Full Stack Development with

MERN Project DocumentatiON

FORMAT

1. Introduction

- **Project Title: Traffic Volume Estimation**
- **Team Members: jammu kalyani**

2. Project Overview

- **Purpose**: The purpose of the Traffic Volume Estimation project is to accurately predict vehicle flow using machine learning techniques. It aims to support intelligent traffic management and urban planning through real-time data analysis.

- **Features:** The Traffic Volume Estimation project offers features like real-time vehicle count prediction using machine learning, support for video and sensor data input, and automated preprocessing with vehicle detection. It includes an interactive React-based dashboard for visualizing traffic trends, analyzing historical data, and aiding decision-making. The system is designed to be scalable and adaptable for use across multiple traffic locations or urban environments.

3. Setup Instructions

- **Prerequisites:** The prerequisites for the Traffic Volume Estimation project include a basic understanding of Python for implementing machine learning models and JavaScript with React for building the frontend interface. Familiarity with machine learning concepts, computer vision using libraries like OpenCV, and experience with data preprocessing and handling video or sensor inputs are essential. Knowledge of React.js, HTML, CSS, and REST APIs is required to develop and connect the user interface with the backend. Additionally, using development tools like Jupyter Notebook, VS Code, and Git will support efficient coding and project management.

4. Folder Structure

- **Client:** The React frontend is structured using a component-based architecture. It includes reusable components like `Navbar`, `Dashboard`, `UploadForm`, and `TrafficChart`. The `App.js` file manages routing between views using React Router. State is managed locally with hooks or globally via a state management tool like Redux or Context API. The frontend interacts with the backend using Axios or Fetch for API calls, and the UI is styled using CSS modules, Tailwind CSS, or styled-components for a clean and responsive design.

- **Server:** The Node.js backend is organized using Express.js to handle API routes such as `/upload`, `/predict`, and `/history`. It follows an MVC (Model-View-Controller) pattern for scalability. Middleware is used for tasks like authentication, file parsing (e.g., with `multer`), and error handling. ML models or scripts are either run via Python shell integration or pre-loaded services. The backend connects to a database (like MongoDB or PostgreSQL) to store and retrieve traffic data and logs, supporting secure and efficient data flow to the frontend.

5. Running the Application

- Provide commands to start the frontend and backend servers locally.
    - **Frontend:** `npm start` in the client directory.
    - This will launch the development server and open the React app in your default browser, where users can upload traffic data, view predictions, and analyze visualizations.

    - **Backend:** `npm start` in the server directory.
    - This command starts the Express server, handles API requests, communicates with the ML model, and serves data to the frontend through defined endpoints.

6. API Documentation

POST `/api/upload` – Uploads traffic video or image file.
Response: `{ status: "success", message: "File uploaded successfully" }`

- **POST `/api/predict`** – Processes uploaded data and returns estimated traffic volume.
  Response: `{ estimated_volume: 138, timestamp: "2025-06-28T14:30:00Z" }`

- **GET `/api/history`** – Fetches past prediction records.
  Response: `[{ filename: "...", estimated_volume: 120, date: "..." }]`

- **GET `/api/health`** – Checks if the server is running.
  Response: `{ status: "online" }`

7. Authentication

- In the **Traffic Volume Estimation** project, authentication and authorization are handled using **JWT (JSON Web Tokens)**. When a user logs in, the backend generates a token containing user credentials and permissions, which is sent to the frontend and stored in the browser (typically in localStorage or cookies). For each protected API request, the frontend includes this token in the request headers. The backend then verifies the token using a secret key to authenticate the user and checks their role or access level to authorize specific actions. This stateless token-based approach ensures secure and scalable access control without relying on server-side sessions.

8. Testing

- The testing strategy for the Traffic Volume Estimation project involves both automated and manual testing. The React frontend is tested using Jest and React Testing Library to validate component behavior and user interactions. The Node.js backend uses tools like Mocha or Jest to test API endpoints and middleware functions. The machine learning model is evaluated using performance metrics during training. Additionally, manual testing is performed using tools like Postman and browser-based checks to ensure real-time predictions, file uploads, and dashboard functionalities work correctly.

.

9. Known Issues

- The Traffic Volume Estimation project has a few known issues, including reduced accuracy in low-light or rainy conditions and delays when uploading large video files. The ML model may also perform poorly with videos captured at unusual angles or with low frame rates. Additionally, the system currently lacks role-based access control and has limited mobile responsiveness, which may affect usability on smaller screens.

10. Future Enhancements

- Future improvements for the Traffic Volume Estimation project could include integrating real-time traffic camera feeds for continuous monitoring, enhancing the ML model to handle diverse weather and lighting conditions, and adding role-based access control for different user types. Additional features may include mobile app support, predictive analytics for future traffic trends, multi-location traffic comparisons, and integration with city traffic systems or navigation apps for dynamic traffic management. Cloud storage and processing can also be added to improve scalability and performance.