, GPS tracking, and advanced data analytics.

6.References

1. Chamalla, S., et al., “Development of a Study Buddy Application using MERN Stack,” *International Journal of Computer Applications*, 2024.
2. Thokala, V., “Data Integrity and Redundancy in Distributed Databases,” *International Journal of Advanced Research in Computer Science*, 2021.
3. Raju, P., et al., “Implementation of Web Applications using MERN Stack,” *Journal of Web Engineering and Technology*, 2021.
4. MongoDB Documentation, “MongoDB Manual,” Available: https://www.mongodb.com/docs/
5. Node.js Documentation, “Node.js Official Documentation,” Available: https://nodejs.org/
6. Express.js Documentation, “Express.js Guide,” Available: https://expressjs.com/
7. React.js Documentation, “React Official Documentation,” Available: https://react.dev/
8. Fielding, R. T., “Architectural Styles and the Design of Network-based Software Architectures,” Doctoral Dissertation, University of California, 2000.
9. Postman, “Postman API Platform Documentation,” Available: https://www.postman.com/
10. Git Documentation, “Git Official Documentation,” Available: https://git-scm.com/docs