Maritime Fleet Tracking, Operations Management & Sea-Route Visualization System

1. Introduction

The Maritime Fleet Manager is a comprehensive full-stack web application designed to simulate, track, and manage global naval operations in real-time. It provides a centralized command center for fleet operators to monitor vessel locations, manage operational status, and analyze fleet performance using live geospatial data. The system replaces static tracking methods with a dynamic, interactive interface powered by the MERN stack.

2. Objectives

Real-Time Visualization: To provide live tracking of vessels on an interactive map using geospatial coordinates.

Automated Simulation: To create a realistic backend engine that autonomously updates vessel positions, fuel consumption, and weather conditions without manual input.

Operational Efficiency: To streamline fleet management tasks such as commissioning ships, updating statuses, and logging docking events.

Security: To ensure data integrity through Role-Based Access Control (RBAC) and secure authentication.

3. System Features

Live Geospatial Tracking: Real-time rendering of vessel movements on an Esri Ocean Basemap using Leaflet.js.

Smart Routing: Autonomous pathfinding for vessels using Turf.js to navigate naturally around landmasses.

Role-Based Access Control (RBAC): Distinct permissions for Admins (full control), Operators (management), and Viewers (read-only).

Dynamic Dashboard: Visual analytics for fuel consumption, submarine depth profiles, and operational alerts.

Auditable Logs: A permanent, immutable history of all vessel docking and decommissioning events.

4. System Architecture

The application follows a **Client-Server Architecture** decoupled into a Frontend and Backend:

- **Frontend:** A React.js Single Page Application (SPA) that consumes REST APIs and renders real-time updates.
- **Backend:** A Node.js/Express REST API that handles logic, authentication, and database operations.
- **Database:** MongoDB Atlas (Cloud) serving as the NoSQL data store.
- **Simulation Service:** A background worker process running alongside the backend server to generate simulation data.

5. Technologies Used

Frontend: React.js (Vite), Tailwind CSS, React-Leaflet, Recharts, Axios, Lucide React.

Backend: Node.js, Express.js, JSON Web Token (JWT), Bcrypt, CORS.

Database: MongoDB, Mongoose ODM.

Geospatial: Turf.js (Bezier Splines & Distance calculation), Leaflet.js.

Deployment: Netlify (Frontend), Render (Backend), MongoDB Atlas.

6. Project Structure

An overview of the Monorepo directory structure:

- **backend/**: Contains the API server, database models (/models), controllers (/controllers), routes (/routes), and the simulation engine (simulationService.js).
- **frontend/**: Contains the React application, including pages (/pages), reusable components (/components), context providers (/context), and assets.

7. Installation & Setup

Step-by-step instructions to deploy the application locally:

1. **Prerequisites:** Node.js (v18+) and MongoDB URI.
2. **Backend Setup:** Install dependencies (npm install), configure .env variables (PORT, MONGO_URI, JWT_SECRET), and start the server (npm run dev).
3. **Frontend Setup:** Install dependencies, configure the API base URL in main.jsx, and start the Vite server.

8. Authentication & RBAC

Security is implemented using **JWT (JSON Web Tokens)**.

- **Login Flow:** Users exchange credentials for a signed Access Token.
- **Middleware:** The authMiddleware.js validates tokens on protected routes.
- **RBAC Logic:**
    - **Admin:** Can create users and delete vessels.
    - **Operator:** Can update vessel status and dock ships.
    - **Viewer:** Read-only access to maps and logs.

9. Data Models

Description of the MongoDB Schemas:

- **User Schema:** Stores username, password hash (Bcrypt), and role.
- **Vessel Schema:** Stores name, type (Submarine/Cargo/Destroyer), coordinates (Lat/Lng), speed, fuel level, and current status.
- **Log Schema:** Stores historical records of docking events with timestamps and locations

10. REST API Endpoints

A summary of key API routes:

- POST /api/auth/login: Authenticate user.
- GET /api/vessels: Retrieve all vessels.
- POST /api/vessels: Add a new vessel (Admin only).
- PUT /api/vessels/:id: Update vessel parameters.
- GET /api/logs: Fetch docking history.

11. Real-Time Simulation Engine

A breakdown of the simulationService.js:

- **Interval:** Runs every 5 seconds.
- **Logic:** Iterates through all "Active" vessels, calculates the next coordinate based on speed and bearing, updates fuel levels based on consumption rates, and randomizes local weather conditions.
- **Persistence:** Automatically saves updated states to MongoDB.

12. User Interface Overview

Dashboard: High-level metrics and charts.

Live Map: The core interface for tracking.

Fleet Manager: A CRUD interface for managing the vessel list.

Docking Logs: A tabular view of historical events.

13. Security Considerations

Password Hashing: All passwords are salted and hashed using Bcrypt.

CORS Policy: Strict Allow-Origin policies to prevent unauthorized domain access.

Environment Variables: Sensitive keys (Database URI, Secrets) are stored in .env files and never exposed in the client code.

14. Deployment

Frontend: Deployed on **Netlify** for global CDN distribution and continuous integration with GitHub.

Backend: Deployed on **Render** as a Web Service to support the persistent background simulation process.

Database: Hosted on **MongoDB Atlas** with secure IP whitelisting.

15. Testing

Manual Testing: Verified authentication flows, RBAC permission barriers, and correct CRUD operations.

Simulation Testing: Verified vessel movement accuracy and persistence over long durations (24+ hours).

API Testing: Endpoints verified using Postman for correct status codes (200, 401, 403, 500).

16. Troubleshooting

Common issues and fixes:

- **"CORS Error":** Ensure the backend cors middleware allows the frontend domain.
- **"Map Not Loading":** Verify internet connection for Leaflet tile servers.
- **"Login Failed":** Check MongoDB connection status in the backend logs.

17. Conclusion

The Maritime Fleet Manager successfully demonstrates the power of the MERN stack in building complex, event-driven applications. By combining real-time simulation with robust management tools, it solves the challenge of visualizing and controlling large-scale fleet operations remotely.