Autonomous Fleet Management System

This comprehensive system integrates Java Servlet technology, MySQL database, and web technologies to efficiently manage and monitor a fleet of autonomous vehicles. The system provides real-time tracking, maintenance logging, and route management capabilities, all accessible through a responsive web interface.

The architecture seamlessly connects user interactions from the browser to the Java Servlet backend, which communicates with a MySQL database to store and retrieve vehicle data. This robust setup enables fleet managers to oversee vehicle statuses, locations, and battery levels with ease and precision.

System Architecture

1

User (Browser/Client)

Initiates requests and receives responses through the web interface.

2

Servlet (Backend)

Processes requests, interacts with the database, and sends responses.

3

Database (MySQL)

Stores and manages vehicle data, maintenance logs, and route information.

The system's architecture facilitates a seamless flow of information from user requests through the Java Servlet to the MySQL database and back. This design ensures efficient data management and real-time updates for the fleet management system.

Servlet Logic (Java)

doGet() Method

Handles GET requests by fetching vehicle data from MySQL using JDBC. It executes queries and returns data as a JSON array.

doPost() Method

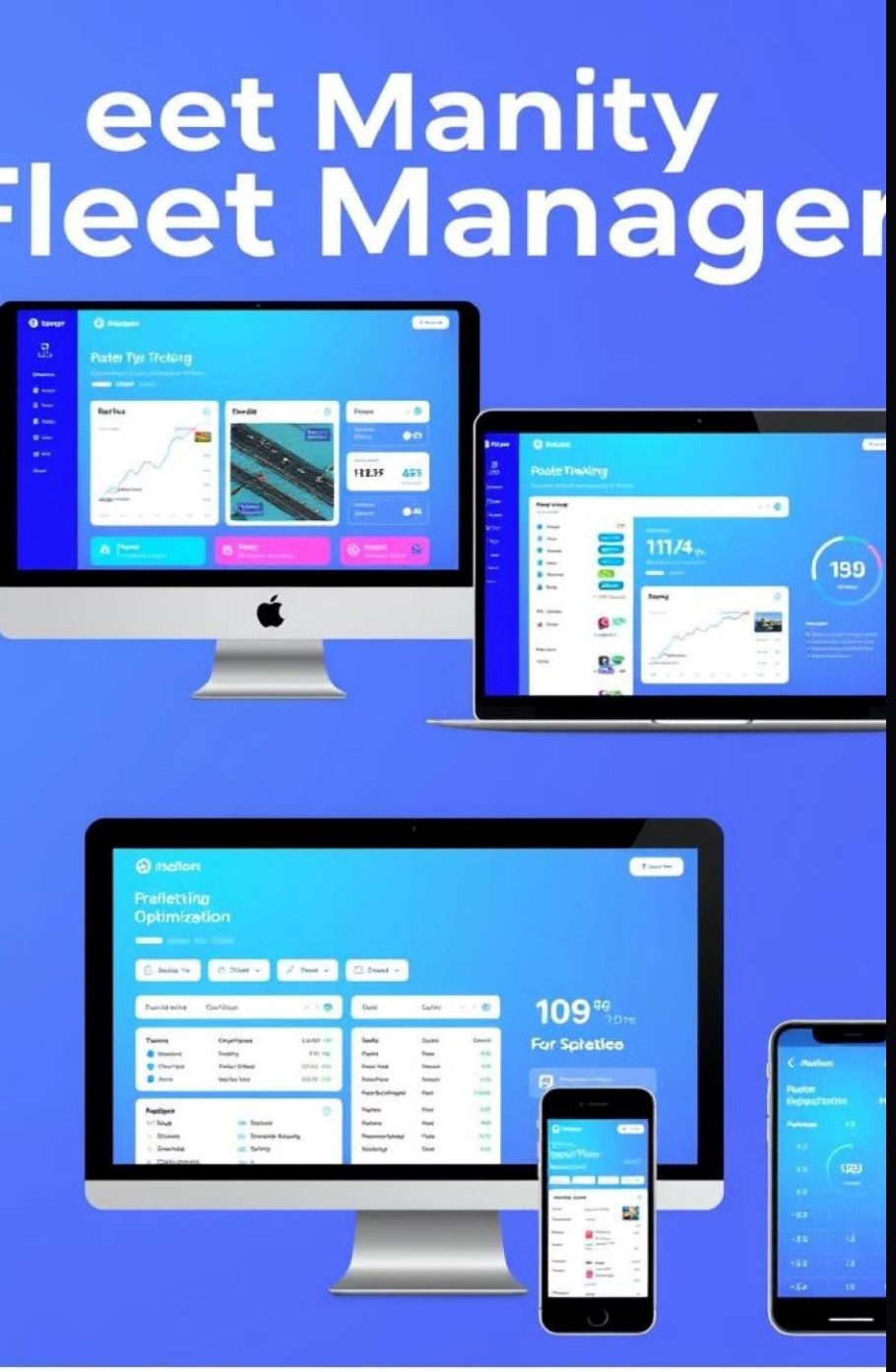
Processes POST requests for vehicle updates, including status, location, and battery level. It uses PreparedStatement to securely update the database.

The servlet acts as the core of the backend, managing database connections and executing SQL queries. It ensures efficient data retrieval and updates, providing a robust interface between the client and the database.

Database Schema (MySQL)

Table	Key Fields
vehicles	vehicle_id, status, current_location, battery_level
maintenance_logs	log_id, vehicle_id, maintenance_date, description
routes	route_id, vehicle_id, start_location, end_location

The MySQL database schema consists of three main tables: vehicles, maintenance_logs, and routes. These tables are interconnected through foreign key relationships, allowing for comprehensive tracking of vehicle information, maintenance history, and route details.



Front-End Design (HTML/CSS)

1 HTML Structure

Includes sidebar navigation, main content area with statistics and real-time map, and modal for vehicle details.

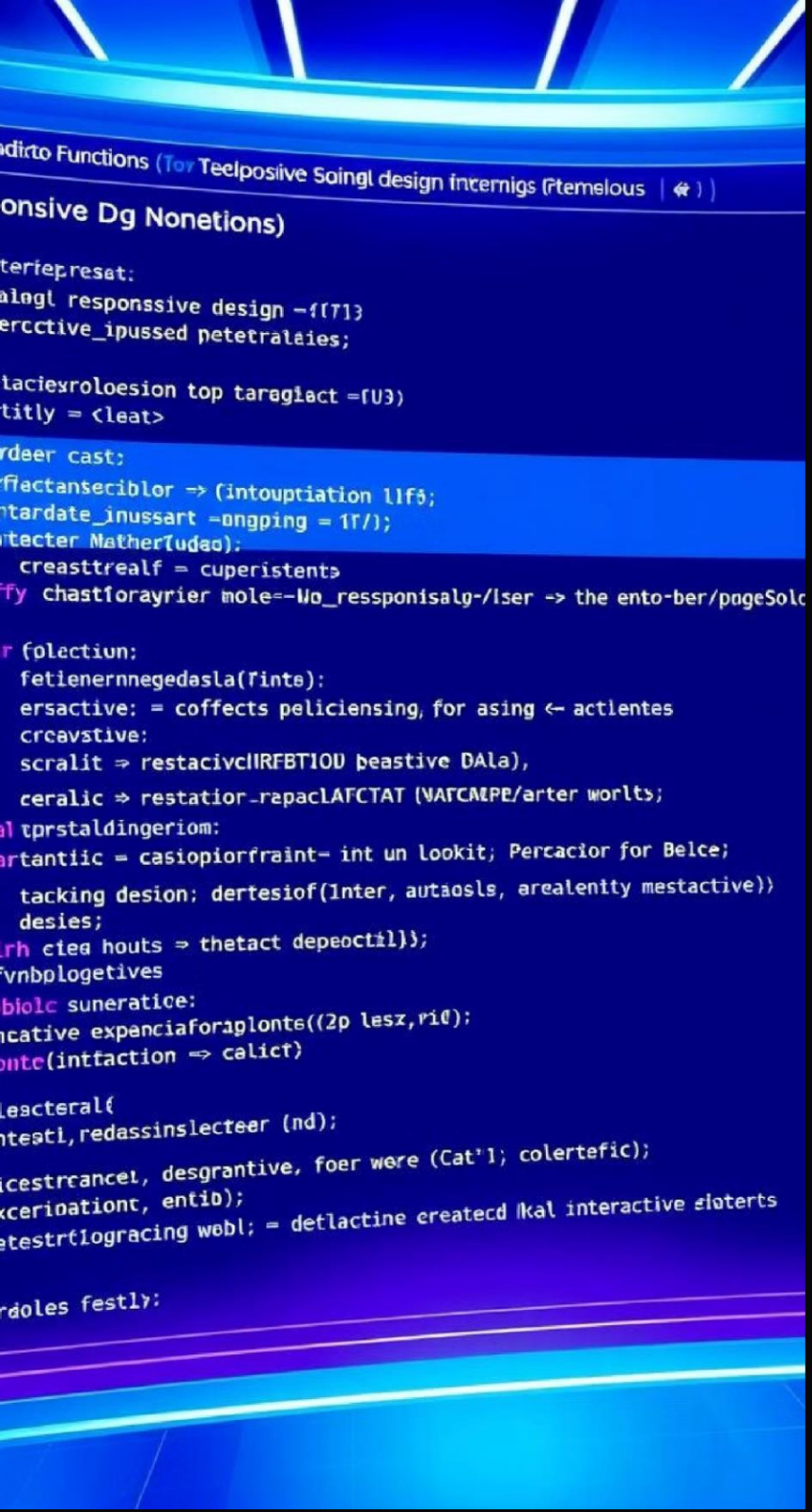
2 CSS Responsive Design

Utilizes media queries for mobile, tablet, and desktop views, ensuring a seamless user experience across devices.

3 Layout Grid

Implements a flexible grid system for organizing statistics, vehicle cards, and map view elements.

The front-end design focuses on responsiveness and user-friendly layout, adapting to various screen sizes while maintaining functionality and aesthetics.



JavaScript Functionality

1. **Mobile Menu Handling**

   Implements sidebar toggle functionality for mobile devices.

2. **Dynamic Grid Layouts**

   Adjusts grid layouts based on screen size for optimal viewing.

3. **Modal Implementation**

   Creates interactive modals for displaying detailed vehicle information.

4. **Responsive Event Listeners**

   Handles resize events and swipe gestures for improved user interaction.

JavaScript enhances the user interface with dynamic content loading, responsive behaviors, and interactive elements, ensuring a smooth and engaging user experience across devices.

Database Operations (SQL)

Fetching Vehicle Data

SELECT * FROM vehicles;

Updating Vehicle Status

UPDATE vehicles SET status=?, current_location=?, battery_level=? WHERE vehicle_id=?;

Inserting Maintenance Log

INSERT INTO maintenance_logs (vehicle_id, maintenance_date, description) VALUES (?, ?, ?);

These SQL operations demonstrate the core database interactions for retrieving vehicle information, updating statuses, and logging maintenance activities. The use of prepared statements ensures security against SQL injection attacks.

System Features

Vehicle Tracking

Real-time updates on vehicle status, location, and battery levels.

Maintenance Monitoring

Comprehensive logging of vehicle maintenance records.

Route Management

Tracking of vehicle routes, including start and end locations, and distances covered.

Analytics & Settings

Placeholder for future extensions in data analysis and system configuration.

These features form the core functionality of the Autonomous Fleet Management System, providing comprehensive tools for efficient fleet oversight and optimization.

Error Handling & Security

Error Handling

The servlet implements robust exception handling, returning error messages as JSON responses. This includes handling SQL errors, malformed queries, and database connection issues, ensuring graceful degradation of the system.

Security Considerations

While authentication and access control are not fully implemented, the system uses prepared statements to prevent SQL injection attacks. Future enhancements should include user authentication and role-based access control for improved security.

Effective error handling and security measures are crucial for maintaining system integrity and protecting sensitive fleet data.

Future Enhancements

1

Real-time Tracking

Implement live vehicle tracking on an interactive map interface.

2

GPS Integration

Integrate with GPS systems or Google Maps API for enhanced location services.

3

WebSocket Implementation

Utilize WebSocket technology for real-time data updates without polling.

4

Advanced Analytics

Develop comprehensive analytics tools for vehicle performance metrics and fleet optimization.

These proposed enhancements aim to further improve the system's capabilities, providing more accurate tracking, real-time updates, and advanced data analysis for optimized fleet management.