NeuroFleetX: AI-Driven Urban Mobility Optimization  
  
  
**Frontend**: Choose **React** with **React Hooks** over Angular. React's component-based architecture and unidirectional data flow make it highly performant and flexible for building dynamic user interfaces, which is crucial for a real-time dashboard. The large community and rich ecosystem of libraries make it easier to find solutions and scale.

* **Backend**: **Spring Boot with a REST API** is an excellent choice. It provides a robust, scalable, and secure foundation. Spring Boot's dependency injection and convention-over-configuration approach accelerate development. It pairs perfectly with **Spring Security + JWT** for a secure, stateless authentication and authorization system, essential for protecting API endpoints and managing different user roles.
* **AI/ML Logic**: Use **Python with Flask microservices**. While JavaML exists, Python is the de-facto standard for AI and machine learning due to its extensive libraries (TensorFlow, PyTorch, Scikit-learn). By encapsulating your AI logic in Flask microservices, you can keep your AI models decoupled from your main Spring Boot backend. Spring Boot can then communicate with these Flask services via internal REST API calls, allowing each part to scale independently.
* **Database**: **MySQL** is a reliable and powerful choice for a relational database. It's well-suited for storing structured data like user profiles, fleet information, vehicle data, and historical traffic logs.
* **Mapping**: Use **Google Maps API**. While Leaflet.js is a great open-source option, Google Maps offers more robust features out-of-the-box, including detailed routing, real-time traffic data, and a vast ecosystem of additional services (like geocoding and directions). Given the project's focus on urban mobility, these features will be invaluable. The cost is a consideration, but for a professional-grade project, the features and reliability are worth it.
* **Data Visualization**: **Recharts** is the ideal choice. Since you're using React, Recharts is built specifically for it and provides a declarative, component-based approach to creating beautiful and complex charts. This aligns perfectly with the React frontend and allows for seamless integration and customization.
* **Other APIs**:
  + **WebSocket for live updates**: This is critical for a mobility project. Use WebSocket to push real-time data from the backend to the frontend, such as a vehicle's live location, traffic condition updates, or new optimization recommendations.
  + **SMTP**: Essential for sending email notifications, such as security alerts, system reports, or user updates.

**Key Features to Add to Your Project**

Here are some features you can build using this tech stack, broken down by project area:

**Core Mobility Features**

* **Dynamic Route Optimization**: The AI logic (Python/Flask) can use real-time traffic data from Google Maps to calculate the most efficient routes for a fleet of vehicles, considering factors like time, fuel consumption, and road closures.
* **Predictive Traffic Flow**: Implement a machine learning model to predict traffic congestion in specific city zones at different times of the day, allowing the system to suggest alternative routes proactively.
* **Live Fleet Tracking**: A dashboard that uses WebSockets to show the real-time location of all vehicles on the Google Maps interface. This is crucial for fleet managers to monitor operations.
* **Demand Prediction**: An AI model that analyzes historical ride-sharing or public transit data to predict areas of high demand, allowing for optimal vehicle dispatching.

**User & Administrative Features**

* **Role-Based Access Control**: Use **Spring Security with JWT** to define roles like "Admin," "Fleet Manager," and "Driver." An Admin can manage users and system settings, a Fleet Manager can see fleet-wide data, and a Driver can only see their own assigned tasks.
* **Comprehensive Dashboard**: Use React and Recharts to create a dashboard with graphs and charts showing key performance indicators (KPIs) like average trip time, fuel efficiency, emissions savings, and fleet utilization rates.
* **User Authentication**: Implement a secure login system using JWT for token-based authentication. This allows for a stateless backend, which is highly scalable.

**Communication & Data Features**

* **Real-time Alerts**: The backend (Spring Boot) can trigger alerts (via WebSockets to the frontend and SMTP for email) for events like a vehicle deviating from its route, a sudden surge in traffic, or a system error.
* **Historical Data Analysis**: The MySQL database can store all historical data, enabling the system to run complex analytics and generate reports that help city planners or fleet managers make long-term strategic decisions.
* **Scalable Architecture**: The use of microservices (Spring Boot and Flask) ensures that as your project grows, you can independently scale the parts that receive the most load, such as the AI logic or the core REST API.

ere is a detailed specification for the NeuroFleetX project, broken down into functional and non-functional requirements.

**1. Functional Requirements**

* **User Management & Access Control**
  + **FR-1.1**: The system must allow users to register, log in, and log out securely.
  + **FR-1.2**: The system must support at least three distinct user roles: **Admin**, **Fleet Manager**, and **Driver**.
  + **FR-1.3**: The system must enforce Role-Based Access Control (RBAC) to restrict access to specific APIs and UI components based on the user's role.
* **Vehicle & Fleet Management**
  + **FR-2.1**: The system must provide an API and UI for an Admin to perform CRUD (Create, Read, Update, Delete) operations on vehicle data.
  + **FR-2.2**: The system must display a list of all vehicles to any authenticated user.
  + **FR-2.3**: The system must display a real-time map showing the current location of all active vehicles.
  + **FR-2.4**: The system must simulate a real-time stream of vehicle location data for testing and demonstration purposes.
* **AI-Driven Optimization (Planned)**
  + **FR-3.1**: The system must have a dedicated Python microservice for AI/ML logic.
  + **FR-3.2**: The AI microservice must implement a **predictive maintenance** model to forecast potential vehicle failures based on simulated IoT sensor data.
  + **FR-3.3**: The AI microservice must implement a **dynamic route optimization** model that provides real-time route recommendations based on traffic, weather, and road closure data.
* **Dashboard & Visualization**
  + **FR-4.1**: The system must provide a dedicated dashboard for Fleet Managers and Admins.
  + **FR-4.2**: The dashboard must display key performance indicators (KPIs) such as vehicle utilization, total trips, and fuel efficiency in a visual format (charts, graphs).
* **Notifications**
  + **FR-5.1**: The system must send real-time alerts to users for critical events (e.g., maintenance flags, route deviation). This can be done via push notifications or email.

**2. Non-Functional Requirements**

* **Performance**: The real-time tracking system must have a latency of less than 2 seconds between a simulated location update and its display on the map.
* **Scalability**: The system's microservices architecture must allow for independent scaling of the Node.js backend and the Python AI microservice.
* **Security**: All API endpoints must be secured using JWT-based authentication. User passwords must be stored securely using hashing.
* **Usability**: The frontend must be intuitive, responsive, and easy to use on both desktop and mobile devices.

**Test Cases**

Here are some example test cases to validate the functionality of the NeuroFleetX platform. This is a mix of functional, integration, and user-flow tests.

**1. Authentication & Authorization Tests**

* **Test Case ID**: AU-01
* **Test Case Name**: Admin Login and RBAC
* **Test Steps**:
  1. Log in as a user with the 'admin' role.
  2. Navigate to the 'Vehicle Management' page.
  3. Verify that 'Add', 'Update', and 'Delete' buttons are visible.
  4. Log out.
  5. Log in as a user with the default 'user' role.
  6. Navigate to the 'Vehicle Management' page.
  7. Verify that 'Add', 'Update', and 'Delete' buttons are **not** visible.
* **Expected Result**: The 'admin' user can see and access all CRUD features, while the 'user' cannot.

**2. Real-Time Tracking Tests**

* **Test Case ID**: RT-01
* **Test Case Name**: Live Vehicle Location Update
* **Test Steps**:
  1. Log in as any authenticated user.
  2. Navigate to the main dashboard with the map.
  3. Observe a vehicle's icon on the map.
  4. Simulate a location update for that vehicle from the backend (e.g., via a manual trigger or script).
  5. Verify that the vehicle's icon on the frontend map moves to the new location within 2 seconds.
* **Expected Result**: The vehicle's icon moves smoothly on the map in near real-time, reflecting the backend update.

**3. API & Data Integrity Tests**

* **Test Case ID**: API-01
* **Test Case Name**: Vehicle Creation by Admin
* **Test Steps**:
  1. Send a POST request to the /api/vehicles endpoint with valid vehicle data while authenticated as an Admin.
  2. Verify the response status code is 201 Created.
  3. Send a GET request to the /api/vehicles endpoint.
  4. Verify that the newly created vehicle is present in the response payload.
* **Expected Result**: A new vehicle is successfully created in the database and is visible via the API.

**4. Planned Feature (AI) Tests**

* **Test Case ID**: AI-01
* **Test Case Name**: Predictive Maintenance Alert Trigger
* **Test Steps**:
  1. Start the Python AI microservice.
  2. Send simulated sensor data to the microservice that meets the criteria for a high-risk maintenance flag (e.g., extremely low tire pressure).
  3. Verify that the microservice's API returns a maintenance alert.
  4. Verify that the Node.js backend receives this alert and sends a notification to the Fleet Manager's email (or a console log in the test environment).
* **Expected Result**: The AI model correctly identifies a maintenance risk and triggers an alert through the notification system.