A
Project
on
Vehicle Performance Dashboard

Synopsis

Team Members

Anjalika Goswami

Shivangi Srivastava

Mentors

Mrs. Ruchi Gupta

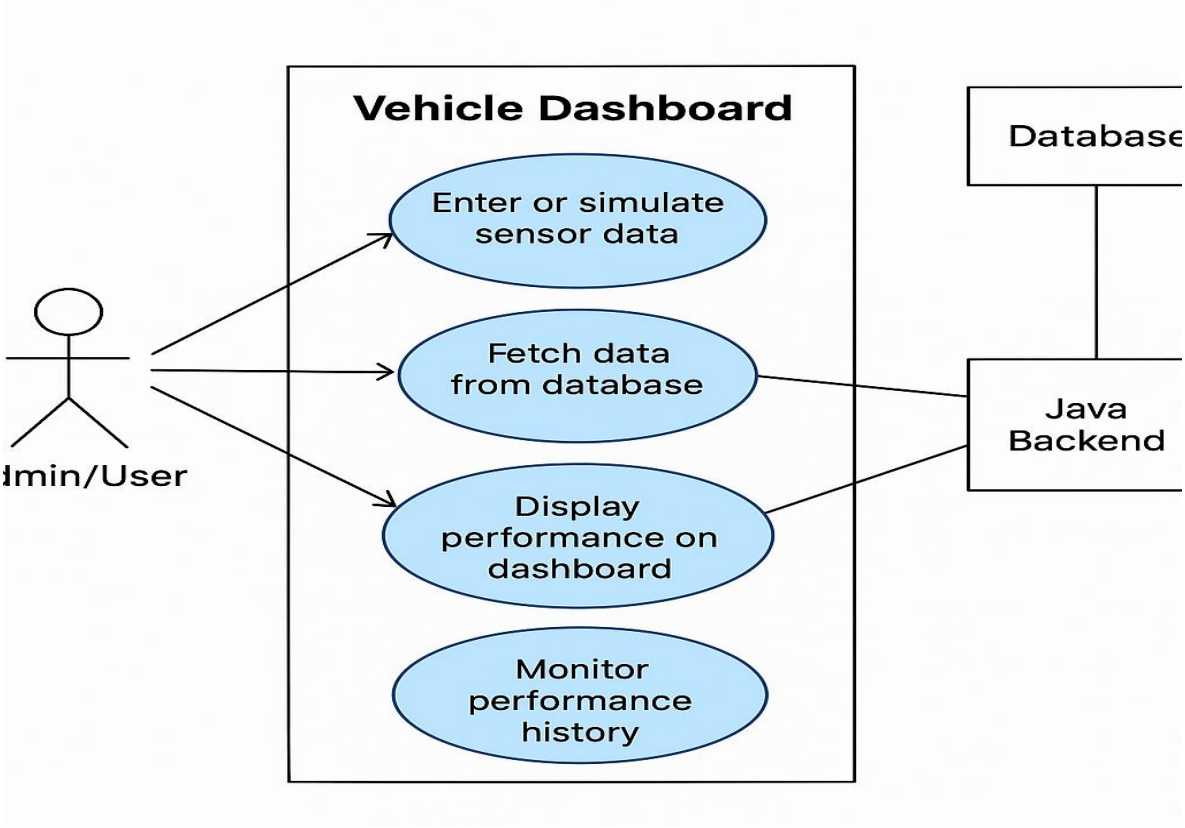
Master Trainer, CEA, GLA

INDEX

INTRODUCTION

The automotive industry has witnessed significant advancements in vehicle design, manufacturing, and performance optimization over the past few decades. With the evolution of technology, modern vehicles are equipped with a wide range of sensors that continuously monitor parameters such as speed, fuel efficiency, engine temperature, brake pressure, and overall health of various subsystems.

The project Vehicle Performance Dashboard aims to build a real-time data visualization platform that can display important vehicle performance metrics. It leverages backend technologies like Java 8, Collections API, and PostgreSQL, along with frontend technologies such as HTML, CSS, and JavaScript, to provide a seamless and interactive experience to users.



TECHNOLOGIES USED

Frontend Tech Stack

1. HTML (Hyper Text Markup Language)

- Structure and content of web pages.
- Defines elements such as headings, paragraphs, forms, and links.

2. CSS (Cascading Style Sheets)

- Styling and layout of HTML elements.

3. JavaScript (ES6+)

- Adds interactivity and dynamic behavior to web pages.

4. JSON (JavaScript Object Notation)

- Lightweight data format used for exchanging data between client and server.

Backend Tech Stack

1. Java 8 (with Collections API)

- **Core Language** for backend logic, APIs, and server-side computations.
- **Java Collections API** used for efficient data storage, retrieval, and manipulation:

2. PostgreSQL (via JDBC Driver)

- **Relational Database** for structured data storage.
- **JDBC (Java Database Connectivity)** used to connect Java applications with the PostgreSQL database.

Development Tools

1. Visual Studio Code (VS Code)

- Lightweight and powerful code editor for frontend and backend development.
- Offers integrated terminal, Git support, and IntelliSense for smart code completions.

REQUIREMENTS AND RESOURCES

Requirement gathering is the initial and critical phase of the project development life cycle. It involves collecting information about the project's objectives, features, and limitations to ensure that the final system meets user needs.

For the Vehicle Performance Dashboard, the requirement gathering process included:

- Study of existing vehicle monitoring systems.
- Analysis of common vehicle performance parameters needed during testing (e.g., speed, fuel level, engine temperature, tire pressure).
- Identification of real-time data needs and database storage capabilities.
- Sources of Requirements:
- Research papers and documentation on infield vehicle testing systems.

Non-functional Requirements

- Non-functional requirements specify the system's operational capabilities and constraints. They define **"how"** the system performs rather than **"what"** it does.

1. **Performance:**System should process and display real-time data with minimal delay (<1 second).

2. **Scalability**: System must support addition of new sensor types in future without

3. **Reliability** :System must handle database downtime gracefully and ensure no data loss.

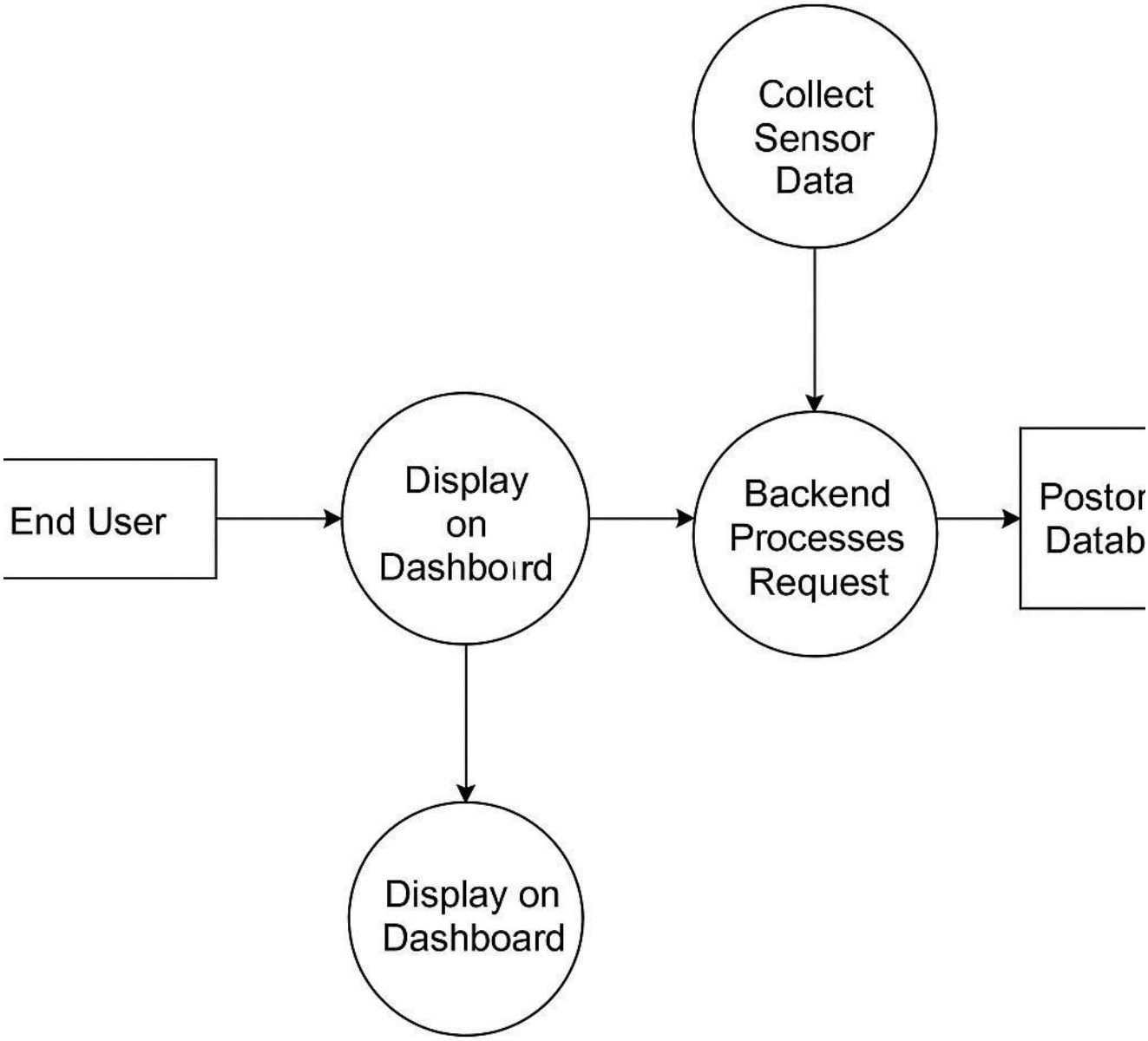
4. **Security**:Data must be securely transmitted using HTTPS; authentication is mandatory.

5. **Usability**:The dashboard interface should be simple , intuitive ,and user-friendly.

DFD

Data Flow Diagrams (DFD):

A Data Flow Diagram (DFD) is a graphical representation that shows how data moves through a system — how it enters, is processed, and stored. It helps in understanding the logic of the system without going into the technical implementation details.



Detailed Proposal

The **aim** of the Vehicle Performance Dashboard is to provide a **real-time, interactive platform** to monitor, analyze, and visualize vehicle sensor data. It helps users or fleet managers to **track vehicle performance**, **identify issues**, and **make informed maintenance or operational decision.**

Key Features:

- **Real-time Sensor Data Monitoring**: Continuously displays live vehicle metrics such as speed, engine temperature, fuel level, etc.
- **Interactive Web Interface**: Built using HTML, CSS, and JavaScript for a responsive and user-friendly experience.
- **Performance Analytics**: Provides insights through data visualization, helping to identify trends and anomalies.
- **Database Integration**: Uses PostgreSQL and JDBC to securely store and retrieve vehicle sensor data.
- **Alert Mechanism (optional)**: Sends alerts when sensor readings exceed predefined thresholds (e.g., overheating, low fuel).
- **Modular Architecture**: Separates frontend, backend, and data access logic to ensure maintainability and scalability.
- **Data History and Reporting**: Maintains historical records for generating performance reports and planning maintenance.

Project Scope

To make the system more powerful, user-friendly, and industry-ready, the following enhancements are proposed.

Real-Time Sensor Hardware Integration
Connect the backend with physical sensors using devices like Arduino, Raspberry Pi, or OBD-II interfaces. This would make the system suitable for direct deployment in vehicles.

Mobile App Interface
Develop a cross-platform mobile application (Android/iOS) or a responsive Progressive Web App (PWA) for field engineers to access live data from mobile devices.

Predictive Data Analytics
Use machine learning models to analyze historical data and predict issues like engine

overheating, low battery, poor fuel efficiency, or worn-out brake pads

Multi-Vehicle Support
Extend the system to simultaneously monitor and display data from multiple vehicles. This is useful for fleet owners, transport companies, or vehicle testing labs.

Security Upgrades
Implement role-based access, password encryption using hashing (e.g., bcrypt), token-based authentication (JWT), and multi-factor authentication (MFA).

Real-Time Alerts & Notifications
Add SMS or email notification services for critical events like:

Offline Data Caching
Use service workers or IndexedDB in browsers to temporarily store data and update the dashboard even when disconnected from the network.

Cloud Hosting and Scalability
Deploy the project on cloud platforms such as AWS, Microsoft Azure, or Google Cloud Platform to allow real-time access, load balancing, and secure backup of sensor data.

Report Generation Tools
Enable users to export vehicle performance summaries as PDF, CSV, or Excel files with charts and summaries.

Third-Party API Integration
Integrate mapping APIs (e.g., Google Maps) to show vehicle location data or use weather APIs to consider external factors affecting vehicle performance.

Enhanced Visualization Tools
Implement libraries such as Chart.js, D3.js, or Highcharts for advanced and interactive graphs and charts.

Voice Assistant or Chatbot Integration

Add a smart assistant feature to help users query the system using natural language (e.g., “Show me yesterday’s fuel efficiency report.”)

System Design

The Vehicle Performance Dashboard system is built on a modular architecture that separates frontend, backend, and database layers. This helps ensure flexibility, scalability, and ease of maintenance. The primary components involved are:

- **Frontend** – built using HTML, CSS, and JavaScript
- **Backend** – implemented using Java 8 with modern Java features and Collections API
- **Database** – PostgreSQL for structured sensor data storage
- **Development Environment** – VS Code used as the primary IDE

5.1 System Architecture

The Vehicle Performance Dashboard uses a three-tier architecture:

- **Presentation Layer (Frontend)**: Built with HTML, CSS, and JavaScript. It is responsible for displaying real-time sensor data and handling user interactions.
- **Application Layer (Backend)**: Developed using Java (Java 8 and Collections API). It processes requests, applies business logic, and communicates with the database.
- **Data Layer (Database)**: PostgreSQL is used to store sensor data, user details, and configuration settings.

Implementation Plan

Introduction

The implementation phase focuses on converting the design and planned architecture into working code. The project was developed using **Java 8** for backend logic, **PostgreSQL** as the relational database, and **HTML, CSS, JavaScript** for the frontend interface. The development was done in **Visual Studio Code (VS Code)** environment.

Technologies Used

Java 8
Java 8 is used for implementing backend logic and API development. Its robust object-oriented features and lambda expressions enhance code efficiency and readability.

Collection API
Java's Collection API manages and processes in-memory data structures like lists and maps. It enables efficient storage, retrieval, and manipulation of sensor data.

JDBC
Java Database Connectivity (JDBC) provides a standardized way to connect the Java application to the PostgreSQL database. It facilitates executing SQL queries and managing data transactions.

PostgreSQL
PostgreSQL is an open-source relational database used to store sensor data securely. It ensures reliable data persistence and supports complex queries for analytics.

HTML/CSS
HTML and CSS are used to structure and style the web dashboard. They ensure a clean layout and responsive design for a better user experience.

JavaScript
JavaScript adds interactivity and dynamic updates to the dashboard. It allows real-time display of sensor metrics without page reloads.

VS Code
Visual Studio Code is the development environment used for coding the application. It supports Java, web technologies, and project integration for efficient development.

Database Implementation

The project uses **PostgreSQL** as the backend database to store and manage sensor data from vehicles. The database is designed to handle large volumes of real-time data efficiently and securely.

- **Schema Design**: Tables are created to store vehicle sensor data such as speed, engine temperature, fuel level, RPM, and other performance metrics. Each record typically includes a timestamp, vehicle ID, and corresponding sensor values.
- **JDBC Connectivity**: The application connects to PostgreSQL using **JDBC (Java Database Connectivity)**. JDBC enables executing SQL queries from the Java backend to perform **CRUD operations** (Create, Read, Update, Delete).
- **Data Insertion**: Sensor data is inserted into the database either in real-time or at regular intervals. The backend ensures data validation before insertion to maintain consistency.
- **Data Retrieval**: The dashboard retrieves relevant sensor information from the database using SQL queries. This data is then processed and sent to the frontend for real-time visualization.
- **Error Handling & Transactions**: Proper exception handling and transaction management are implemented to ensure data integrity and prevent corruption during database operations.
- **Scalability**: PostgreSQL's indexing and relational capabilities help in querying and scaling data efficiently, especially when dealing with multiple vehicles or long-term performance data.

References:

1. Oracle Java Documentation – https://docs.oracle.com/javase/8/docs/
2. PostgreSQL Official Documentation – https://www.postgresql.org/docs/
3. Mozilla Developer Network (MDN Web Docs) – https://developer.mozilla.org/
4. W3Schools – https://www.w3schools.com/
5. GeeksforGeeks – https://www.geeksforgeeks.org/
6. Stack Overflow – https://stackoverflow.com/