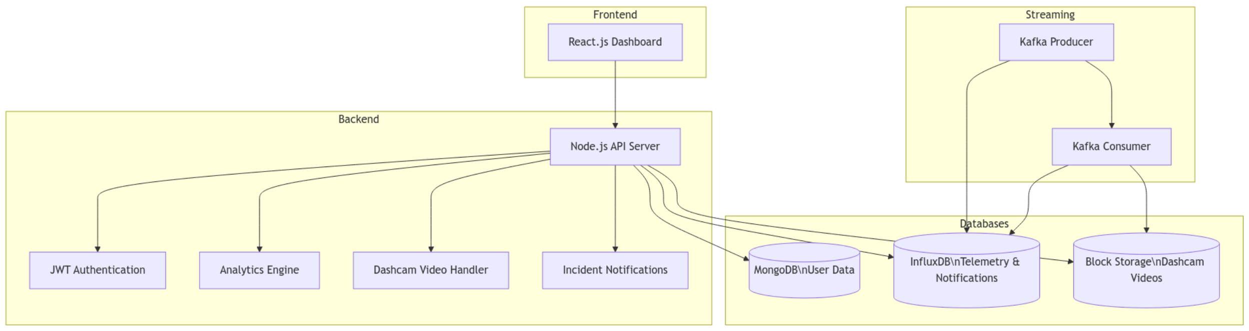
Fleet Management System

The Fleet Management System is designed to monitor and analyze vehicle fleets in real-time. It integrates GPS data, vehicle telemetry, and dashcam footage into a unified platform that provides real-time analytics, incident detection, and role-based management for fleet operators.

High-Level Overview

The system is a **microservices-based architecture** with clear separation of concerns. It uses **Kafka** for real-time data streaming, **InfluxDB** for telemetry and notifications storage, **MongoDB** for user authentication, and React for the frontend. It is built to be horizontally scalable, capable of handling large amounts of telemetry and video data while providing insights to fleet operators.



Core Features

1. **Real-Time Vehicle Monitoring**:
    - Tracks vehicle telemetry data (GPS, speed, fuel, etc.).
    - Maps vehicle locations and status on a live dashboard.
2. **Incident Detection**:
    - Analyzes sudden speed drops or anomalies to detect accidents.
    - Generates notifications and stores them in InfluxDB.
3. **Dashcam Footage Management**:
    - Streams and stores dashcam videos in 10-second chunks.
    - Allows operators to retrieve video footage by timestamp.
4. **Analytics Dashboard**:
    - Provides fleet-wide performance metrics, including average speed, fuel efficiency, and mileage.
5. **Role-Based Access Control**:
    - Supports multi-user access with authentication and role-specific views.

Implemented Functionality

1. User Authentication

Files:

- Frontend: `frontend/src/components/Login.js`, `frontend/src/components/Register.js`
- Backend: `backend/routes/auth.js`

Purpose:

- Provides a secure login and registration system for users.
- Role-based access ensures different levels of permission for admins, managers, and drivers.

How It Works:

- User data (username, password, role) is stored in **MongoDB**.
- Upon login, a **JWT (JSON Web Token)** is generated and returned to the frontend.
- The JWT is used to secure subsequent API requests.

Technologies:

- **MongoDB**: Chosen for its schema-less nature, making it flexible for storing user data.
- **JWT**: Ensures stateless, secure authentication.

---

2. Real-Time Telemetry Data Processing

Files:

- Backend: `backend/routes/analytics.js`

Purpose:

- Tracks and stores telemetry data (speed, fuel level, GPS, etc.) for real-time analytics and monitoring.

How It Works:

1. A **Kafka producer** streams telemetry data from vehicles to a Kafka topic.

2. A **Kafka consumer** reads this data and stores it in **InfluxDB**.
3. InfluxDB is queried to retrieve data for:
    - Live dashboards.
    - Historical trends and analytics.

Technologies:

- **Apache Kafka**: Handles high-frequency streaming of telemetry data.
- **InfluxDB**: Optimized for time-series data like telemetry, enabling fast and efficient queries.

---

3. Analytics Dashboard

Files:

- Frontend: `frontend/src/components/Analytics.js`
- Backend: `backend/routes/analytics.js`

Purpose:

- Visualizes key performance indicators (KPIs) such as:
    - Average speed.
    - Total mileage.
    - Fuel consumption trends.
    - Accident detection alerts.

How It Works:

1. The backend queries telemetry data from **InfluxDB** using its timestamp-based indexing.
2. Data is processed into meaningful metrics.
3. The frontend renders the data using charts and graphs for user-friendly visualization.

Features:

- Real-time updates.
- Historical data visualization for up to 30 days.

---

4. Incident Notifications

Files:

- Backend: `backend/routes/notifications.js`

Purpose:

- Detects incidents (e.g., accidents based on speed drop patterns).
- Generates and stores notifications in **InfluxDB** for easy retrieval.

How It Works:

1. Telemetry data is analyzed in real-time.
2. Anomalies (e.g., speed dropping to 0) trigger notifications.
3. Notifications are stored with timestamps in InfluxDB.

Why InfluxDB for Notifications?

- **Time-series optimization**: Perfect for storing timestamped events like notifications.
- **Scalable storage**: Handles high-frequency notifications efficiently.

---

5. Dashcam Footage Management

Files:

- Frontend: `frontend/src/components/VideoPlayback.js`
- Backend: `backend/routes/video.js`

Purpose:

- Stores and streams dashcam videos in 10-second chunks for playback.

How It Works:

1. **Kafka Producer** streams video chunks from vehicles to a Kafka topic.
2. **Kafka Consumer**:
   - Renames videos with timestamps.
   - Stores videos in a **block storage** system (local or cloud).
3. Videos can be retrieved via timestamp and played back in the frontend.

Features:

- Efficient chunk-based storage to reduce latency.
- Simple retrieval using timestamps.

System Architecture

Component	Technology	Purpose
Frontend	React.js	User interface for real-time tracking and analytics.
Backend	Node.js with Express.js	API server to handle business logic and integrations.
Database	MongoDB	User authentication and role management.
	InfluxDB	Time-series database for telemetry data and notifications.
Streaming	Apache Kafka	Real-time streaming of telemetry data and dashcam footage.
Storage	Block storage (e.g., AWS S3 or local)	Dashcam video storage for playback.

Technologies Used

Why We Used InfluxDB and MongoDB

InfluxDB

- Designed for **time-series data**, which is ideal for telemetry and notifications.
- **Efficient querying** of timestamped data, enabling real-time dashboards and analytics.

- **High throughput**: Handles large volumes of incoming telemetry data with minimal latency.

Use Cases:

- Storing and querying vehicle telemetry data.
- Storing incident notifications with timestamps.

MongoDB

- Flexible **document-based schema**, ideal for storing user credentials and role data.
- Simplifies user management for authentication and authorization.
- Scalability: Easily handles growing numbers of users and roles.

Use Cases:

- User authentication and role-based access control.

Future Enhancements

1. **Video Analysis**:
   - Implement AI/ML to analyze dashcam footage for events like harsh braking or lane changes.
2. **Scalable Deployment**:
   - Use container orchestration tools like Kubernetes to manage scalability.
3. **Advanced Analytics**:
   - Add predictive analytics for maintenance scheduling based on telemetry trends.