**Project Specifications:**

**Functional Requirements:-**

**The system must support the following features based on the current implementation:**

1. **User Authentication**
   * **Users can register with a name, email, and password.**
   * **Users can log in with email and password.**
   * **Validation for email format (must include '@').**
   * **Display success/error messages via popups (e.g., "Login successful (dummy for now!)").**
   * **Redirect or block access to protected pages (e.g., dashboard) if not authenticated.**
2. **Navigation and Pages**
   * **Global navigation bar with links: Home, Dashboard, Login, Register.**
   * **Home Page: Display welcome message and project description.**
   * **Dashboard: Accessible only to logged-in users; currently shows access denial message if unauthenticated.**
3. **UI Elements**
   * **Forms with input fields, buttons (e.g., blue "Login", green "Register").**
   * **Emojis for visual feedback (e.g., for welcome, for not logged in).**
   * **Links between pages (e.g., "Register here" on login page).**
4. **Future Functional Extensions (Planned)**
   * **Integrate AI for traffic prediction using models like LSTM.**
   * **Fleet data simulation and visualization on dashboard (e.g., maps with markers).**
   * **Route optimization via graph algorithms.**

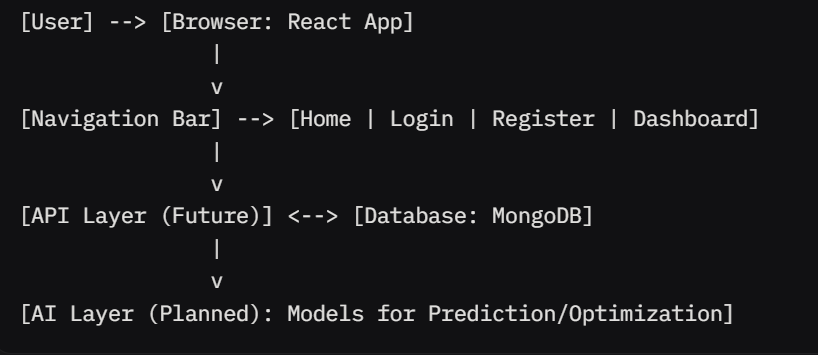
**Non-Functional Requirements:-**

* **Performance: Pages load in <2 seconds on local setup. Handle up to 10 concurrent users in prototype.**
* **Security: (Current: Basic client-side validation; Planned: Server-side auth with JWT, hashed passwords, HTTPS). No real data storage yet to avoid vulnerabilities.**
* **Usability: Intuitive UI with clear error messages, responsive design (mobile-friendly planned).**
* **Scalability: Designed for easy backend integration; use modular components in React.**
* **Accessibility: Basic (e.g., alt text for emojis planned); comply with WCAG 2.1 Level A.**
* **Reliability: Error handling for invalid inputs; dummy successes ensure flow continuity.**
* **Compatibility: Works on modern browsers (Chrome, Edge, as shown in screenshots).**

**System Architecture:-**

* **High-Level Design:Client-server model (frontend React app communicating with future backend API).**
  + **Frontend: React components for pages (e.g., Login.js, Register.js, Dashboard.js). State management via React hooks (e.g., useState for form data).**
  + **Backend: (Placeholder) Future: RESTful API for /login, /register, /fleet-data. Database: MongoDB for users and fleet data.**
  + **Data Flow: User interacts with UI → Form submission → API call (dummy now) → Response popup → Redirect if success.**

**Components Diagram (Text-Based):**



**User Roles and Permissions:-**

* **Guest (Unauthenticated): Access Home, Login, Register. Blocked from Dashboard.**
* **Registered User: Access all pages, including Dashboard. (Current: No role differentiation; all users equal).**
* **Admin (Planned): Manage fleet data, view analytics.**

**APIs and Endpoints:-**

* **Current (Dummy): Handled client-side.**
* **Planned REST Endpoints:**
  + **POST /register: Create user.**
  + **POST /login: Authenticate and return token.**
  + **GET /dashboard: Fetch user-specific data (auth required).**
  + **GET /fleet-data: Retrieve simulated fleet info.**

**Test Cases:**

**Unit Tests:-**

**Focus on individual components/functions. Assume Jest for React testing.**

1. **Email Validation Function**
   * **Test Case 1: Valid email ("**[**anuragtiwari3005@gmail.com**](mailto:anuragtiwari3005@gmail.com)**") → Expect no error.**
   * **Test Case 2: Invalid email ("anuragtiwari3005ail.com") → Expect error: "missing '@'".**
   * **Test Case 3: Empty email → Expect error: "Email required".**
2. **Form Submission (Login)**
   * **Test Case 1: Valid credentials (dummy) → Expect success popup.**
   * **Test Case 2: Invalid password → Expect error message (future backend).**
3. **Navigation Link Rendering**
   * **Test Case 1: Render navbar → Expect 4 links present.**

**Integration Tests:-**

**Test interactions between components (e.g., using React Testing Library).**

1. **Login Flow**
   * **Test Case 1: Enter valid email/password → Submit → See success popup → Redirect to Dashboard.**
   * **Test Case 2: Enter invalid email → Submit → See validation error, no submission.**
2. **Registration to Login**
   * **Test Case 1: Register new user → Success popup → Click "Login here" → Navigate to login page.**
3. **Dashboard Access**
   * **Test Case 1: Unauthenticated access → See "You are not logged in" message.**
   * **Test Case 2: After login → See dashboard content (placeholder).**

**UI/Functional Tests:-**

**Manual or automated (e.g., Cypress for end-to-end).**

1. **Page Loading**
   * **Test Case 1: Load Home → Verify welcome text and emoji.**
   * **Test Case 2: Load Login → Verify form fields pre-filled or empty.**
2. **Popup Interactions**
   * **Test Case 1: Successful login → Click "OK" on popup → Dismiss popup.**
3. **Cross-Page Navigation**
   * **Test Case 1: From Login → Click "Register here" → Load Register page.**

**Edge Cases and Error Handling:**

1. **Invalid Inputs**
   * **Test Case 1: Password <6 chars on register → Expect client-side error (add if not present).**
   * **Test Case 2: Duplicate email on register → Expect backend error (future).**
2. **Network/Backend Failures**
   * **Test Case 1: Simulate API failure → Show generic error popup.**
3. **Browser Compatibility**
   * **Test Case 1: Test on Chrome/Edge → No layout breaks.**
   * **Test Case 2: Resize window → Responsive (if implemented).**

**Test Coverage Goal: 80% for current features. Use tools like Jest for automation.**

**Assumptions and Dependencies:**

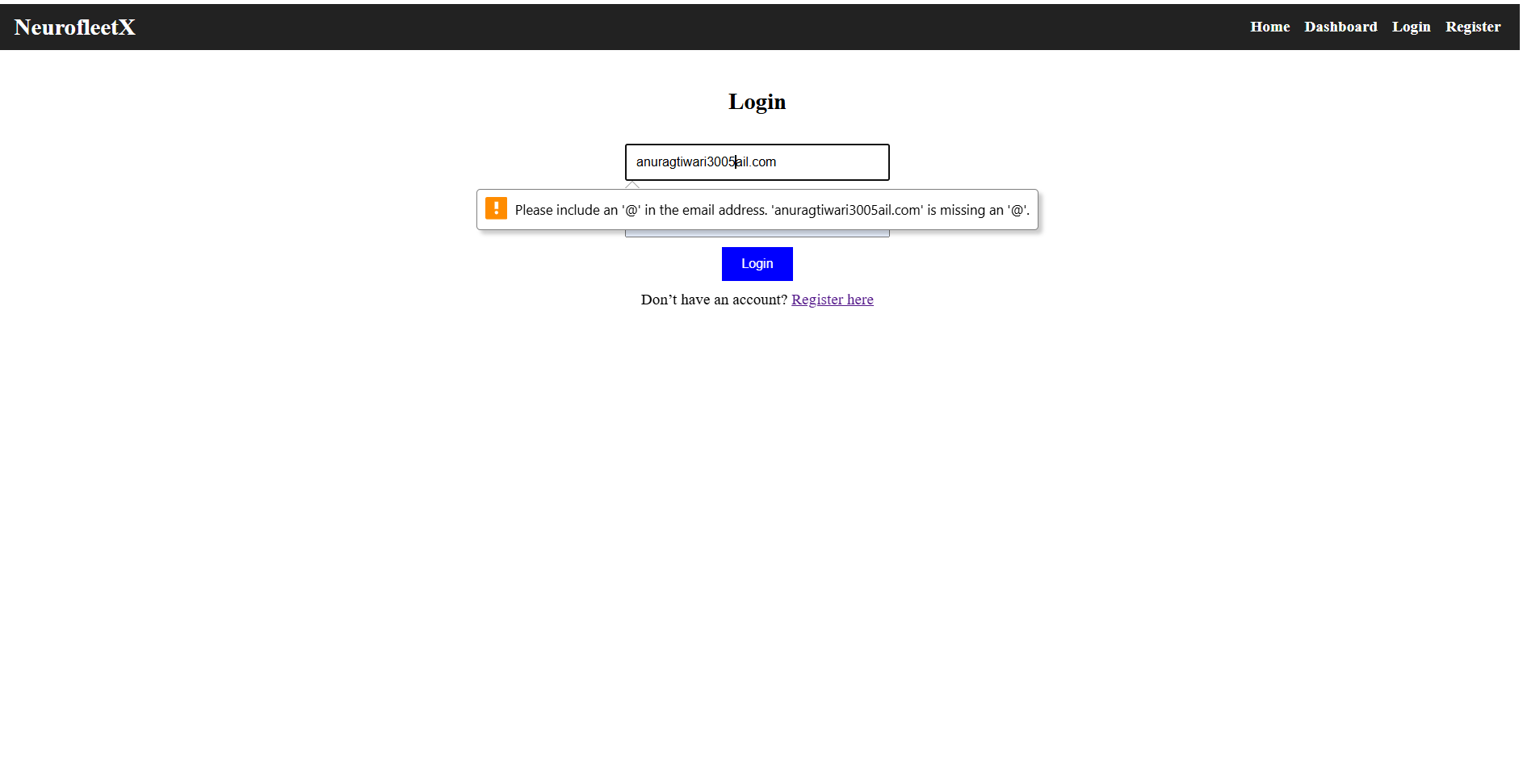
* **Assumptions: Users have modern browsers; no real auth backend yet (dummy only).**
* **Dependencies: React 18+, Node.js for build; MongoDB for future data. No external APIs integrated yet.**

**Future Enhancements:**

* **Full backend with auth and database.**
* **AI integration (e.g., LSTM for traffic).**
* **Map visualization on dashboard.**
* **Mobile app companion**

**Appendix:-**

* **Screenshots:-**



* **Code Snippets (Example for Login Component):-**

**// Login.js (pseudocode)**

**import React, { useState } from 'react';**

**const Login = () => {**

**const [email, setEmail] = useState('');**

**const [password, setPassword] = useState('');**

**const [error, setError] = useState('');**

**const handleSubmit = (e) => {**

**e.preventDefault();**

**if (!email.includes('@')) {**

**setError("Missing '@' in email");**

**} else {**

**// Dummy success**

**alert("Login successful (dummy for now!)");**

**}**

**};**

**return (**

**<form onSubmit={handleSubmit}>**

**<input value={email} onChange={(e) => setEmail(e.target.value)} />**

**<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} />**

**{error && <p>{error}</p>}**

**<button type="submit">Login</button>**

**</form>**

**);**

**};**