

1. Features to Implement

- a. - User Authentication: Users should be able to register and log in using email/password. Implement authentication using JWT or Passport.js.
- b. - Route Planning and Display: Users should be able to enter their start and destination locations. Integrate Google Maps API (or OpenStreetMap, Mapbox) to display the route.
- c. - Live Traffic Updates (Optional): Display live traffic data for better route optimization.
- d. - Commute History Management: Store user commute history in MongoDB. Allow users to view past routes taken.
- e. - Fare Estimation (Optional): Estimate travel cost based on distance and transportation mode.
- f. - User Dashboard: Display commute statistics (e.g., total distance traveled, average commute time).

2. Technical Requirements

- a. Backend (Node.js & Express.js): RESTful API endpoints for user authentication, route planning, and history management.
- b. Frontend (Optional): Use HTML, CSS, JavaScript, and frontend frameworks (React.js/Vue.js) if implementing a UI.
- c. Google Maps API Integration: Use Google Maps API to fetch and display routes and locations.

- d. Deployment: Deploy the application using Render/Heroku/Vercel for demonstration.

3. Project Setup Instruction.

- a. Clone the Repository
 - i. bash
 - ii. git clone https://github.com/Rajatt-Chawla/daily_commute_portal.git
 - iii. cd daily_commute_portal
- b. Install Dependencies
- c. Backend (Node.js, Python, Java, etc. – depending on your tech stack)
 - i. Bas
 - ii. # Example for Node.js:
 - 1. cd backend
 - 2. npm install
- d. Frontend (React, Angular, Vue, etc. – as required)
 - i. bash
 - ii. # Example for React:
 - 1. cd ../frontend
 - 2. npm install
- e. Set Environment Variables
- f. Create a file like .env (or .env.local) in both frontend and backend (if required) to store any secret keys or environment-specific variables.
 - i. Example:
 - ii. env
 - 1. REACT_APP_API_URL=http://localhost:5000/api

2. DB_CONNECTION_STRING=your_database_connection_string
- g. Database Setup
 - h. Run the Application
 - i. Backend:
 1. bash
 2. npm run start
 3. or
 4. Bash
 - ii. npm run dev
 - iii. Frontend:
 1. bash
 2. npm start
 - iv. The frontend should now be accessible in your browser at something like http://localhost:3000 (depending on your settings).
 - v. Build for Production (Optional)
 1. For the frontend, you might run:
 2. bash
 3. npm run build
 4. Serve the build directory through your chosen server or cloud hosting.

4. API Endpoints and Functionalities

- a. Below is a quick overview of key API endpoints. (Adjust based on your actual routes and controllers.)
- b. User Authentication
 - i. POST /api/auth/register
 - ii. Description: Creates a new user account.
 - iii. Request Body: { username, email, password }
 - iv. Response: Success or error message.

- v. POST /api/auth/login
 - vi. Description: Authenticates user.
 - vii. Request Body: { email, password }
 - viii. Response: Auth token or error.
- c. Commute Scheduling
 - i. GET /api/commutes
 - ii. Description: Retrieves a list of available commutes/routes.
 - iii. Query Params: Optional filters (e.g., origin, destination).
 - iv. Response: List of commute objects.
 - v. POST /api/commutes
 - vi. Description: Creates a new commute listing (e.g., a carpool schedule).
 - vii. Request Body: Commute details (e.g., { origin, destination, time, seatsAvailable }).
 - viii. Response: Newly created commute object or error message.
 - d. Booking / Joining Commutes
 - i. POST /api/bookings
 - ii. Description: Book a seat in a listed commute.
 - iii. Request Body: { commuteId, userId }.
 - iv. Response: Booking confirmation or error.

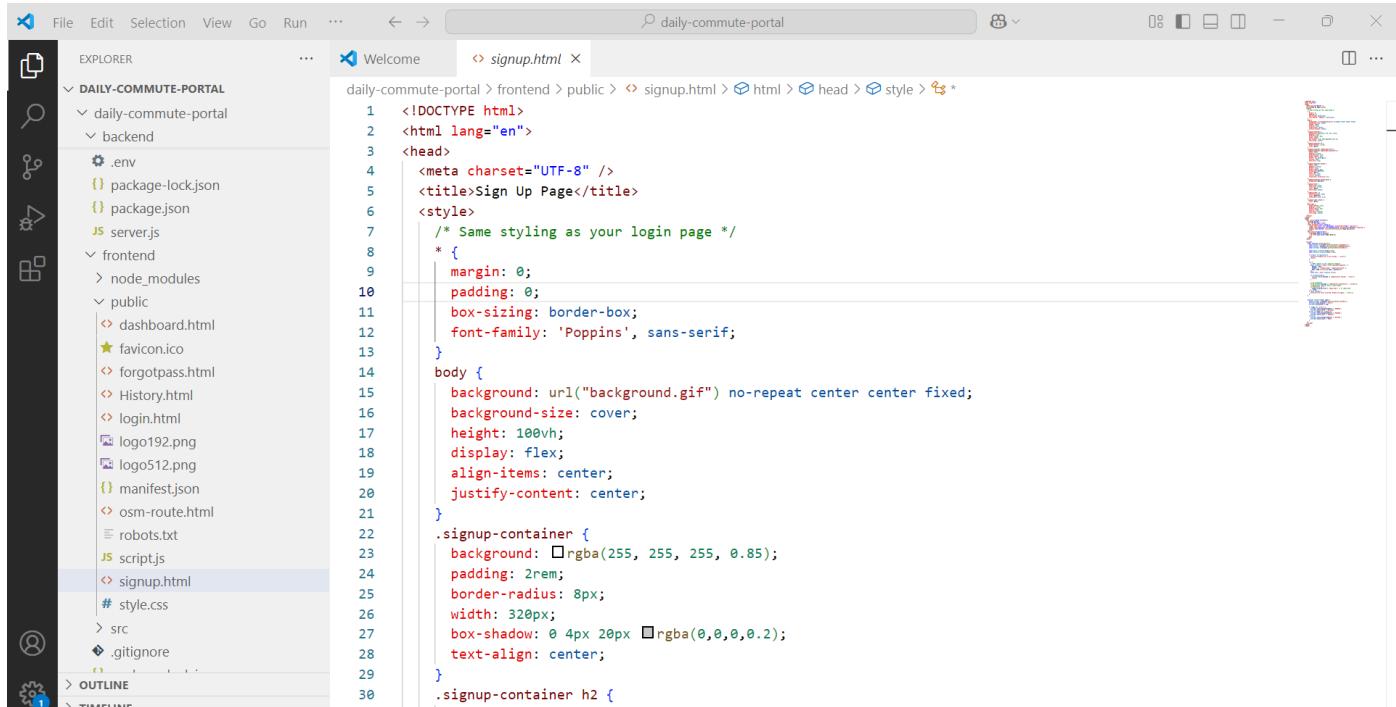
5. Challenges Faced and Solutions Applied

- a. Large File / Media Handling

- i. Challenge: Needed to store or serve large media files (like a background image) which exceeded GitHub's size limit.
 - ii. Solution: Used Git LFS for large files or compressed the file size to stay within limits. Alternatively, used external storage (CDN / cloud storage).
- b. Database Integration
- i. Challenge: Ensuring real-time or fast data retrieval for route listings.
 - ii. Solution: Indexed frequently queried fields (e.g., origin, destination) in the database and implemented caching where necessary.
- c. Authentication & Security
- i. Challenge: Protecting endpoints from unauthorized access.
 - ii. Solution: Implemented JWT-based authentication (or session-based) and middleware checks for role-based access control if needed.
- d. Handling Concurrent Bookings
- i. Challenge: Avoiding seat overbooking when multiple users simultaneously try to book.
 - ii. Solution: Implemented transaction-based approach or used database-level locks/optimistic concurrency checks to ensure seat count updates were atomic.
- e. Performance Optimization
- i. Challenge: Large data sets or multiple concurrent users slowing down the app.

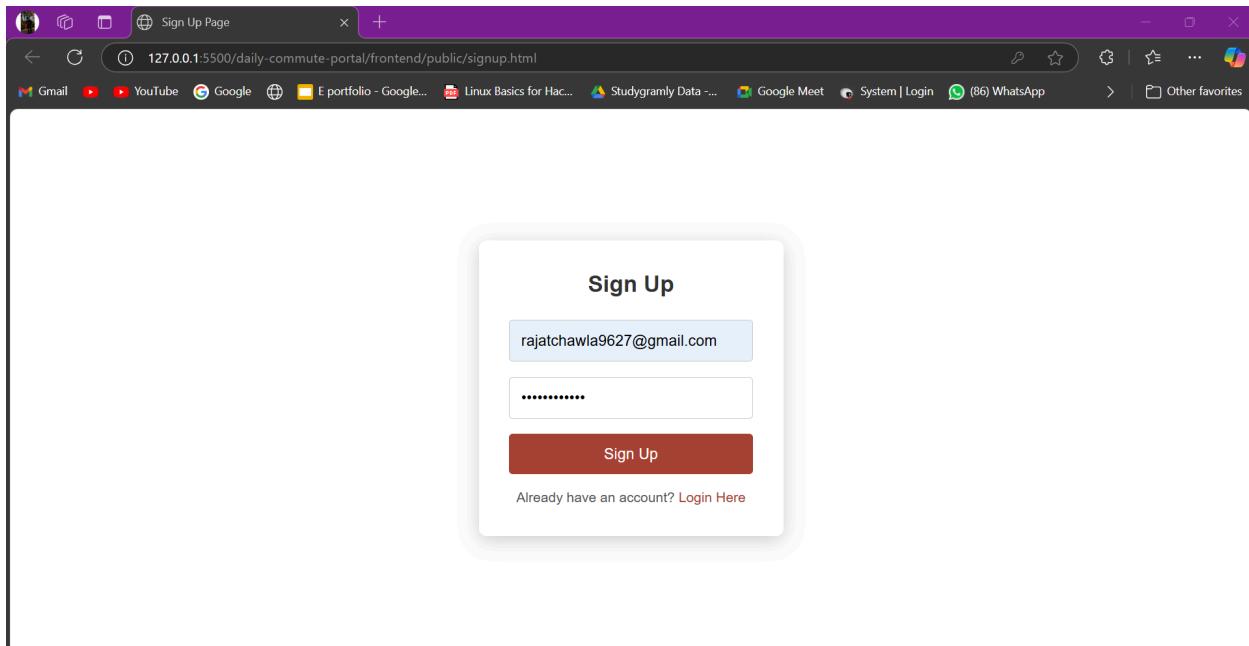
- ii. Solution: Added server-side pagination, used efficient queries, and introduced caching (e.g., Redis) for frequently accessed data.

6. Sign Up page

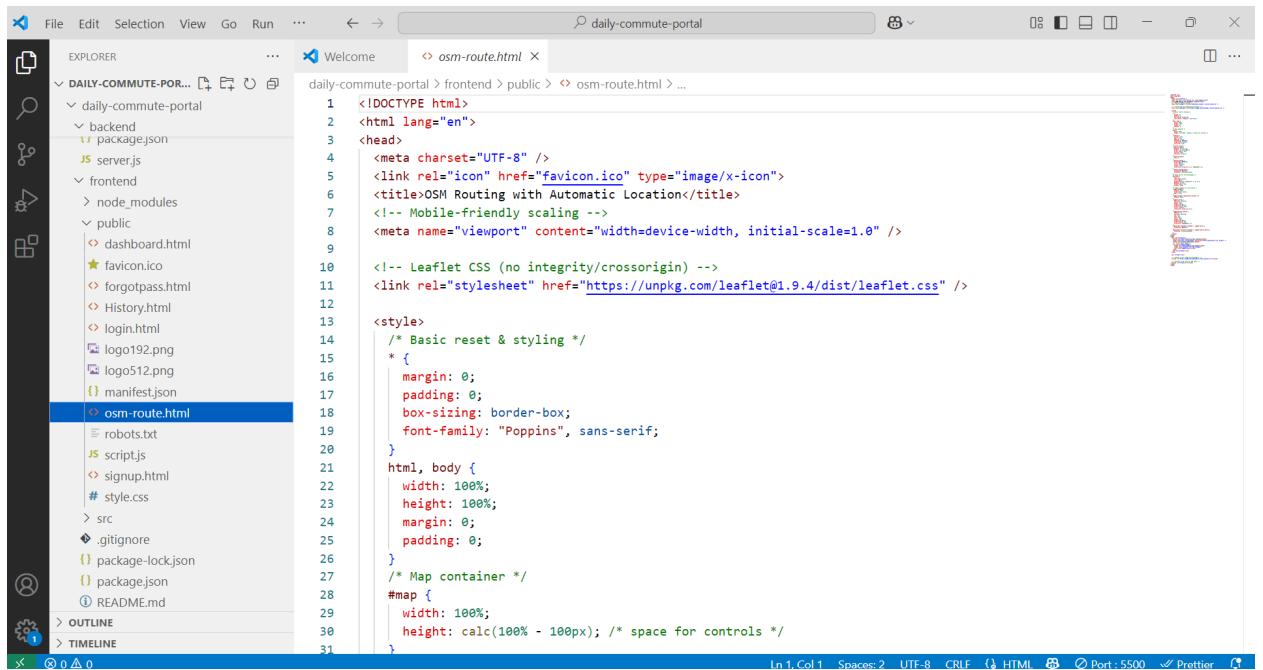


The screenshot shows a code editor interface with the following details:

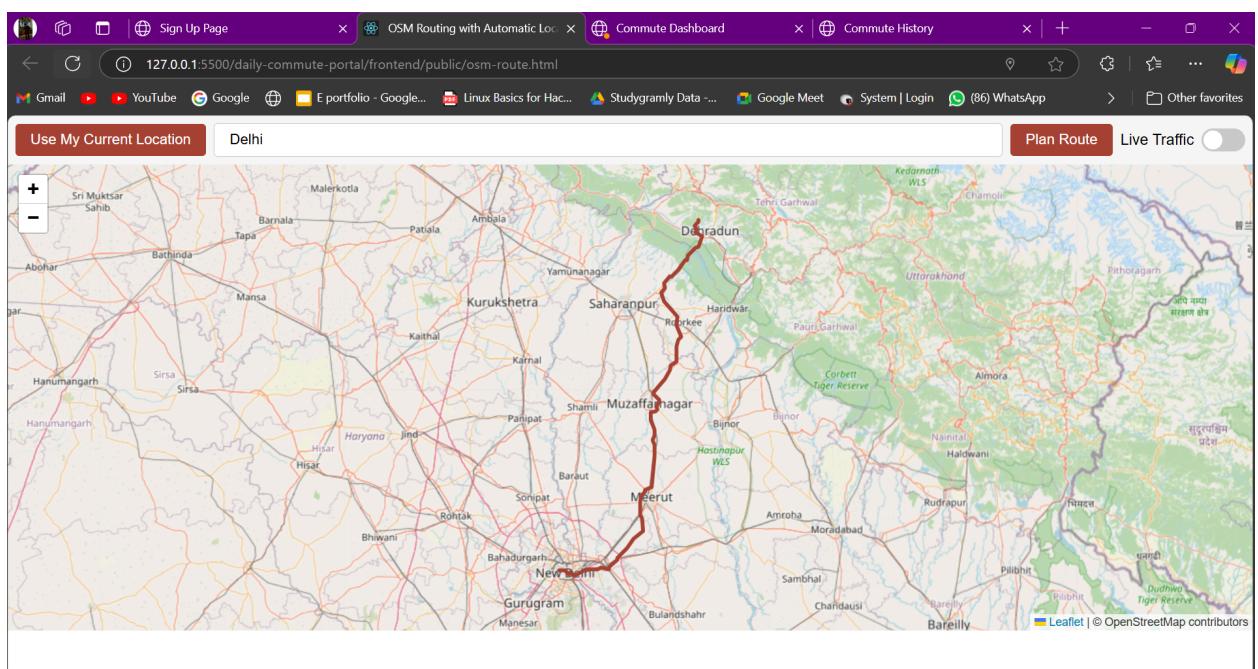
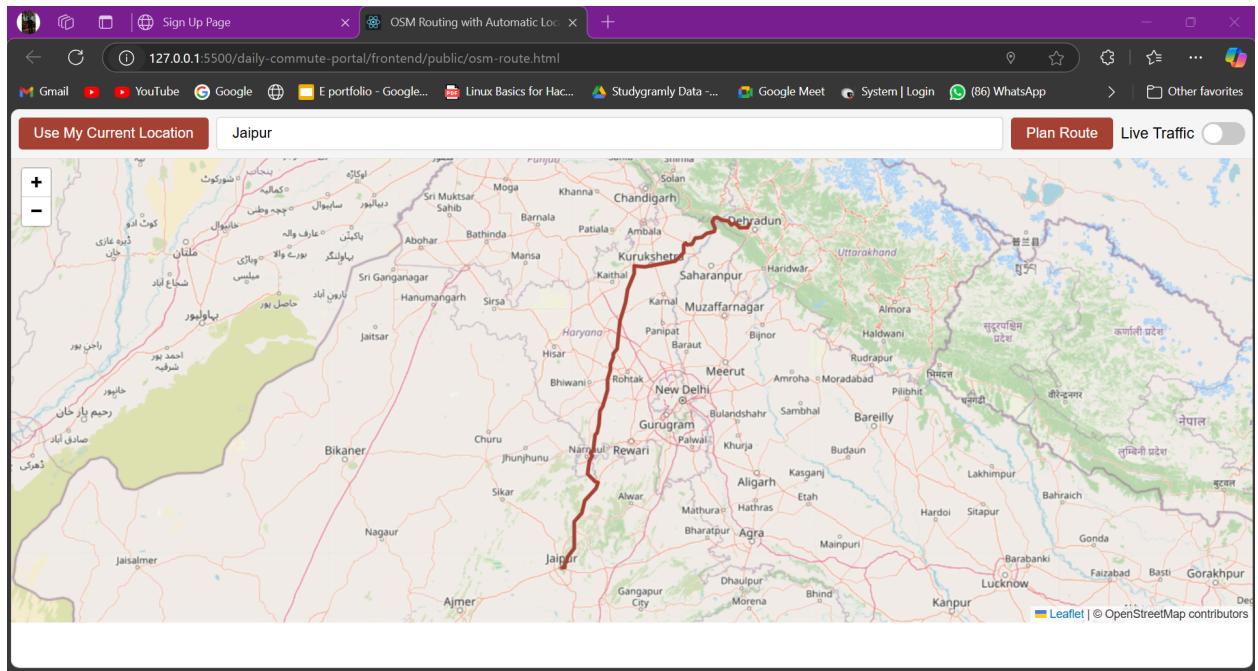
- File Path:** daily-commute-portal > frontend > public > signup.html
- Editor View:** The code editor displays the content of the 'signup.html' file. The file contains an HTML document structure with a title 'Sign Up Page' and a body section styled with CSS. The CSS includes rules for the 'body' and '.signup-container' classes, setting backgrounds, padding, and shadows.
- SIDE BAR:** The left sidebar shows the project structure under 'DAILY-COMMUTE-PORTAL'. It includes folders for 'backend', 'frontend', and 'public', along with files like '.env', 'package-lock.json', 'package.json', 'server.js', 'dashboard.html', 'favicon.ico', 'forgotpass.html', 'History.html', 'login.html', 'logo192.png', 'logo512.png', 'manifest.json', 'osm-route.html', 'robots.txt', 'script.js', 'style.css', and '.gitignore'.
- STATUS BAR:** The bottom status bar shows the file name 'signup.html' and its line count (30).



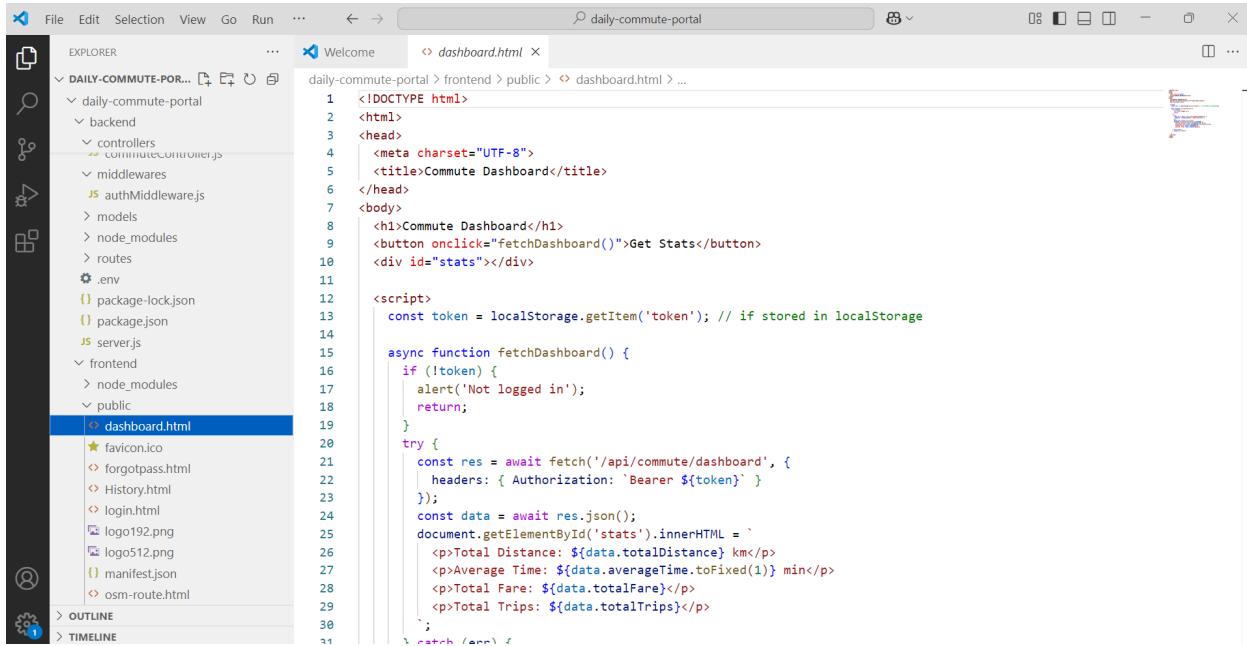
7. Route Planning

A screenshot of a Visual Studio Code (VS Code) interface. The title bar says "osm-route.html". The left sidebar shows a file tree with a project structure: "DAILY-COMMUTE-PORTAL", "backend", "frontend", "public", and files like "osm-route.html", "style.css", and "script.js". The main editor area displays the content of "osm-route.html". The code is an HTML file with some CSS styles. The status bar at the bottom shows "In 1, Col 1" and "Port: 5500".

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<link rel="icon" href="favicon.ico" type="image/x-icon">
<title>OSM Routing with Automatic Location</title>
<!-- Mobile-friendly scaling -->
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<!-- Leaflet CSS (no integrity/crossorigin) -->
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
<style>
/* Basic reset & styling */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: "Poppins", sans-serif;
}
html, body {
    width: 100%;
    height: 100%;
    margin: 0;
    padding: 0;
}
/* Map container */
#map {
    width: 100%;
    height: calc(100% - 100px); /* space for controls */
}
```



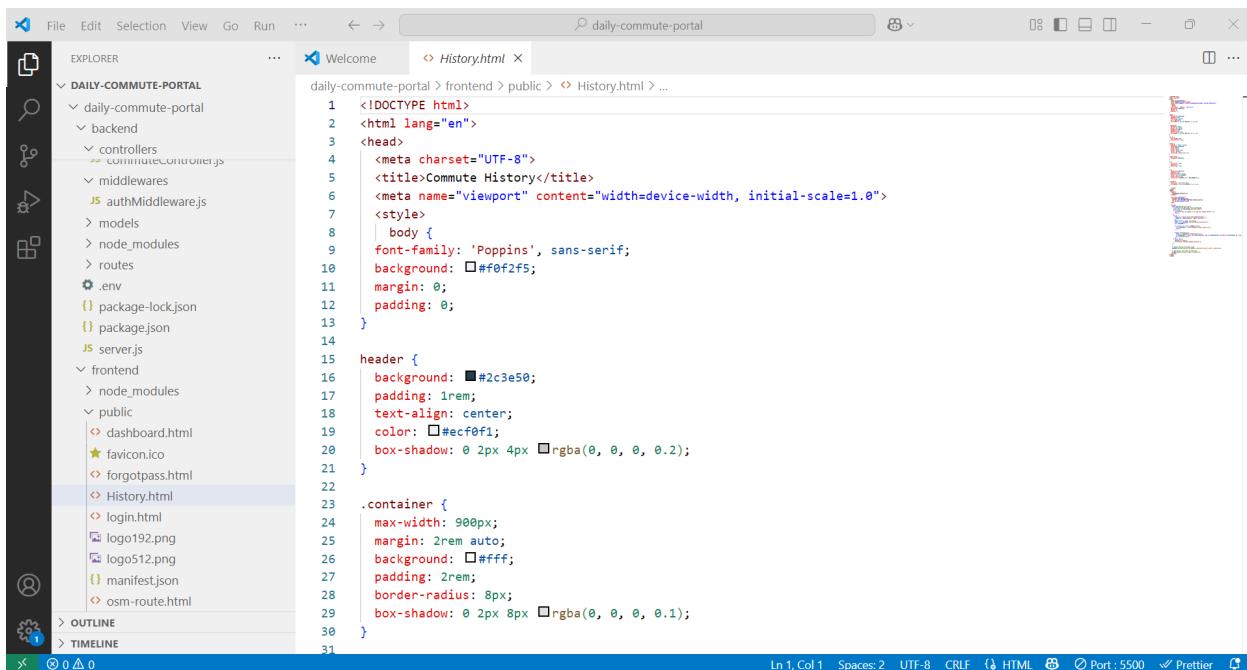
8. Commute History Dashboard



The screenshot shows a code editor interface with the file `dashboard.html` open. The left sidebar displays the project structure for a "DAILY-COMMUTE-PORTAL" application, including a "backend" folder with "controllers" and "middlewares" subfolders, and a "frontend" folder with "public" and "src" subfolders. The "src/public" folder contains files like `favicon.ico`, `forgotpass.html`, `History.html`, `login.html`, `logo192.png`, `logo512.png`, `manifest.json`, and `osm-route.html`. The "src/frontend/src/public" folder contains the `dashboard.html` file, which is selected in the sidebar. The main editor area shows the following HTML and JavaScript code:

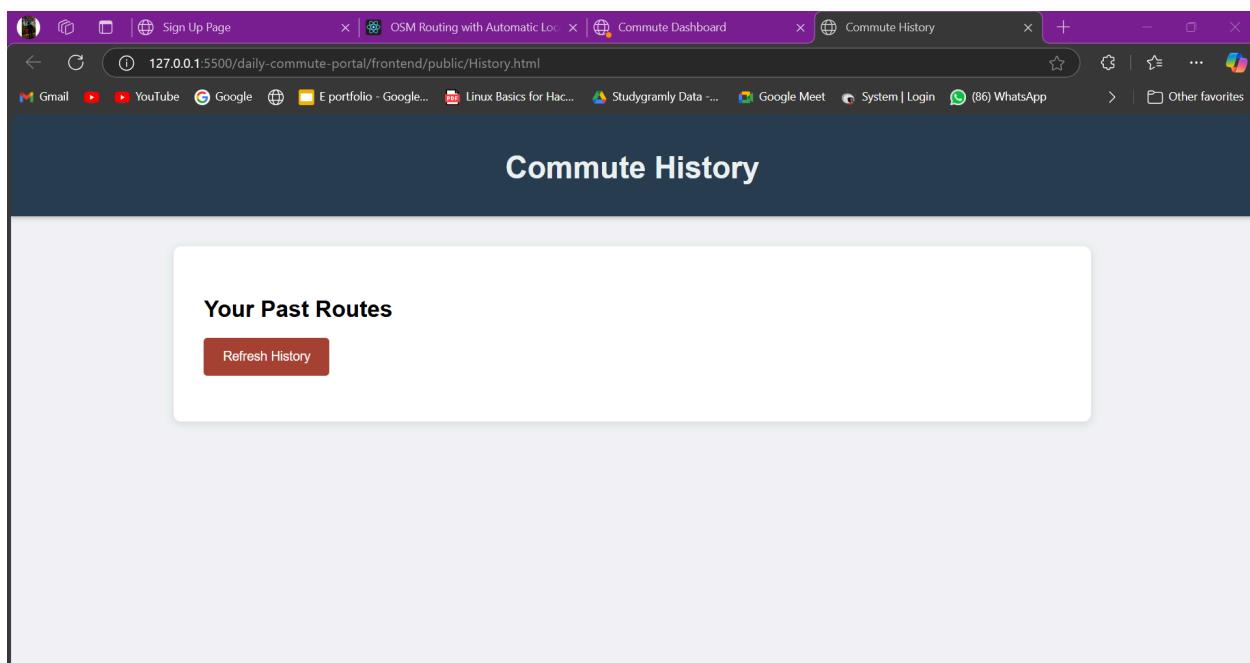
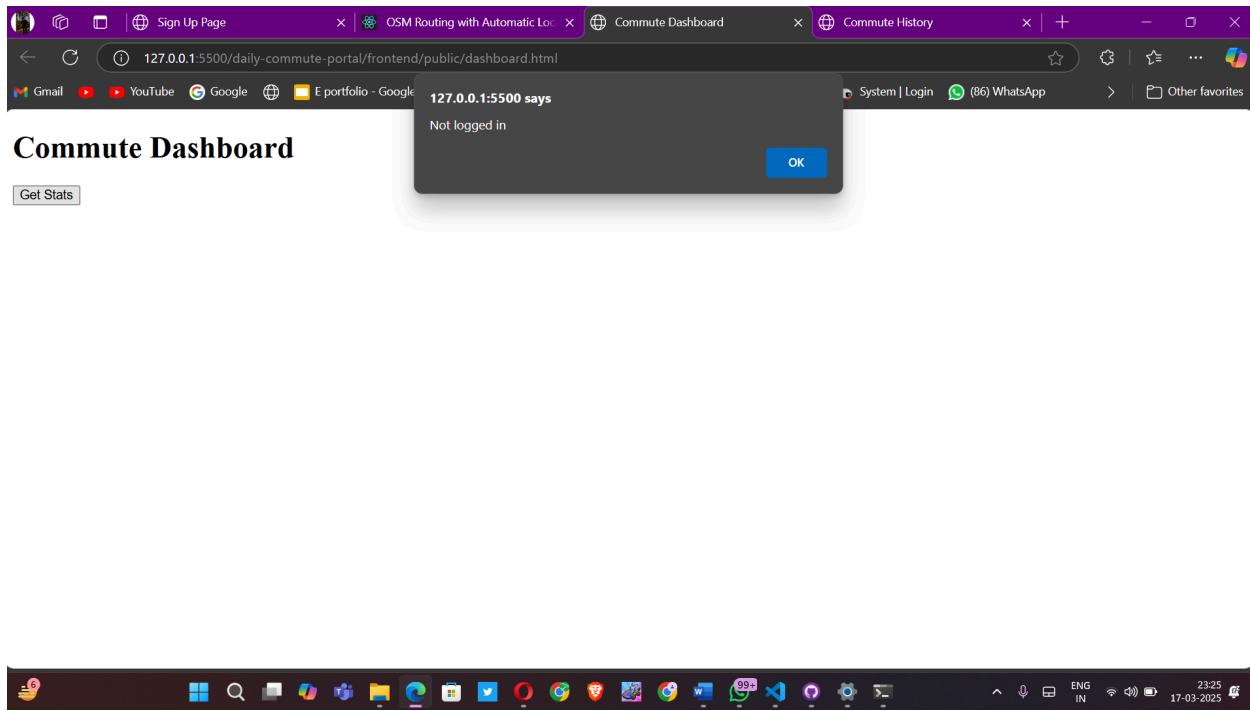
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Commute Dashboard</title>
</head>
<body>
<h1>Commute Dashboard</h1>
<button onclick="fetchDashboard()">Get Stats</button>
<div id="stats"></div>
<script>
const token = localStorage.getItem('token'); // if stored in localStorage

async function fetchDashboard() {
  if (!token) {
    alert('Not logged in');
    return;
  }
  try {
    const res = await fetch('/api/commute/dashboard', {
      headers: { Authorization: `Bearer ${token}` }
    });
    const data = await res.json();
    document.getElementById('stats').innerHTML = `
      <p>Total Distance: ${data.totalDistance} km</p>
      <p>Average Time: ${data.averageTime.toFixed(1)} min</p>
      <p>Total Fare: ${data.totalFare}</p>
      <p>Total Trips: ${data.totalTrips}</p>
    `;
  } catch (err) {
    console.error(err);
  }
}
</script>
```



The screenshot shows a code editor interface with the file `History.html` open. The left sidebar displays the project structure for a "DAILY-COMMUTE-PORTAL" application, identical to the previous screenshot. The "src/public" folder contains files like `favicon.ico`, `forgotpass.html`, `History.html`, `login.html`, `logo192.png`, `logo512.png`, `manifest.json`, and `osm-route.html`. The "src/frontend/src/public" folder contains the `History.html` file, which is selected in the sidebar. The main editor area shows the following HTML and CSS code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Commute History</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
  body {
    font-family: 'Poppins', sans-serif;
    background: #f0f2f5;
    margin: 0;
    padding: 0;
  }
  header {
    background: #2c3e50;
    padding: 1rem;
    text-align: center;
    color: #ecf0f1;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
  }
  .container {
    max-width: 900px;
    margin: 2rem auto;
    background: #fff;
    padding: 2rem;
    border-radius: 8px;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
  }
</style>
```



9. User Dashboard

The screenshot shows a Windows desktop environment with a code editor and a web browser.

Code Editor: The top half of the screen displays a code editor window for the file `dashboard.html`. The code is an HTML page with some JavaScript. It includes a title "Commute Dashboard", a button to "Get Stats", and a `fetchDashboard()` function that checks for a token in localStorage and makes a fetch request to the API if no token is found.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Commute Dashboard</title>
</head>
<body>
<h1>Commute Dashboard</h1>
<button onclick="fetchDashboard()">Get Stats</button>
<div id="stats"></div>
<script>
const token = localStorage.getItem('token'); // if stored in localStorage

async function fetchDashboard() {
  if (!token) {
    alert('Not logged in');
    return;
  }
  try {
    const res = await fetch('/api/commute/dashboard', {
      headers: { Authorization: `Bearer ${token}` }
    });
    const data = await res.json();
    document.getElementById('stats').innerHTML =
      `<p>Total Distance: ${data.totalDistance} km</p>
      <p>Average Time: ${data.averageTime.toFixed(1)} min</p>
      <p>Total Fare: ${data.totalFare}</p>
      <p>Total Trips: ${data.totalTrips}</p>
    `;
  } catch (err) {
}
```

Browser: Below the code editor is a browser window showing the URL `127.0.0.1:5500/daily-commute-portal/frontend/public/dashboard.html`. A modal dialog box is displayed, stating "Not logged in" with an "OK" button. The browser's address bar also shows other tabs like "Sign Up Page", "OSM Routing with Automatic Loc", and "Commute History".

Taskbar: At the bottom of the screen is the Windows taskbar, showing various pinned icons and the date/time "17-03-2025 23:25".