Connected Car Fleet Management System

Problem Statement

Design and implement a **Connected Car Fleet Management System** that manages vehicle fleets across multiple manufacturers, processes real-time telemetry data, and provides analytics for fleet optimization.

Phase 1: Core Implementation
(4 Hours - LLM Usage **NOT ALLOWED)**

Development Approach

Focus on fundamentals first. Ensure basic functionality is working completely before attempting advanced features. A working basic system is better than an incomplete advanced one.

Follow clean coding practices:

Core Functionality Requirements

Vehicle Fleet Management

- The system manages vehicles from **multiple manufacturers** (Tesla, BMW, Ford, Toyota, etc.)
- Each vehicle has:
  - **VIN** (Vehicle Identification Number) - unique identifier
  - **Manufacturer** and **Model**
  - **Fleet ID** (vehicles can belong to different fleets like "Corporate", "Rental", "Personal")
  - **Owner/Operator** information
  - **Registration status** (Active, Maintenance, Decommissioned)

Real-time Telemetry Data Processing

- Vehicles send telemetry data every **30 seconds** containing:
  - **GPS coordinates** (latitude, longitude)
  - **Speed** (current speed in km/h)
  - **Engine status** (On/Off/Idle)

    - **Fuel/Battery level** (percentage)
    - **Odometer reading** (total kilometers)
    - **Diagnostic codes** (if any errors)
    - **Timestamp** of the reading

Basic Analytics

- Provide fleet-level analytics:
    - **Active vs Inactive vehicles** count
      Inactive - if a vehicle has not received data for the last 24 hours
    - **Average fuel/battery levels** across fleet
    - **Total distance traveled** by fleet in last 24 hours
    - **Alert summary** (count by type and severity)

Alert System

- Generate alerts based on telemetry data:
    - **Speed violations** (exceeding predefined speed limits)
    - **Low fuel/battery** (below 15%)

API Endpoints Required

- **Vehicle Management** - **create, list, query, delete vehicles**
- **Telemetry Data** - **receive telemetry data** for **specific** and **multiple** vehicles, query for **telemetry history** and **latest telemetry**
- **Alerts - query for all alerts / by alert id**

Data Storage

Implement data persistence using your preferred approach:

- **Option 1:** Start with in-memory storage (arrays, maps) for rapid prototyping, then migrate to database in Phase 2
- **Option 2:** Begin with a database from the start if you're comfortable with database setup

Note: Both approaches are acceptable. Choose based on your experience and time management.

Post-Phase 1: System Analysis (Before Phase 2)

Before moving to Phase 2 enhancements, take time to analyze your system:

Identify Edge Cases: Review your Phase 1 implementation and identify scenarios that could break your system. Consider data anomalies, connectivity issues, invalid inputs, and system failures that real-world connected car systems face.

Document Weaknesses: Make a list of areas where your current system is vulnerable or could be improved. This analysis will guide your Phase 2 improvements.

Phase 2: Enhanced System (4 Hours - LLM Usage **Allowed)**

Development Strategy

Build upon your working Phase 1 system. Only move to performance optimizations after ensuring all basic functionality is solid and tested.

Recommended priority order:

1. First, ensure Phase 1 features are robust and handle edge cases properly
2. Add Docker containerization for basic deployment
3. Then progressively add performance and advanced features

Areas to Consider and Improve

- **System robustness -** address edge cases and weaknesses identified in your analysis
- **Database optimization -** migrate from in-memory storage or optimize existing database queries
- **Concurrency**
- **Caching**
- **Rate limiting**
- **API optimizations**
- **Authentication**
- **Containerization - Dockerize the application for deployment**

Bonus Features (If Time Permits)

Simple Analytics

- **Real-time dashboard** showing live vehicle count and alerts
- **Basic charts** for telemetry trends
- **CSV export** functionality for analytics data

Technology Stack

You are free to choose any programming language, framework, and database that best suits the problem. Popular choices include Node.js/Express, Python/FastAPI, Java/Spring Boot, or similar.

Evaluation Criteria

This assignment tests your ability to handle real-world connected car platform challenges including data processing, analytics, and integration with modern AI tools.

Phase 1 Focus:

- **Working core functionality -** all basic features must work reliably
- **Clean code architecture -** proper use of OOP principles, modular design
- **API design and data modeling -** well-structured endpoints and database schema
- **Basic error handling -** system handles invalid inputs gracefully

Phase 2 Focus:

- **System optimization** built on top of working Phase 1 foundation
- **Modern development practices -** containerization and deployment readiness
- **Effective AI tool integration** for development acceleration
- **Performance improvements** only after core functionality is solid

Remember: A complete, working basic system scores higher than an incomplete advanced system.

Submission Requirements

- **Working application** with all core APIs implemented
- **Version control -** Git repository pushed to GitHub with clear commit history
- **Containerized deployment -** Docker configuration with setup instructions
- **Code documentation** explaining design decisions and architecture
- **Demonstration** of key features and system capabilities