NeuroFleetX Documentation

AI-Driven Urban Mobility Optimization Platform

# 1. Problem Statement

Urban transportation faces challenges like:  
- Increasing traffic congestion leading to time delays.  
- Inefficient fleet utilization, with empty or underutilized vehicles.  
- Lack of real-time traffic prediction and adaptive route management.  
- Difficulty in forecasting passenger demand during peak and off-peak hours.  
- Rising fuel costs and carbon emissions, affecting sustainability goals.  
- Limited integration between different transport modes (bus, metro, taxi, EVs, bike sharing).  
  
NeuroFleetX addresses these problems by applying AI-driven optimization to reduce congestion, maximize fleet usage, lower costs, and enhance passenger experience.

# 2. Technology Stack

## Core AI & Infrastructure

- AI/ML Frameworks: TensorFlow, PyTorch, Scikit-learn  
- Big Data Processing: Apache Kafka, Apache Spark  
- Databases: PostgreSQL, MongoDB, InfluxDB (for time-series)  
- Cloud Platforms: AWS / Azure / GCP  
- IoT & Edge Computing: MQTT, Node-RED, Edge AI devices

## MERN Stack (Web Platform & Dashboard)

- MongoDB: Stores mobility data (routes, trips, fleet logs, passengers).  
- Express.js: Backend framework for building REST APIs and GraphQL endpoints.  
- React.js: Frontend UI for dashboards, operator controls, and passenger apps.  
- Node.js: Server-side runtime for managing requests, AI model APIs, and integration services.  
- Additional Tools: TailwindCSS, Bootstrap, GTFS, Google Maps API, OpenWeather API, JWT, OAuth 2.0

# 3. Approach

Step 1: Define Scope & MVP  
- Start with core features like real-time fleet tracking, route optimization, operator dashboard, and basic demand prediction.  
  
Step 2: System Architecture Blueprint  
- Data Sources: GPS, traffic APIs, weather data, public transport schedules.  
- Backend: Node.js + Express.js for APIs and AI microservices integration.  
- Database: MongoDB collections for vehicles, routes, trips, traffic\_data, users.  
- Frontend: React.js dashboard and passenger app.  
- AI/ML Module: Demand forecasting, route optimization, congestion prediction.  
  
Step 3: Setup Tech Stack  
- Initialize MERN stack.  
- Setup AI microservices with Python or TensorFlow.js.  
- Integrate APIs like Google Maps and OpenWeather.  
  
Step 4: Development Phases  
- Phase 1: Fleet tracking + live map.  
- Phase 2: AI demand prediction.  
- Phase 3: Route optimization.  
- Phase 4: Full operator dashboard.  
- Phase 5: Passenger-facing app.  
  
Step 5: Deployment  
- Use Docker containers.  
- Deploy MERN app on AWS/Azure/Heroku.  
- Setup CI/CD with GitHub Actions or Jenkins.  
  
Step 6: Future Enhancements  
- Edge computing for real-time traffic.  
- Blockchain for secure ticketing.  
- Autonomous vehicle integration.

# 4. Development Plan (Step-by-Step)

This plan outlines an 8-week roadmap to build NeuroFleetX using the MERN stack with AI microservices. Each week includes focus areas, key tasks, and concrete deliverables.

|  |  |  |  |
| --- | --- | --- | --- |
| Week | Focus | Key Tasks | Deliverables |
| 1 | MERN Scaffold | - Repo setup, code style, Docker baseline - Express API + MongoDB connection - React app scaffold (Vite/CRA) + UI skeleton | ✅ Running MERN boilerplate |
| 2 | Data Model & Ingestion | - Define Mongo collections: users, vehicles, routes, trips, traffic\_data - Seed with GTFS/OSM samples - Build REST endpoints for routes/vehicles | ✅ Postman collection ✅ Seeded DB + CRUD APIs |
| 3 | Real‑Time Tracking MVP | - GPS simulator + Socket.IO/WebSockets - React map (Leaflet/Google Maps) - Show live vehicle markers & status | ✅ Live map with simulated fleet |
| 4 | Demand Forecasting v1 (AI) | - Prep dataset (rides, weather, events) - Baseline model (moving average/linear reg) - Expose FastAPI/Flask /forecast service - Integrate Node → Python | ✅ /forecast API + accuracy report (MAE/RMSE) |
| 5 | Routing & Optimization v1 | - Graph build from OSM; OSRM/ custom A\*/Dijkstra - Dynamic re‑routing based on traffic - ETA computation & route suggestions | ✅ /optimize API + route results |
| 6 | Operator Dashboard & Auth | - KPIs: on‑time %, occupancy, avg wait - Alerts & CRUD tools - AuthN/AuthZ (JWT, RBAC) | ✅ Secure dashboard screens |
| 7 | Hardening, DevOps & Deploy | - Tests (Jest/Mocha, PyTest), linting - Logging/metrics; CI/CD (GitHub Actions) - Containerize & deploy (AWS/Azure/Render) | ✅ Cloud URL + load test report |

## Acceptance Criteria (per phase)

- Real-time map shows ≥ 50 vehicles updating at 1 Hz without UI freezes.  
- Forecast MAE ≤ 15% on held-out data; documented evaluation.  
- Optimization returns routes within ≤ 1s for 100 requests/min.  
- Auth: protected APIs with role-based access; tokens rotated.  
- 80% unit test coverage backend; CI passing; container images reproducible.

## Tooling & Libraries

- Backend: Node.js, Express, Mongoose, Socket.IO, Joi/Zod for validation  
- Frontend: React, React Router, TanStack Query, Leaflet or @react-google-maps/api, TailwindCSS  
- AI: Python, FastAPI, scikit-learn/PyTorch, pandas, numpy  
- Routing: OSRM/GraphHopper or custom A\*/Dijkstra with OSMnx  
- DevOps: Docker, GitHub Actions, Terraform (optional), Nginx  
- Observability: Winston/PM2 logs, OpenTelemetry, Prometheus + Grafana (optional)

## Risks & Mitigations

- Data Scarcity → Start with synthetic/SUMO data; augment with NYC TLC & OpenWeather.  
- Model Drift → Scheduled retraining and backtesting; monitor error metrics.  
- API Cost/Rate Limits → Cache responses; fall back to OSM/OSRM offline routing.  
- Performance Bottlenecks → Queue with Kafka/RabbitMQ; scale with horizontal pods/PM2 clusters.

# 5. Analysis

The analysis of NeuroFleetX focuses on evaluating system performance, impact, and feasibility.  
  
1. Data Analysis:  
- Identify peak vs. off-peak traffic patterns.  
- Evaluate fleet utilization rates.  
- Compare predicted demand vs. actual demand.  
- Identify congestion hotspots.  
  
2. System Performance Analysis:  
- Accuracy of AI models (MAE, RMSE).  
- Reduction in average waiting time.  
- Energy efficiency improvement.  
  
3. Comparative Analysis:  
- Traditional static schedules vs. AI-driven dynamic optimization.  
- Baseline vs. optimized dispatching.  
  
4. User Experience Analysis:  
- Operators: Ease of monitoring and control.  
- Passengers: Shorter wait times, better ETAs.  
- Authorities: Policy insights and planning data.  
  
5. Limitations:  
- Real-time data reliability issues.  
- Initial model accuracy may be limited with small datasets.  
- Integration with legacy systems.

# 6. Use Cases

1. Smart Public Transport:  
- Optimize bus/metro scheduling in real time.  
  
2. Ride-Hailing Services:  
- Improve dispatch efficiency and reduce passenger wait times.  
  
3. Logistics & Delivery:  
- Last-mile delivery optimization.  
  
4. EV Fleet Management:  
- Battery-aware routing and charging optimization.  
  
5. Smart City Planning:  
- Provide data insights for traffic policy, infrastructure expansion, and emission reduction.

# 7. Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Description | Input | Expected Output |
| TC-01 | Verify fleet location updates on live map | GPS coordinates every second | Map markers update in real-time without lag |
| TC-02 | Validate demand forecasting service | Historical ridership + weather data | Forecast MAE ≤ 15% |
| TC-03 | Check route optimization API | Origin, destination, live traffic data | Shortest-time route returned within 1s |
| TC-04 | Test operator dashboard authentication | Valid JWT token with operator role | Access granted; invalid/expired → denied |
| TC-05 | Verify dispatch service assignment | Incoming ride requests + idle vehicles | Passengers matched to nearest optimal vehicle |