BCSE302P – Database Systems Lab

Case study for the Project

Smart City Traffic Management System

22BCB0041 HIMANSHU UPADHYAY

23BCE0048 MANYA KOTHARI

23BCE0106 SHRISH SINGH SOURYA

23BCE0182 DHRITI SAXENA

School of Computer Science and Engineering

(SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

TABLE OF CONTENTS

Abstract

Modern urban environments generate a vast and continuous stream of heterogeneous data from transportation networks. The effective management of this data is paramount to solving complex challenges like traffic congestion and public safety. This case study presents the comprehensive database design and implementation for a Smart City Traffic Management System, built on a modern technology stack including Node.js, Express, React, and MySQL. It details the systematic process of database development, beginning with a conceptual model that is translated into a robust, normalized relational schema. Key aspects of the design, including the specification of primary and foreign keys to ensure data integrity, are thoroughly examined. The study then delves into the practical application of this database, detailing the system architecture, specific SQL queries, and real-time data synchronization workflows using Socket.io that support the system's core features, from live monitoring dashboards to incident management and user authentication.

1. INTRODUCTION

1.1. The Urban Traffic Challenge as a Data Management Problem

The core challenge of modern urban traffic management is fundamentally a data management problem. Cities are complex systems that generate immense volumes of high-velocity, heterogeneous data from a multitude of sources: vehicle movements, sensor readings, video feeds, incident reports, and pedestrian activity. Traditional traffic management systems, with their static, pre-programmed logic, are incapable of effectively capturing, storing, processing, and querying this data in real-time. The consequences are significant: chronic congestion, increased pollution, compromised public safety, and substantial economic losses.

Solving this requires a paradigm shift towards a data-centric approach. An Intelligent Transportation System (ITS) treats the urban transportation network as a dynamic, data-producing ecosystem. The system's ability to make intelligent, real-time decisions is entirely dependent on the quality, availability, and integrity of its underlying data. Therefore, the design of a robust, scalable, and efficient database system is not merely a technical implementation detail; it is the foundational prerequisite for the success of the entire project. This case study focuses on the design of such a database, which must serve as the single source of truth for all operational and analytical functions of a Smart City Traffic Management System.

1.2. Project Goals: Designing a Robust Database Backend

The primary goal of this project is to design and detail the complete database backend for a Smart City Traffic Management System. This involves a systematic progression from conceptual modeling to a logical relational schema, ensuring the final design meets the stringent demands of a real-time, mission-critical application.

The specific objectives of this database design are:

- To accurately model the traffic management domain: Identify all relevant entities, their attributes, and the relationships between them.
- To ensure data integrity and consistency: Implement a schema with strong integrity constraints (primary, foreign, and candidate keys) to prevent data anomalies and ensure the data is reliable.
- To achieve high performance and scalability: Design a structure that supports fast data ingestion and low-latency querying to meet real-time operational needs.
- To support diverse application needs: Create a database that can serve as the backend for real-time operational transactions (OLTP) and strategic analysis (OLAP).

1.3. Scope: Defining the Boundaries of the Database System

The scope of this database design is to create the data model and schema for the core functionalities of the traffic management system. This includes the data structures necessary to support:

- User authentication and role-based authorization.
- Configuration and real-time status of traffic signals.
- Reporting, tracking, and lifecycle management of traffic incidents.
- Storage of historical congestion metrics for analytics.

The scope focuses on the database design and its direct interaction with the application backend, providing the essential blueprint and operational query examples.

2. SYSTEM ARCHITECTURE AND TECHNOLOGY STACK

The system is built on a modern, decoupled architecture designed for real-time responsiveness and scalability.

2.1. Technology Stack

- Backend: Node.js with the Express framework, providing a fast and efficient REST API.
- Database: MySQL, a robust and widely-used relational database management system (RDBMS).
- Frontend: React with TypeScript, built using Vite for a modern and performant user interface.
- Real-time Communication: Socket.io for enabling bidirectional, real-time communication between the server and clients.
- Mapping: Leaflet, an open-source JavaScript library for interactive maps.
- Authentication: JSON Web Tokens (JWT) for secure, stateless

authentication.

2.2. Architectural Flow

The system follows a classic client-server model, enhanced with a real-time communication layer. The MySQL database serves as the single source of truth for all application data.

- Standard Interaction (HTTP): The React frontend communicates with the Express backend via a REST API for standard data operations like fetching initial data, submitting forms, and user authentication.
- Real-time Interaction (WebSockets): For live updates, the server uses Socket.io to push data to all connected clients immediately after a change occurs in the database (e.g., a new incident is reported or a signal's status is changed). This avoids the need for clients to constantly poll the server for new information.

Frontend (React) ↔ REST API (Express) ↔ MySQL Database
↕
Socket.io (Real-time)

3. DATABASE DESIGN: FROM CONCEPT TO SCHEMA

The database design process began with a conceptual model to identify the core entities and relationships, which was then translated into a logical relational schema for implementation in MySQL.

3.1. Conceptual Model and Entities

The system revolves around four primary entities:

- Users: Represents individuals who interact with the system, with distinct roles (admin, authority, user) that govern their permissions.
- Signals: Represents physical traffic signals, storing their location, configuration, and current operational mode.
- Incidents: Represents traffic-related events reported by users, including their type, location, severity, and status in a resolution workflow.
- Congestion_Metrics: A table designed to store historical data about traffic congestion levels at various road segments for analytical purposes.

The key relationship in this model is between Users and Incidents. An incident is always reported by a user, creating a one-to-many relationship where one user can report multiple incidents. This is enforced via a foreign key.

3.2. Logical Relational Schema

The conceptual model is translated into the following physical schema in MySQL. This schema is designed to be in at least Third Normal Form (3NF) to reduce data redundancy and ensure data integrity.

1. Users Table Used for authentication, authorization, and linking incidents to

the users who reported them.

Column	Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the user.
name	VARCHAR		User's full name.
email	VARCHAR	UNIQUE	User's email, used for login.
password_hash	VARCHAR		Hashed password (using bcrypt).
role	ENUM	'admin', 'authority', 'user'	User's role for access control.
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of user creation.

2. Signals Table Stores the location, configuration, and current state of all traffic signals.

Column	Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the signal.
name	VARCHAR		A human-readable name for the signal.
lat	DECIMAL(9,6)		Latitude of the signal's location.
lng	DECIMAL(9,6)		Longitude of the signal's location.
cycle_seconds	INT		Total cycle time in seconds.
green_seconds	INT		Duration of the green light.
yellow_seconds	INT		Duration of the yellow light.
red_seconds	INT		Duration of the red light.

mode	ENUM	'normal', 'flashing', 'emergency'	Current operational mode.
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of signal creation.

3. Incidents Table The central table for tracking all reported traffic incidents.

Column	Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the incident.
reporter_id	INT	FOREIGN KEY -> users(id)	The ID of the user who reported it.
type	VARCHAR		Type of incident (e.g., 'accident').
description	TEXT		A detailed description of the incident.
lat	DECIMAL(9,6)		Latitude of the incident's location.
lng	DECIMAL(9,6)		Longitude of the incident's location.
severity	INT		Severity rating from 1 to 5.
status	ENUM	'reported', 'verified', 'resolved'	Current stage in the workflow.
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of incident report.

4. Congestion_Metrics Table Designed for storing analytical data, though not fully implemented in the current workflows.

Column	Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the metric.
segment	VARCHAR		Name or ID of the road segment.
congestion_level	INT		A numerical value for congestion.

lat	DECIMAL(9,6)		Latitude of the segment.
lng	DECIMAL(9,6)		Longitude of the segment.
observed_at	TIMESTAMP	INDEXED	Timestamp of the observation.

4. DESIGN APPROACH AND DETAILS

The database is the core of all system operations. This section details how each major component of the application interacts with the MySQL database through specific queries and real-time events.

4.1. Authentication and Authorization

The Users table is central to securing the system.

- Registration: A new user is created by hashing their password and executing an INSERT query. The email is checked for uniqueness first to enforce the UNIQUE constraint.

SQL

```
INSERT INTO users (name, email, password_hash, role) VALUES (?,?,?, 'user');
```

- Login: The system retrieves a user's record by their email. The provided password is then compared against the stored password_hash using bcrypt.

SQL

```
SELECT id, name, email, password_hash, role FROM users WHERE email =?;
```

- Authorization: After login, a JWT containing the user.id and role is issued. This token is used to authorize subsequent API requests. For protected actions, the backend middleware checks the user's role before proceeding. The user.id from the token is used in queries that require user context, such as reporting an incident.

4.2. Real-Time Dashboard

The dashboard provides a live map view of all signals and incidents.

- Initial Data Load: When the dashboard loads, it makes two REST API calls to fetch the current state from the database.

SQL

```
-- Fetch recent incidents
SELECT * FROM incidents ORDER BY created_at DESC LIMIT 500;
-- Fetch all signals
SELECT * FROM signals ORDER BY id;
```

- Real-time Updates: The dashboard does not write to the database. Instead, it listens for Socket.io events. When another user or an authority performs an

action that changes data (e.g., reports an incident), the backend writes to the database and then emits an event (e.g., incident:new). The dashboard receives this event and updates its state in real-time without needing to re-query the database.

4.3. Incident Management Workflow

This workflow demonstrates the full cycle of data creation, updating, and real-time broadcasting.

1. Reporting an Incident: A logged-in user submits a form. The backend validates the data and uses the reporter_id from the user's JWT to insert a new record into the incidents table with a default status of 'reported'.

SQL
```
INSERT INTO incidents (reporter_id, type, description, lat, lng, severity, status)
VALUES (?,?,?,?,?,?, 'reported');
```

2. Broadcasting the Update: Immediately after the INSERT is successful, the backend emits a Socket.io event to all connected clients.

JavaScript
```
req.io.emit('incident:new', newIncidentData);
```

3. Verifying/Resolving an Incident: An admin or authority user can change the status of an incident. This triggers a PATCH request to the API. The backend first verifies the user's role, then executes an UPDATE query on the incidents table.

SQL
```
UPDATE incidents SET status='verified' WHERE id=?;
```

4. Broadcasting the Status Change: Following the successful UPDATE, another Socket.io event is emitted to inform all clients of the status change, which allows the dashboard to update the incident's color in real-time.

JavaScript
```
req.io.emit('incident:update', { id: incidentId, status: 'verified' });
```

4.4. Signal Management

Only users with admin or authority roles can manage signals.

- Viewing Signals: A simple SELECT query retrieves all signals for display in a list or on the map.

SQL
```
SELECT * FROM signals ORDER BY id;
```

- Changing Signal Mode: An authorized user can change a signal's mode (e.g., to 'emergency'). This action triggers an UPDATE query in the database.

SQL
```
UPDATE signals SET mode=? WHERE id=?;
```

- Broadcasting the Change: Like with incidents, a signal:mode event is broadcast via Socket.io, ensuring all connected dashboards reflect the

change immediately.

4.5. Analytics

The analytics component performs aggregate queries on the database to provide summary statistics.

- Summary Data: The backend runs simple COUNT queries on the incidents and signals tables.

SQL
```
SELECT COUNT(*) as count FROM incidents;
SELECT COUNT(*) as count FROM signals;
```

- Heatmap Data: To generate data for an incident heatmap, the backend queries the incidents table for recent events, using the severity as a weight.

SQL
```
SELECT lat, lng, LEAST(5, GREATEST(1, severity)) as weight
FROM incidents
WHERE created_at > NOW() - INTERVAL 30 DAY;
```

5. METHODOLOGY

The system's functionality is achieved through a structured methodology that combines a modular development approach with a real-time, event-driven communication model.

5.1. System Development Approach

The project was developed using a component-based, modular architecture. This methodology involves a clear separation of concerns, which enhances maintainability and scalability:

- **Frontend (React):** Responsible for all user interface elements and client-side state management. It interacts with the backend via a defined set of API endpoints.
- **Backend (Node.js/Express):** Acts as the central logic and data access layer. It handles business logic, user authentication, and serves as the sole intermediary between the client and the database. This abstraction prevents direct client access to the database, enhancing security.
- **Database (MySQL):** Serves as the persistent data store and the single source of truth for the entire application.
- **Real-time Layer (Socket.io):** Operates in parallel with the REST API to push live updates from the server to all connected clients, enabling a dynamic and responsive user experience.

5.2. Data Acquisition and Processing

The system acquires data primarily through user-submitted reports. This participatory sensing approach leverages the on-the-ground knowledge of commuters and authorities. The methodology for data acquisition is as follows:

1. **Data Capture:** The React frontend provides intuitive forms for users to report incidents, capturing structured data such as type, description, location, and severity.
2. **Data Validation:** Before submission, client-side validation ensures data integrity. Upon receiving the data, the Express backend performs a second, more robust validation using schemas (e.g., Joi) to ensure all data conforms to the required format and constraints before it is sent to the database.
3. **Data Persistence:** Validated data is then translated into SQL queries and persisted in the MySQL database. For instance, a new incident report triggers an INSERT statement for the incidents table.

5.3. Real-Time Data Synchronization

A core methodological choice was to implement a real-time, push-based communication model using Socket.io rather than a traditional polling mechanism. This approach is significantly more efficient and provides instantaneous updates. The workflow is event-driven:

1. **Database Write Operation:** An action (e.g., creating an incident, changing a signal mode) results in a write operation (INSERT or UPDATE) to the MySQL database.
2. **Server-Side Event Emission:** Upon successful completion of the database transaction, the Express backend immediately emits a named Socket.io event (e.g., incident:new or signal:mode) containing the relevant new data.
3. **Client-Side Event Listening:** All connected clients are actively listening for these events. When an event is received, the frontend application updates its state directly with the new data, causing the UI to re-render and display the change instantly.

This methodology ensures that all users have a synchronized, up-to-the-minute view of the traffic situation without the overhead of continuous HTTP requests.

6.CONCLUSION

This case study has detailed the systematic design and implementation of a database for a Smart City Traffic Management System. The architecture, centered around a MySQL relational database, demonstrates how a well-designed schema is critical for application functionality, data integrity, and security. The use of a Node.js backend provides a clear separation between the database and the client,

while Socket.io bridges the gap for real-time data synchronization.
The specific SQL queries and workflows illustrate the direct relationship between user actions and database transactions, highlighting the database's role as the active, authoritative core of the system. This practical, model-driven approach—from conceptual design to concrete implementation—provides a robust blueprint for developing the data backend of any large-scale, data-intensive smart city application.

7. REFERENCES

1. Symmetry Electronics. (n.d.). *What is a Smart Traffic Management System?* Retrieved from https://www.symmetryelectronics.com/blog/what-is-a-smart-traffic-management-system/
2. Omnisight. (n.d.). *Smart City Traffic Management: The Complete Guide*. Retrieved from https://omnisightusa.com/blog/smart-city-traffic-management-complete-guide
3. Eastgate. (n.d.). *Exploring the Advantages and Disadvantages of Intelligent Traffic Systems*. Medium. Retrieved from https://medium.com/@eastgate/exploring-the-advantages-and-disadvantages-of-intelligent-traffic-systems-12f27cc858a0
4. HashStudioz. (n.d.). *IoT in Traffic Management: Transforming Urban Mobility*. Retrieved from https://www.hashstudioz.com/blog/iot-in-traffic-management/
5. GCPIT. (n.d.). *The ML Framework for Traffic Management in Smart Cities*. Retrieved from https://gcpit.org/the-ml-framework-for-traffic-management-in-smart-cities/
6. vCloud Tech. (n.d.). *The Role of Intelligent Traffic Management System in Modern Cities*. Retrieved from https://blog.vcloudtech.com/the-role-of-intelligent-traffic-management-system-in-modern-cities/
7. Intellias. (n.d.). *Intelligent Traffic Management: How to Build a Smart System*. Retrieved from https://intellias.com/intelligent-traffic-management/
8. Helix Traffic Solutions. (n.d.). *Emerging Technologies in Traffic Control*. Retrieved from https://www.helixtraffic.com/blog/emerging-technologies-in-traffic-control/
9. The Times of India. (2025, October 25). *Dubai launches AI system to link signals directly to vehicles, cutting congestion by up to 37%*. Retrieved from https://timesofindia.indiatimes.com/world/middle-east/dubai-launches-ai-system-to-link-signals-directly-to-vehicles-cutting-congestion-by-up-to-37/articleshow/124778177.cms

BCSE302P – Database Systems Lab

Report for the Project

Smart City Traffic Management System

22BCB0041 HIMANSHU UPADHYAY

23BCE0048 MANYA KOTHARI

23BCE0106 SHRISH SINGH SOURYA

23BCE0182 DHRITI SAXENA

School of Computer Science and Engineering

(SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT

Urban traffic congestion presents a significant and growing challenge to cities worldwide, leading to economic losses, environmental degradation, and a reduced quality of life. This report details the design and implementation of a full-stack Smart City Traffic Management System, a web application developed to address these challenges through real-time monitoring and control. The system is built on a robust three-tier architecture, utilizing a modern technology stack comprising React with TypeScript for the frontend, Node.js with Express for the backend, and MySQL as the relational database. Real-time, bidirectional communication is achieved using Socket.IO, enabling instantaneous updates on a live dashboard. The core functionalities include a multi-role user authentication system using JSON Web Tokens (JWT), a comprehensive incident reporting and management module, and a remote traffic signal control system for authorized personnel. The database is meticulously designed with a normalized relational schema to ensure data integrity, transactional reliability, and scalability. This project demonstrates a practical and effective database-centric solution for modern urban traffic management, providing a scalable foundation for future smart city integrations.

2. METHODOLOGY

The system is designed using a multi-tier architecture that separates concerns into a presentation tier (frontend), an application tier (backend), and a data tier (database). This model enhances scalability, maintainability, and security by decoupling the user interface from the core business logic and data storage.

The primary data flow is as follows:

- The Presentation Tier (React frontend) communicates with the Application Tier (Express backend) via a REST API for data retrieval and modification requests.
- The Application Tier contains the business logic, processing requests, and interacting with the Data Tier (MySQL database) to perform the necessary CRUD (Create, Read, Update, Delete) operations.
- For real-time updates, a persistent Socket.IO connection allows the Application Tier to push data changes to the Presentation Tier immediately after a database write operation, ensuring all clients are synchronized.

The technology stack includes:

- Frontend: React with TypeScript (Vite)
- Backend: Node.js with Express.js
- Database: MySQL
- Real-time Communication: Socket.IO
- Maps: Leaflet
- Authentication: JSON Web Tokens (JWT)

The data tier employs a relational database model implemented with MySQL, chosen for its high performance, reliability, and robust features for ensuring data

integrity and transactional support, which are paramount for a mission-critical system. The database is designed around four core
entities: Users, Signals, Incidents, and Congestion_Metrics, with relationships enforced through foreign key constraints to maintain referential integrity.
Module Description
Each functional module of the application is designed to perform specific CRUD operations on the database, which serves as the single source of truth.
User Authentication and Authorization This module manages user identity and access control, interacting exclusively with the Users table.

- Registration (Create): A new user is created via an INSERT statement. The password is first hashed using bcrypt in the application layer before being stored for security.
- Login (Read): User authentication involves a SELECT query to retrieve a user's record by their unique email. The application then compares the provided password with the stored password_hash.
- Authorization: Role-Based Access Control (RBAC) is implemented using middleware that checks the user's role (e.g., admin, authority, user), which is embedded in their JWT. This restricts access to sensitive operations like verifying incidents or changing signal modes.

Incident Management System This module handles the lifecycle of a traffic incident, primarily interacting with the Incidents table.

- Report Incident (Create): An authenticated user reports an incident via an INSERT operation. The reporter_id is linked to the Users table, enforcing referential integrity.
- View Incidents (Read): The dashboard is populated using a SELECT query to retrieve recent incidents.
- Verify/Resolve Incident (Update): Authorized users update an incident's status, which triggers an UPDATE statement in the database.

Traffic Signal Management System This module allows authorized users to control traffic signals, performing read and update operations on the Signals table.

- View Signals (Read): A list of all signals and their states is retrieved with a SELECT query.
- Change Signal Mode/Timing (Update): An authorized user can change a signal's parameters, which executes an UPDATE statement on the specific signal's record.

Analytics System This module performs data aggregation to provide insights, executing read-only queries on the database.

- Summary Statistics (Read): The system calculates summary metrics using aggregate functions like COUNT().
- Incident Heatmap (Read): Data for heatmap visualization is generated by selecting the location and severity of recent incidents within a given time frame.

3. PROJECT DEMONSTRATION – CODE, OUTPUT screen shot

This section demonstrates the key functionalities of the Smart City Traffic Management System through code snippets and screenshots of the user interface. Real-time Monitoring Dashboard The dashboard provides a live, map-based view of all traffic signals and incidents. Data is loaded initially via a REST API call and then updated in real-time using Socket.IO.

**

Incident Reporting System Users can report new incidents through a dedicated form. The data is sent to the backend, stored in the database, and broadcast to all connected clients.

**

Frontend Code for Incident Form (IncidentsPage.tsx):

Configuration Files

client/package.json

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc -b && vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "axios": "^1.13.1",
    "leaflet": "^1.9.4",
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-leaflet": "^5.0.0",
    "react-router-dom": "^7.9.5",
    "recharts": "^3.3.0",
    "socket.io-client": "^4.8.1"
  },
  "devDependencies": {
    "@eslint/js": "^9.36.0",
    "@types/node": "^24.6.0",
    "@types/react": "^19.1.16",
    "@types/react-dom": "^19.1.9",
    "@vitejs/plugin-react": "^5.0.4",
    "eslint": "^9.36.0",
```

```
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.22",
    "globals": "^16.4.0",
    "typescript": "~5.9.3",
    "typescript-eslint": "^8.45.0",
    "vite": "^7.1.7"
  }
}
```

client/index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>client</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

client/vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
// https://vite.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

client/tsconfig.json

```
{
  "files": [],
  "references": [
    { "path": "./tsconfig.app.json" },
    { "path": "./tsconfig.node.json" }
  ]
}
```

client/tsconfig.app.json

```
{
  "compilerOptions": {
    "tsBuildInfoFile": "./node_modules/.tmp/tsconfig.app.tsbuildinfo",
    "target": "ES2022",
    "useDefineForClassFields": true,
```

```
    "lib": ["ES2022", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "types": ["vite/client"],
    "skipLibCheck": true,
    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "verbatimModuleSyntax": true,
    "moduleDetection": "force",
    "noEmit": true,
    "jsx": "react-jsx",
    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "erasableSyntaxOnly": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true
  },
  "include": ["src"]
}
```

client/tsconfig.node.json

```
{
  "compilerOptions": {
    "tsBuildInfoFile": "./node_modules/.tmp/tsconfig.node.tsbuildinfo",
    "target": "ES2023",
    "lib": ["ES2023"],
    "module": "ESNext",
    "types": ["node"],
    "skipLibCheck": true,
    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "verbatimModuleSyntax": true,
    "moduleDetection": "force",
    "noEmit": true,
    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "erasableSyntaxOnly": true,
    "noFallthroughCasesInSwitch": true,
    "noUncheckedSideEffectImports": true
```

```
  },
  "include": ["vite.config.ts"]
}
```

client/eslint.config.js

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'
import { defineConfig, globalIgnores } from 'eslint/config'
export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{ts,tsx}'],
    extends: [
      js.configs.recommended,
      tseslint.configs.recommended,
      reactHooks.configs['recommended-latest'],
      reactRefresh.configs.vite,
    ],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
  },
])
```

client/env.example

```
VITE_API_BASE=http://localhost:4000/api
VITE_SOCKET_URL=http://localhost:4000
```

Source Files

client/src/main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { BrowserRouter } from 'react-router-dom'
import App from './App.tsx'
import './index.css'
import 'leaflet/dist/leaflet.css'
import { AuthProvider } from './state/AuthContext.tsx'
import { ThemeProvider } from './state/ThemeContext.tsx'
ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <BrowserRouter>
```

```
      <ThemeProvider>
        <AuthProvider>
          <App />
        </AuthProvider>
      </ThemeProvider>
    </BrowserRouter>
  </React.StrictMode>,
)
```

client/src/App.tsx

```
import { Navigate, Route, Routes } from 'react-router-dom'
import { useAuth } from './state/AuthContext'
import LoginPage from './pages/LoginPage'
import DashboardPage from './pages/DashboardPage'
import IncidentsPage from './pages/IncidentsPage'
import SignalsPage from './pages/SignalsPage'
import AnalyticsPage from './pages/AnalyticsPage'
import Layout from './components/Layout'
function Protected({ children }: { children: JSX.Element }) {
  const { token } = useAuth()
  if (!token) return <Navigate to="/login" replace />
  return children
}
export default function App() {
  return (
    <Routes>
      <Route path="/login" element={<LoginPage />} />
      <Route path="/" element={<Protected><Layout><DashboardPage /></Layout
></Protected>} />
      <Route path="/incidents" element={<Protected><Layout><IncidentsPage /></
Layout></Protected>} />
      <Route path="/signals" element={<Protected><Layout><SignalsPage /></Lay
out></Protected>} />
      <Route path="/analytics" element={<Protected><Layout><AnalyticsPage /></
Layout></Protected>} />
    </Routes>
  )
}
```

client/src/index.css

```
:root {
  --bg: #0b0f14;
  --panel: #121821;
  --text: #e6edf3;
  --muted: #9fb0c3;
```

```
  --primary: #4f8cff;
  --primary-600: #3b6ed1;
  --danger: #ff6b6b;
  --success: #38d996;
  --border: #223042;
}
[data-theme="light"] {
  --bg: #f7f9fc;
  --panel: #ffffff;
  --text: #0f1720;
  --muted: #5b6b7c;
  --primary: #2b67f6;
  --primary-600: #1f4fbd;
  --danger: #e14d4d;
  --success: #2fb07c;
  --border: #e6eef7;
}
* { box-sizing: border-box; }
html, body, #root { height: 100%; }
body {
  margin: 0;
  background: var(--bg);
  color: var(--text);
  font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;
}
/* Layout */
.app-shell { display: grid; grid-template-columns: 260px 1fr; grid-template-
rows: 56px 1fr; height: 100%; }
.app-shell .header { grid-column: 1 / -1; display: flex; align-
items: center; gap: 12px; padding: 0 16px; border-bottom: 1px solid var(--
border); background: var(--panel); position: sticky; top: 0; z-index: 5; }
.app-shell .sidebar { display: flex; flex-direction: column; gap: 8px; border-
right: 1px solid var(--border); background: var(--panel); padding: 12px; }
.app-shell .content { padding: 16px; overflow: auto; }
.logo { font-weight: 700; letter-spacing: 0.3px; }
.spacer { flex: 1 1 auto; }
/* Links and buttons */
a { color: var(--primary); text-decoration: none; }
a:hover { text-decoration: underline; }
.button { background: var(--primary); color: white; border: none; border-
radius: 8px; padding: 10px 14px; cursor: pointer; }
.button.secondary { background: transparent; color: var(--
text); border: 1px solid var(--border); }
```

```
.button.ghost { background: transparent; color: var(--muted); }
.button:disabled { opacity: 0.6; cursor: not-allowed; }
/* Nav */
.nav-link { display: block; padding: 10px 12px; border-radius: 8px; color: var(--
text); }
.nav-link.active, .nav-link:hover { background: rgba(79,140,255,0.15); }
/* Cards */
.card { background: var(--panel); border: 1px solid var(--border); border-
radius: 12px; padding: 16px; }
.grid { display: grid; gap: 16px; }
.grid.cols-2 { grid-template-columns: repeat(2, minmax(0, 1fr)); }
.grid.cols-3 { grid-template-columns: repeat(3, minmax(0, 1fr)); }
@media (max-width: 900px) {
  .app-shell { grid-template-columns: 1fr; grid-template-rows: 56px auto 1fr; }
  .app-shell .sidebar { grid-row: 2; flex-direction: row; overflow-x: auto; }
}
/* Forms & tables */
input, select { width: 100%; padding: 10px 12px; background: transparent; color: v
ar(--text); border: 1px solid var(--border); border-radius: 8px; }
label { display: block; font-size: 12px; color: var(--muted); margin-bottom: 6px; }
.table { width: 100%; border-collapse: collapse; }
.table th, .table td { padding: 10px 12px; border-bottom: 1px solid var(--border); }
.table th { text-align: left; color: var(--muted); font-weight: 500; }
/* Map */
.map-card { padding: 0; overflow: hidden; }
.map-header { display: flex; align-items: center; justify-content: space-
between; padding: 12px 16px; border-bottom: 1px solid var(--border); }
.map-container { height: 70vh; }
```

client/src/App.css

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
```

```
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}
@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}
.card {
  padding: 2em;
}
.read-the-docs {
  color: #888;
}
```

Components

client/src/components/Layout.tsx

```
import { NavLink } from 'react-router-dom'
import { useAuth } from '../state/AuthContext'
import { useTheme } from '../state/ThemeContext'
export default function Layout({ children }: { children: React.ReactNode }) {
  const { user, logout } = useAuth()
  const { theme, toggle } = useTheme()
  return (
    <div className="app-shell">
      <div className="header">
        <div className="logo">Traffic Control</div>
        <div className="spacer" />
        <button className="button ghost" onClick={toggle}>{theme === 'dark' ? 'Li
ght' : 'Dark'} mode</button>
        {user ? (
          <button className="button secondary" onClick={logout} style={{ marginL
eft: 8 }}>Logout</button>
        ) : null}
```

```
      </div>
      <aside className="sidebar">
        <NavLink to="/" className={({ isActive }) => `nav-
link ${isActive ? 'active' : ''}`}>Dashboard</NavLink>
        <NavLink to="/incidents" className={({ isActive }) => `nav-
link ${isActive ? 'active' : ''}`}>Incidents</NavLink>
        <NavLink to="/signals" className={({ isActive }) => `nav-
link ${isActive ? 'active' : ''}`}>Signals</NavLink>
        <NavLink to="/analytics" className={({ isActive }) => `nav-
link ${isActive ? 'active' : ''}`}>Analytics</NavLink>
        <div className="spacer" />
        <div style={{ fontSize: 12, color: 'var(--
muted)' }}>{user ? `${user.name} • ${user.role}` : 'Guest'}</div>
      </aside>
      <main className="content">
        {children}
      </main>
    </div>
  )
}
```

client/src/components/LiveMap.tsx

```
import { MapContainer, TileLayer, Marker, Popup, Circle } from 'react-leaflet'
import { useEffect, useState } from 'react'
import axios from 'axios'
import { io, Socket } from 'socket.io-client'
const SOCKET_URL = import.meta.env.VITE_SOCKET_URL || 'http://localhost:
4000'
type Incident = { id: number; type: string; description: string; lat: number; lng: nu
mber; severity: number; status: string }
type Signal = { id: number; name: string; lat: number; lng: number; mode: string }
export default function LiveMap() {
  const [incidents, setIncidents] = useState<Incident[]>([])
  const [signals, setSignals] = useState<Signal[]>([])
  useEffect(() => {
    axios.get<Incident[]>('/incidents').then(r => setIncidents(r.data))
    axios.get<Signal[]>('/traffic/signals').then(r => setSignals(r.data))
  }, [])
  useEffect(() => {
    const s: Socket = io(SOCKET_URL, { transports: ['websocket'] })
    s.on('incident:new', (i: Incident) => setIncidents(prev => [i, ...prev]))
    s.on('incident:update', (u: Partial<Incident> & { id: number }) => setIncidents(pr
ev => prev.map(p => p.id === u.id ? { ...p, ...u } : p)))
    s.on('signal:mode', (u: { id: number; mode: string }) => setSignals(prev => prev.
```

```
map(p => p.id === u.id ? { ...p, mode: u.mode } : p)))
    s.on('signal:update', (u: any) => {})
    return () => { s.disconnect() }
  }, [])
  return (
    <MapContainer center={[28.6139, 77.2090]} zoom={12} style={{ height: '70vh
', width: '100%' }}>
      <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" attribution
="&copy; OpenStreetMap contributors" />
      {signals.map(s => (
        <Marker key={s.id} position={[s.lat, s.lng]}>
          <Popup>
            <div>
              <b>{s.name}</b>
              <div>Mode: {s.mode}</div>
            </div>
          </Popup>
        </Marker>
      ))}
      {incidents.map(i => (
        <Circle key={i.id} center={[i.lat, i.lng]} radius={50 + i.severity * 25} pathO
ptions={{ color: i.status === 'resolved' ? 'green' : i.status === 'verified' ? 'orange' : '
red' }}>
          <Popup>
            <div>
              <b>{i.type}</b>
              <div>{i.description}</div>
              <div>Status: {i.status}</div>
            </div>
          </Popup>
        </Circle>
      ))}
    </MapContainer>
  )
}
```

Pages

client/src/pages/LoginPage.tsx

```
import { useState } from 'react'
import { useAuth } from '../state/AuthContext'
import { useNavigate } from 'react-router-dom'
export default function LoginPage() {
  const { login, register } = useAuth()
```

```
  const nav = useNavigate()
  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')
  const [name, setName] = useState('')
  const [mode, setMode] = useState<'login' | 'register'>('login')
  const [error, setError] = useState<string | null>(null)
  const [loading, setLoading] = useState(false)
  const submit = async (e: React.FormEvent) => {
    e.preventDefault()
    setError(null)
    setLoading(true)
    try {
      if (mode === 'login') {
        await login(email, password)
      } else {
        await register(name, email, password)
      }
      nav('/')
    } catch (err: any) {
      setError(err?.response?.data?.message || 'Failed')
    } finally {
      setLoading(false)
    }
  }
  return (
    <div style={{ display: 'grid', placeItems: 'center', height: '100%' }}>
      <div className="card" style={{ width: 380 }}>
        <h2 style={{ marginTop: 0, marginBottom: 8 }}>{mode === 'login' ? 'Welco
me back' : 'Create account'}</h2>
        <p style={{ color: 'var(--
muted)', marginTop: 0 }}>{mode === 'login' ? 'Sign in to continue' : 'Join to report
 and monitor traffic'}</p>
        <form onSubmit={submit} className="grid" style={{ gap: 12 }}>
          {mode === 'register' && (
            <div>
              <label>Name</label>
              <input placeholder="Your name" value={name} onChange={e => setNa
me(e.target.value)} required />
            </div>
          )}
          <div>
            <label>Email</label>
            <input placeholder="you@example.com" value={email} onChange={e =>
```

```
setEmail(e.target.value)} type="email" required />
          </div>
          <div>
            <label>Password</label>
            <input placeholder="••••••••" value={password} onChange={e => setPass
word(e.target.value)} type="password" required />
          </div>
          {error && <div style={{ color: 'var(--danger)' }}>{error}</div>}
          <button className="button" type="submit" disabled={loading}>{loading ?
'Please wait...' : (mode === 'login' ? 'Login' : 'Create account')}</button>
        </form>
        <button className="button ghost" style={{ width: '100%', marginTop: 8 }}
onClick={() => setMode(mode === 'login' ? 'register' : 'login')}>
          {mode === 'login' ? 'Need an account?' : 'Have an account?'}
        </button>
      </div>
    </div>
  )
}
```

client/src/pages/DashboardPage.tsx

```
import LiveMap from '../components/LiveMap'
export default function DashboardPage() {
  return (
    <div className="grid">
      <div className="card map-card">
        <div className="map-header">
          <h2 style={{ margin: 0 }}>Real-time Traffic Monitoring</h2>
          <div style={{ color: 'var(--
muted)' }}>Incidents and signals update live</div>
        </div>
        <div className="map-container">
          <LiveMap />
        </div>
      </div>
    </div>
  )
}
```

client/src/pages/IncidentsPage.tsx

```
import { useEffect, useState } from 'react'
import axios from 'axios'
type Incident = { id: number; type: string; description: string; lat: number; lng: nu
mber; severity: number; status: string; created_at: string }
export default function IncidentsPage() {
```

```
  const [incidents, setIncidents] = useState<Incident[]>([])
  const [form, setForm] = useState({ type: 'accident', description: '', lat: 28.6139, ln
g: 77.2090, severity: 2 })
  const [error, setError] = useState<string | null>(null)
  const [loading, setLoading] = useState(false)
  useEffect(() => {
    axios.get<Incident[]>('/incidents').then(r => setIncidents(r.data))
  }, [])
  const submit = async (e: React.FormEvent) => {
    e.preventDefault()
    setError(null)
    setLoading(true)
    try {
      const res = await axios.post('/incidents', form)
      setIncidents(prev => [res.data, ...prev])
      setForm({ ...form, description: '' })
    } catch (err: any) {
      setError(err?.response?.data?.message || 'Failed')
    } finally {
      setLoading(false)
    }
  }
  return (
    <div className="grid cols-2">
      <div className="card">
        <h3 style={{ marginTop: 0 }}>Report Incident</h3>
        <form onSubmit={submit} className="grid" style={{ gap: 12 }}>
          <div>
            <label>Type</label>
            <select value={form.type} onChange={e => setForm({ ...form, type: e.targ
et.value })}>
              <option value="accident">Accident</option>
              <option value="breakdown">Breakdown</option>
              <option value="construction">Construction</option>
              <option value="closure">Closure</option>
              <option value="hazard">Hazard</option>
              <option value="other">Other</option>
            </select>
          </div>
          <div>
            <label>Description</label>
            <input placeholder="What happened?" value={form.description} onChang
e={e => setForm({ ...form, description: e.target.value })} />
```

```
        </div>
        <div style={{ display: 'flex', gap: 8 }}>
          <div style={{ flex: 1 }}>
            <label>Latitude</label>
            <input type="number" step="0.0001" value={form.lat} onChange={e =>
setForm({ ...form, lat: Number(e.target.value) })} />
          </div>
          <div style={{ flex: 1 }}>
            <label>Longitude</label>
            <input type="number" step="0.0001" value={form.lng} onChange={e =>
setForm({ ...form, lng: Number(e.target.value) })} />
          </div>
          <div style={{ width: 120 }}>
            <label>Severity</label>
            <input type="number" min={1} max={5} value={form.severity} onChan
ge={e => setForm({ ...form, severity: Number(e.target.value) })} />
          </div>
        </div>
        {error && <div style={{ color: 'var(--danger)' }}>{error}</div>}
        <div>
          <button className="button" type="submit" disabled={loading}>{loading
? 'Reporting...' : 'Report'}</button>
        </div>
      </form>
    </div>
    <div className="card">
      <h3 style={{ marginTop: 0 }}>Recent Incidents</h3>
      <table className="table">
        <thead>
          <tr><th>Type</th><th>Status</th><th>Severity</th><th>Time</th></tr>
        </thead>
        <tbody>
          {incidents.map(i => (
            <tr key={i.id}>
              <td style={{ textTransform: 'capitalize' }}>{i.type}</td>
              <td>{i.status}</td>
              <td>{i.severity}</td>
              <td>{new Date(i.created_at).toLocaleString()}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
```

```
    </div>
  )
}
```

client/src/pages/SignalsPage.tsx

```
import { useEffect, useState } from 'react'
import axios from 'axios'
interface Signal { id: number; name: string; lat: number; lng: number; mode: 'norm
al' | 'flashing' | 'emergency' }
export default function SignalsPage() {
  const [signals, setSignals] = useState<Signal[]>([])
  const [saving, setSaving] = useState<number | null>(null)
  useEffect(() => {
    axios.get<Signal[]>('/traffic/signals').then(r => setSignals(r.data))
  }, [])
  const setMode = async (id: number, mode: Signal['mode']) => {
    setSaving(id)
    try {
      await axios.put(`/traffic/signals/${id}/mode`, { mode })
      setSignals(prev => prev.map(s => s.id === id ? { ...s, mode } : s))
    } finally {
      setSaving(null)
    }
  }
  return (
    <div className="card">
      <h3 style={{ marginTop: 0 }}>Traffic Signals</h3>
      <table className="table">
        <thead>
          <tr><th>Name</th><th>Coordinates</th><th>Mode</th><th>Actions</th>
</tr>
        </thead>
        <tbody>
          {signals.map(s => (
            <tr key={s.id}>
              <td>{s.name}</td>
              <td>{s.lat.toFixed(4)}, {s.lng.toFixed(4)}</td>
              <td>{s.mode}</td>
              <td style={{ display: 'flex', gap: 6 }}>
                <button className="button secondary" disabled={saving===s.id} onCli
ck={() => setMode(s.id, 'normal')}>Normal</button>
                <button className="button secondary" disabled={saving===s.id} onCli
ck={() => setMode(s.id, 'flashing')}>Flashing</button>
                <button className="button" disabled={saving===s.id} onClick={() =>
```

```
setMode(s.id, 'emergency')}>Emergency</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
 )
}
```

client/src/pages/AnalyticsPage.tsx

```
import { useEffect, useState } from 'react'
import axios from 'axios'
import { LineChart, Line, XAxis, YAxis, Tooltip, ResponsiveContainer } from 'rec
harts'
type Summary = { incidentCount: number; signalCount: number }
type Point = { x: string; y: number }
export default function AnalyticsPage() {
 const [summary, setSummary] = useState<Summary | null>(null)
 const [trend, setTrend] = useState<Point[]>([])
 useEffect(() => {
  axios.get<Summary>('/analytics/summary').then(r => setSummary(r.data))
  setTrend(Array.from({ length: 12 }).map((_, i) => ({ x: `M${i+1}`, y: Math.rou
nd(10 + Math.random()*50) })))
 }, [])
 return (
  <div className="grid cols-2">
   <div className="card">
    <h3 style={{ marginTop: 0 }}>Summary</h3>
    {summary && (
     <div style={{ display: 'flex', gap: 24 }}>
      <div><b>Incidents:</b> {summary.incidentCount}</div>
      <div><b>Signals:</b> {summary.signalCount}</div>
     </div>
    )}
   </div>
   <div className="card" style={{ gridColumn: '1 / -1', height: 360 }}>
    <h3 style={{ marginTop: 0 }}>Monthly Incident Trend</h3>
    <div style={{ height: 300 }}>
     <ResponsiveContainer width="100%" height="100%">
      <LineChart data={trend}>
       <XAxis dataKey="x" />
       <YAxis />
       <Tooltip />
```

```
          <Line type="monotone" dataKey="y" stroke="#4f8cff" strokeWidth={2}
/>
         </LineChart>
        </ResponsiveContainer>
       </div>
     </div>
   </div>
  )
}
```

State Management

client/src/state/AuthContext.tsx

```
import React, { createContext, useContext, useEffect, useMemo, useState } from 'r
eact'
import axios from 'axios'
type User = { id: number; name: string; email: string; role: 'admin' | 'authority' | 'us
er' }
type AuthContextValue = {
  user: User | null
  token: string | null
  login: (email: string, password: string) => Promise<void>
  register: (name: string, email: string, password: string) => Promise<void>
  logout: () => void
}
const AuthContext = createContext<AuthContextValue | undefined>(undefined)
const API_BASE = import.meta.env.VITE_API_BASE || 'http://localhost:4000/api'
axios.defaults.baseURL = API_BASE
export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [token, setToken] = useState<string | null>(() => localStorage.getItem('toke
n'))
  const [user, setUser] = useState<User | null>(null)
  useEffect(() => {
    if (token) {
      axios.defaults.headers.common['Authorization'] = `Bearer ${token}`
      localStorage.setItem('token', token)
      axios.get<User>('/auth/me').then(r => setUser(r.data)).catch(() => setUser(null)
)
    } else {
      delete axios.defaults.headers.common['Authorization']
      localStorage.removeItem('token')
      setUser(null)
    }
  }, [token])
```

```
  const login = async (email: string, password: string) => {
    const res = await axios.post('/auth/login', { email, password })
    setToken(res.data.token)
    setUser(res.data.user)
  }
  const register = async (name: string, email: string, password: string) => {
    await axios.post('/auth/register', { name, email, password })
    await login(email, password)
  }
  const logout = () => setToken(null)
  const value = useMemo(() => ({ user, token, login, register, logout }), [user, toke
n])
  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider
>
}
export function useAuth() {
  const ctx = useContext(AuthContext)
  if (!ctx) throw new Error('useAuth must be used within AuthProvider')
  return ctx
}
```

client/src/state/ThemeContext.tsx

```
import React, { createContext, useContext, useEffect, useMemo, useState } from 'r
eact'
type Theme = 'dark' | 'light'
type ThemeContextValue = {
  theme: Theme
  toggle: () => void
}
const ThemeContext = createContext<ThemeContextValue | undefined>(undefine
d)
export function ThemeProvider({ children }: { children: React.ReactNode }) {
  const [theme, setTheme] = useState<Theme>(() => (localStorage.getItem('theme')
 as Theme) || 'dark')
  useEffect(() => {
    localStorage.setItem('theme', theme)
    if (theme === 'light') {
      document.documentElement.setAttribute('data-theme', 'light')
    } else {
      document.documentElement.removeAttribute('data-theme')
    }
  }, [theme])
  const toggle = () => setTheme(prev => (prev === 'dark' ? 'light' : 'dark'))
  const value = useMemo(() => ({ theme, toggle }), [theme])
```

```
  return <ThemeContext.Provider value={value}>{children}</ThemeContext.Prov
ider>
}
export function useTheme() {
  const ctx = useContext(ThemeContext)
  if (!ctx) throw new Error('useTheme must be used within ThemeProvider')
  return ctx
}
```

Backend Code for Creating an Incident (routes/incidents.js):

JavaScript

```
//
```

Configuration files

server/package.json

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "City Traffic Management Backend (Express + MySQL + Socket.i
o)",
  "main": "src/index.js",
  "scripts": {
    "dev": "nodemon src/index.js",
    "start": "node src/index.js",
    "lint": "echo 'no linter configured'"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "bcryptjs": "^3.0.2",
    "cors": "^2.8.5",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "express-rate-limit": "^8.2.0",
    "helmet": "^8.1.0",
    "joi": "^18.0.1",
    "jsonwebtoken": "^9.0.2",
    "morgan": "^1.10.1",
    "mysql2": "^3.15.3",
    "socket.io": "^4.8.1"
  },
  "devDependencies": {
    "nodemon": "^3.1.10"
```

```
  }
}
```

Main application
server/src/index.js

```
const http = require('http');
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
const { Server } = require('socket.io');
const dotenv = require('dotenv');
dotenv.config({ path: process.env.ENV_PATH || undefined });
const { getDbPool } = require('./lib/db');
const authRoutes = require('./routes/auth');
const trafficRoutes = require('./routes/traffic');
const incidentRoutes = require('./routes/incidents');
const analyticsRoutes = require('./routes/analytics');
const adminRoutes = require('./routes/admin');
const { attachIoToRequest } = require('./socket/attachIo');
const app = express();
const server = http.createServer(app);
const io = new Server(server, {
  cors: {
    origin: process.env.CORS_ORIGIN || '*',
    methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE']
  }
});
// Basic security and parsing
app.use(helmet());
app.use(cors({ origin: process.env.CORS_ORIGIN || '*', credentials: true }));
app.use(express.json({ limit: '1mb' }));
app.use(morgan('dev'));
// Rate limit public endpoints
const limiter = rateLimit({ windowMs: 60 * 1000, max: 120 });
app.use(limiter);
// Health
app.get('/health', async (_req, res) => {
  try {
    const pool = getDbPool();
    await pool.query('SELECT 1');
    res.json({ status: 'ok' });
```

```
    } catch (e) {
        res.status(500).json({ status: 'error' });
    }
});
// Attach io to request for emitting inside routes
app.use(attachIoToRequest(io));
// API routes
app.use('/api/auth', authRoutes);
app.use('/api/traffic', trafficRoutes);
app.use('/api/incidents', incidentRoutes);
app.use('/api/analytics', analyticsRoutes);
app.use('/api/admin', adminRoutes);
// Socket events
io.on('connection', (socket) => {
    // Rooms by role or city segment can be added later
    socket.on('disconnect', () => {});
});
const PORT = process.env.PORT || 4000;
server.listen(PORT, () => {
    console.log(`Server listening on :${PORT}`);
});
```

Database

server/src/lib/db.js

```
const mysql = require('mysql2/promise');
const dotenv = require('dotenv');
dotenv.config({ path: process.env.ENV_PATH || undefined });
let pool;
function createPool() {
    if (pool) return pool;
    pool = mysql.createPool({
        host: process.env.DB_HOST || 'localhost',
        port: Number(process.env.DB_PORT || 3306),
        user: process.env.DB_USER || 'root',
        password: process.env.DB_PASSWORD || '',
        database: process.env.DB_NAME || 'traffic_db',
        waitForConnections: true,
        connectionLimit: 10,
        queueLimit: 0,
    });
    return pool;
}
function getDbPool() {
```

```
  return pool || createPool();
}
module.exports = { getDbPool };
```

server/db/schema.sql

```
-- Database: traffic_db
CREATE TABLE IF NOT EXISTS users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('admin','authority','user') NOT NULL DEFAULT 'user',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE IF NOT EXISTS signals (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  lat DECIMAL(10,7) NOT NULL,
  lng DECIMAL(10,7) NOT NULL,
  cycle_seconds INT NOT NULL DEFAULT 90,
  green_seconds INT NOT NULL DEFAULT 40,
  yellow_seconds INT NOT NULL DEFAULT 5,
  red_seconds INT NOT NULL DEFAULT 45,
  mode ENUM('normal','flashing','emergency') NOT NULL DEFAULT 'normal',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE IF NOT EXISTS incidents (
  id INT AUTO_INCREMENT PRIMARY KEY,
  reporter_id INT NOT NULL,
  type ENUM('accident','breakdown','construction','closure','hazard','other') NOT
NULL,
  description VARCHAR(500) NOT NULL DEFAULT '',
  lat DECIMAL(10,7) NOT NULL,
  lng DECIMAL(10,7) NOT NULL,
  severity TINYINT NOT NULL DEFAULT 2,
  status ENUM('reported','verified','resolved') NOT NULL DEFAULT 'reported',
  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (reporter_id) REFERENCES users(id) ON DELETE CASCAD
E
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE TABLE IF NOT EXISTS congestion_metrics (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  segment VARCHAR(100) NOT NULL,
  congestion_level TINYINT NOT NULL,
```

```
    lat DECIMAL(10,7) NOT NULL,
    lng DECIMAL(10,7) NOT NULL,
    observed_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    INDEX (segment),
    INDEX (observed_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Middleware

server/src/middleware/auth.js

```
const jwt = require('jsonwebtoken');
function authenticateJwt(req, res, next) {
    const authHeader = req.headers.authorization || '';
    const token = authHeader.startsWith('Bearer ') ? authHeader.slice(7) : null;
    if (!token) return res.status(401).json({ message: 'Unauthorized' });
    try {
        const payload = jwt.verify(token, process.env.JWT_SECRET || 'dev');
        req.user = payload;
        return next();
    } catch (err) {
        return res.status(401).json({ message: 'Invalid token' });
    }
}
function requireRoles(...roles) {
    return (req, res, next) => {
        if (!req.user) return res.status(401).json({ message: 'Unauthorized' });
        if (!roles.includes(req.user.role)) {
            return res.status(403).json({ message: 'Forbidden' });
        }
        next();
    };
}
module.exports = { authenticateJwt, requireRoles };
```

Routes

server/src/routes/auth.js

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const Joi = require('joi');
const { getDbPool } = require('../lib/db');
const { authenticateJwt } = require('../middleware/auth');
const router = express.Router();
const registerSchema = Joi.object({
```

```
    name: Joi.string().min(2).max(100).required(),
    email: Joi.string().email().required(),
    password: Joi.string().min(8).max(100).required(),
    role: Joi.string().valid('admin', 'authority', 'user').default('user'),
});
router.post('/register', async (req, res) => {
    const { error, value } = registerSchema.validate(req.body);
    if (error) return res.status(400).json({ message: error.message });
    const { name, email, password, role } = value;
    const pool = getDbPool();
    try {
        const [existing] = await pool.query('SELECT id FROM users WHERE email
= ?', [email]);
        if (existing.length) return res.status(409).json({ message: 'Email already regist
ered' });
        const hashed = await bcrypt.hash(password, 10);
        await pool.query(
            'INSERT INTO users (name, email, password_hash, role) VALUES (?, ?, ?
, ?)',
            [name, email, hashed, role]
        );
        return res.status(201).json({ message: 'Registered' });
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
const loginSchema = Joi.object({
    email: Joi.string().email().required(),
    password: Joi.string().required(),
});
router.post('/login', async (req, res) => {
    const { error, value } = loginSchema.validate(req.body);
    if (error) return res.status(400).json({ message: error.message });
    const { email, password } = value;
    const pool = getDbPool();
    try {
        const [rows] = await pool.query('SELECT id, name, email, password_hash, ro
le FROM users WHERE email = ?', [email]);
        if (!rows.length) return res.status(401).json({ message: 'Invalid credentials' });
        const user = rows[0];
        const ok = await bcrypt.compare(password, user.password_hash);
        if (!ok) return res.status(401).json({ message: 'Invalid credentials' });
        const token = jwt.sign({ id: user.id, role: user.role, name: user.name }, proces
```

```
s.env.JWT_SECRET || 'dev', {
          expiresIn: process.env.JWT_EXPIRES_IN || '7d',
        });
        return res.json({ token, user: { id: user.id, name: user.name, email: user.email,
 role: user.role } });
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
router.get('/me', authenticateJwt, async (req, res) => {
    const pool = getDbPool();
    try {
        const [rows] = await pool.query('SELECT id, name, email, role FROM users
WHERE id = ?', [req.user.id]);
        if (!rows.length) return res.status(404).json({ message: 'Not found' });
        return res.json(rows[0]);
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
module.exports = router;
```

server/src/routes/incidents.js

```
const express = require('express');
const Joi = require('joi');
const { authenticateJwt, requireRoles } = require('../middleware/auth');
const { getDbPool } = require('../lib/db');
const router = express.Router();
const createSchema = Joi.object({
    type: Joi.string().valid('accident', 'breakdown', 'construction', 'closure', 'hazard', '
other').required(),
    description: Joi.string().max(500).allow('').default(''),
    lat: Joi.number().required(),
    lng: Joi.number().required(),
    severity: Joi.number().integer().min(1).max(5).default(2),
});
router.post('/', authenticateJwt, async (req, res) => {
    const { error, value } = createSchema.validate(req.body);
    if (error) return res.status(400).json({ message: error.message });
    const { type, description, lat, lng, severity } = value;
    try {
        const [result] = await getDbPool().query(
          'INSERT INTO incidents (reporter_id, type, description, lat, lng, severity, s
tatus) VALUES (?, ?, ?, ?, ?, ?, ?)',
```

```
            [req.user.id, type, description, lat, lng, severity, 'reported']
        );
        const incident = { id: result.insertId, type, description, lat, lng, severity, status:
 'reported' };
        req.io.emit('incident:new', incident);
        return res.status(201).json(incident);
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
router.get('/', authenticateJwt, async (_req, res) => {
    try {
        const [rows] = await getDbPool().query('SELECT * FROM incidents ORDE
R BY created_at DESC LIMIT 500');
        return res.json(rows);
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
router.patch('/:id/verify', authenticateJwt, requireRoles('admin', 'authority'), async (
req, res) => {
    const { id } = req.params;
    try {
        await getDbPool().query('UPDATE incidents SET status=? WHERE id=?', ['v
erified', id]);
        req.io.emit('incident:update', { id: Number(id), status: 'verified' });
        return res.json({ message: 'Verified' });
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
router.patch('/:id/resolve', authenticateJwt, requireRoles('admin', 'authority'), async
(req, res) => {
    const { id } = req.params;
    try {
        await getDbPool().query('UPDATE incidents SET status=? WHERE id=?', ['r
esolved', id]);
        req.io.emit('incident:update', { id: Number(id), status: 'resolved' });
        return res.json({ message: 'Resolved' });
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
```

```
module.exports = router;
```

server/src/routes/traffic.js

```
const express = require('express');
const { authenticateJwt, requireRoles } = require('../middleware/auth');
const { getDbPool } = require('../lib/db');
const router = express.Router();
router.get('/signals', authenticateJwt, async (_req, res) => {
  try {
    const [rows] = await getDbPool().query('SELECT * FROM signals ORDER
BY id');
    return res.json(rows);
  } catch (e) {
    return res.status(500).json({ message: 'Server error' });
  }
});
router.put('/signals/:id/timing', authenticateJwt, requireRoles('admin', 'authority'), a
sync (req, res) => {
  const { id } = req.params;
  const { cycle_seconds, green_seconds, yellow_seconds, red_seconds } = req.bo
dy || {};
  try {
    await getDbPool().query(
      'UPDATE signals SET cycle_seconds=?, green_seconds=?, yellow_second
s=?, red_seconds=? WHERE id=?',
      [cycle_seconds, green_seconds, yellow_seconds, red_seconds, id]
    );
    req.io.emit('signal:update', { id: Number(id), cycle_seconds, green_seconds, y
ellow_seconds, red_seconds });
    return res.json({ message: 'Updated' });
  } catch (e) {
    return res.status(500).json({ message: 'Server error' });
  }
});
router.put('/signals/:id/mode', authenticateJwt, requireRoles('admin', 'authority'), as
ync (req, res) => {
  const { id } = req.params;
  const { mode } = req.body || {}; // normal, flashing, emergency
  try {
    await getDbPool().query('UPDATE signals SET mode=? WHERE id=?', [mo
de, id]);
    req.io.emit('signal:mode', { id: Number(id), mode });
    return res.json({ message: 'Mode updated' });
  } catch (e) {
```

```
        return res.status(500).json({ message: 'Server error' });
    }
});
module.exports = router;
```

server/src/routes/analytics.js

```
const express = require('express');
const { authenticateJwt, requireRoles } = require('../middleware/auth');
const { getDbPool } = require('../lib/db');
const router = express.Router();
router.get('/summary', authenticateJwt, requireRoles('admin', 'authority'), async (_r
eq, res) => {
    try {
        const pool = getDbPool();
        const [[{ count: incidentCount }]] = await pool.query('SELECT COUNT(*) a
s count FROM incidents');
        const [[{ count: signalCount }]] = await pool.query('SELECT COUNT(*) as c
ount FROM signals');
        return res.json({ incidentCount, signalCount });
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
router.get('/incidents/heatmap', authenticateJwt, requireRoles('admin', 'authority'), a
sync (_req, res) => {
    try {
        const [rows] = await getDbPool().query(
            'SELECT lat, lng, LEAST(5, GREATEST(1, severity)) as weight FROM in
cidents WHERE created_at > NOW() - INTERVAL 30 DAY'
        );
        return res.json(rows);
    } catch (e) {
        return res.status(500).json({ message: 'Server error' });
    }
});
module.exports = router;
```

server/src/routes/admin.js

```
const express = require('express');
const Joi = require('joi');
const { authenticateJwt, requireRoles } = require('../middleware/auth');
const { getDbPool } = require('../lib/db');
const router = express.Router();
router.use(authenticateJwt, requireRoles('admin'));
const signalSchema = Joi.object({
```

```
  name: Joi.string().required(),
  lat: Joi.number().required(),
  lng: Joi.number().required(),
  cycle_seconds: Joi.number().integer().min(20).max(300).default(90),
  green_seconds: Joi.number().integer().min(5).max(180).default(40),
  yellow_seconds: Joi.number().integer().min(3).max(15).default(5),
  red_seconds: Joi.number().integer().min(5).max(180).default(45),
});
router.post('/signals', async (req, res) => {
  const { error, value } = signalSchema.validate(req.body);
  if (error) return res.status(400).json({ message: error.message });
  try {
    const { name, lat, lng, cycle_seconds, green_seconds, yellow_seconds, red_se
conds } = value;
    const [result] = await getDbPool().query(
      `INSERT INTO signals (name, lat, lng, cycle_seconds, green_seconds, yell
ow_seconds, red_seconds, mode)
       VALUES (?, ?, ?, ?, ?, ?, ?, 'normal')`,
      [name, lat, lng, cycle_seconds, green_seconds, yellow_seconds, red_second
s]
    );
    return res.status(201).json({ id: result.insertId });
  } catch (e) {
    return res.status(500).json({ message: 'Server error' });
  }
});
router.get('/users', async (_req, res) => {
  try {
    const [rows] = await getDbPool().query('SELECT id, name, email, role, creat
ed_at FROM users ORDER BY created_at DESC');
    return res.json(rows);
  } catch (e);
  } catch (e) {
    return res.status(500).json({ message: 'Server error' });
  }
});
module.exports = router;
```

Socket.io

server/src/socket/attachIo.js

```
function attachIoToRequest(io) {
  return (req, _res, next) => {
    req.io = io;
```

```
        next();
    };
}
module.exports = { attachIoToRequest };
```

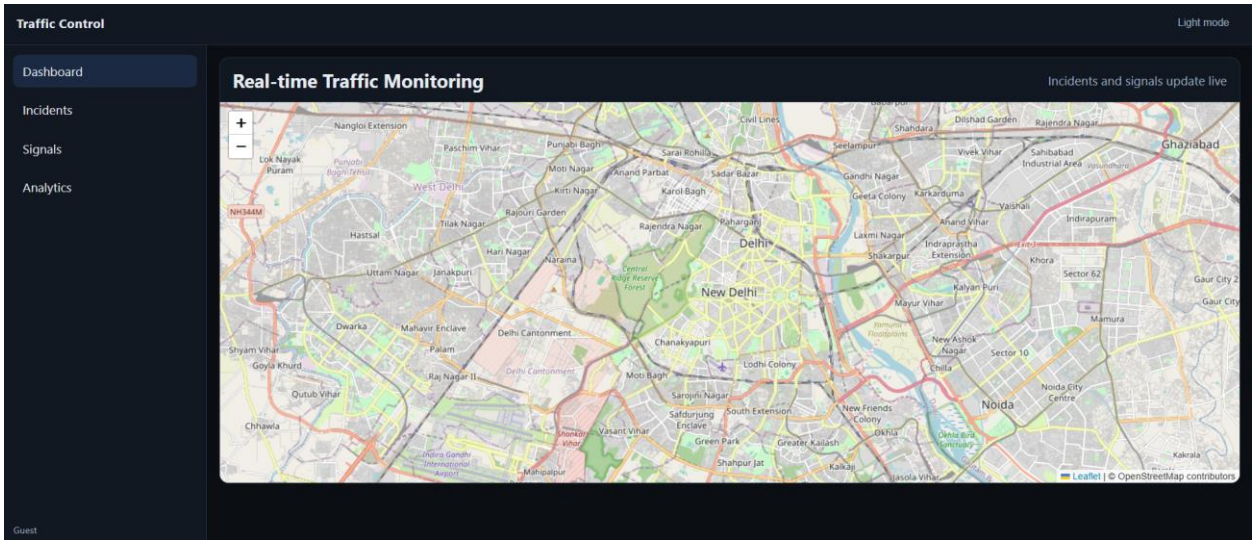
Environment variables

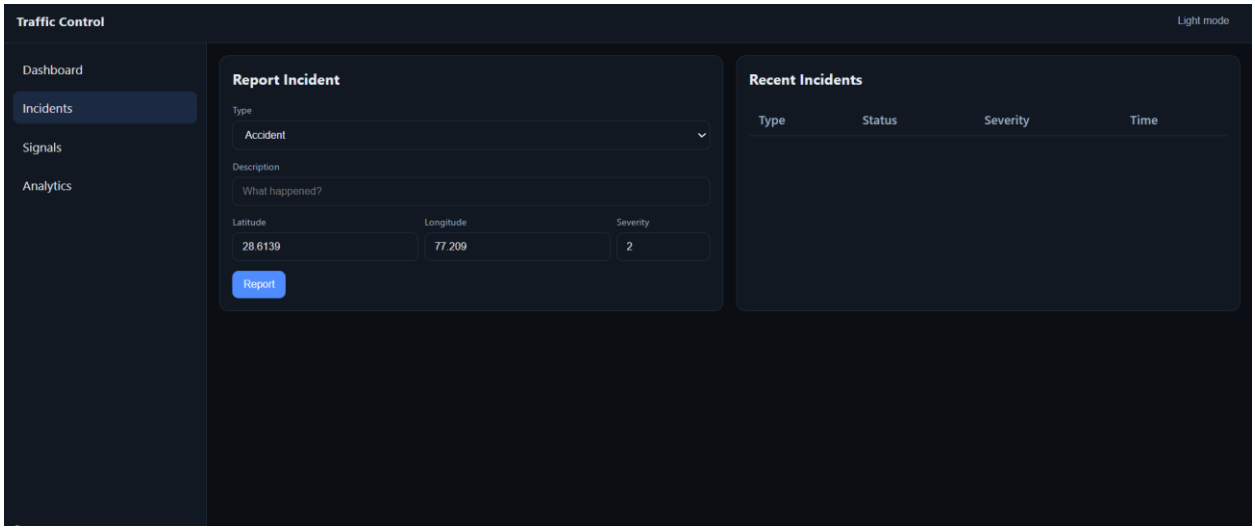
Create a .env file in the server directory with:

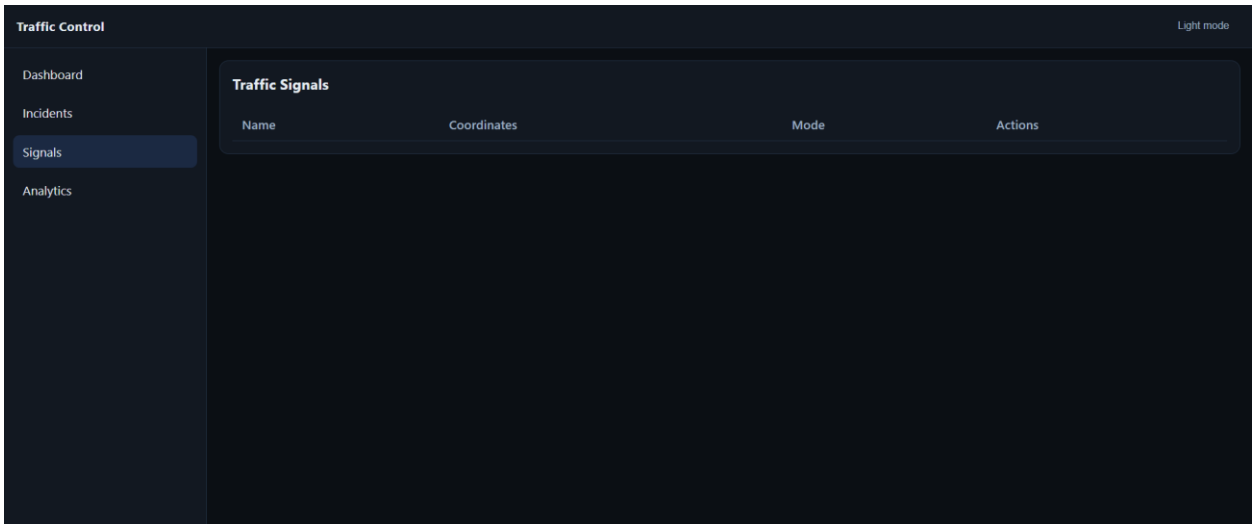
```
# Database
DB_HOST=localhost
DB_PORT=3306
DB_USER=root
DB_PASSWORD=your_password
DB_NAME=traffic_db
# Server
PORT=4000
CORS_ORIGIN=http://localhost:5173
# JWT
JWT_SECRET=your_secret_key_here
JWT_EXPIRES_IN=7d
# Optional
ENV_PATH=
```

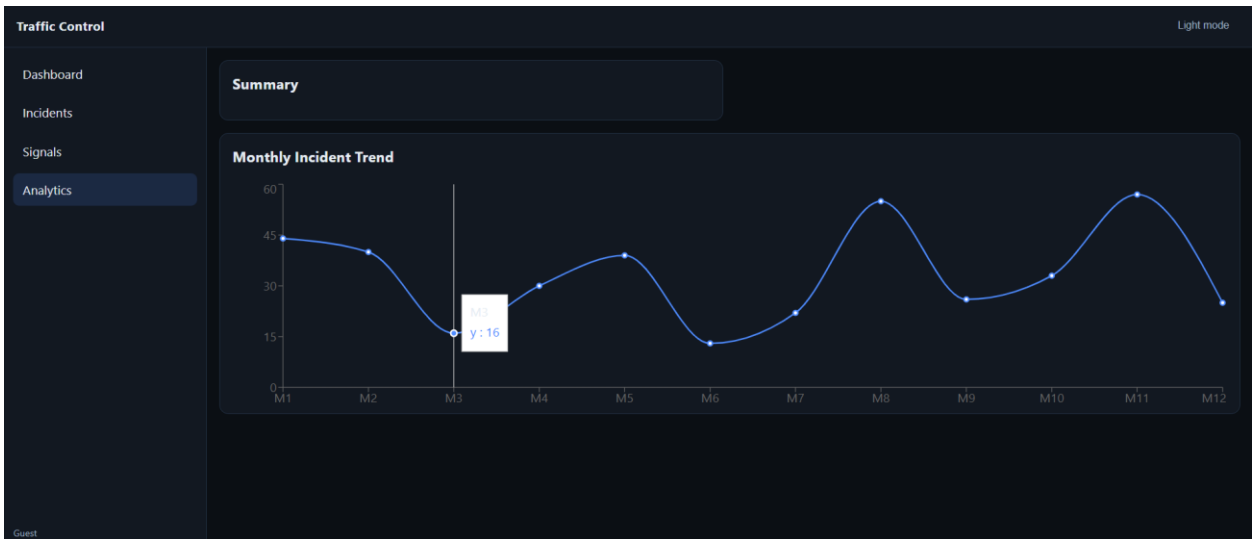
Signal Control Panel Authorized users (admin or authority) can view and modify the operational mode and timings of traffic signals from a dedicated page.

**









7. RESULT AND DISCUSSION

The project successfully resulted in a functional, full-stack web application for real-time traffic management. The system effectively demonstrates the power of a

modern technology stack to address complex urban challenges.
The three-tier architecture proved to be a sound design choice, providing a clear separation of concerns that simplifies development and enhances scalability. The backend, built with Node.js and Express, efficiently handles concurrent requests and business logic, while the React frontend offers a responsive and interactive user experience.
A key result of this project is the successful implementation of real-time data synchronization using Socket.IO. When an incident is reported or a signal is changed, the update is immediately reflected on all connected dashboards without requiring users to refresh the page. This is critical for a traffic management system where timely information is paramount for effective decision-making.
The use of a MySQL relational database is central to the system's reliability. The normalized schema with enforced referential integrity ensures that the data remains consistent and accurate. For example, the foreign key constraint between the Incidents and Users tables guarantees that every reported incident is linked to a valid user. Security has been addressed through robust password hashing using bcrypt and a granular Role-Based Access Control (RBAC) system. By embedding the user's role within the JWT, the API can effectively protect sensitive endpoints, ensuring that only users with the appropriate privileges (e.g., admin or authority) can perform critical actions like verifying an incident or modifying a signal's timing.
In discussion, while the system is currently reactive, the database schema, particularly the Congestion_Metrics table with its indexed timestamp, lays the groundwork for future predictive analytics. The collection of historical incident and traffic data is the first step toward building machine learning models that could proactively manage congestion.

8. CONCLUSION

This project successfully demonstrates the development of a comprehensive Smart City Traffic Management System. By integrating a React frontend, a Node.js backend, and a MySQL database, the application provides a robust platform for real-time traffic monitoring, incident management, and signal control. The architecture emphasizes scalability, security, and data integrity, making it a viable solution for modern urban environments.
The implementation of real-time updates via Socket.IO and a secure, role-based authentication system using JWT are key achievements that ensure the system is both responsive and secure. The relational database serves as a solid foundation, providing the transactional reliability necessary for a mission-critical application.
Looking forward, the system is well-positioned for future enhancements. The collected data can be leveraged to train machine learning models for predictive traffic analysis, transforming the system from a reactive to a proactive management tool. Further integration with public transportation APIs, IoT sensors,

and autonomous vehicle networks could expand its capabilities, contributing to a more efficient, safer, and more sustainable urban transportation ecosystem.

9. REFERENCES

MTC. (n.d.). *The Dangers of Traffic Congestion*. Metropolitan Transportation Commission. "Traffic congestion," *Wikipedia*. [Online]. Available: (https://en.wikipedia.org/wiki/Traffic_congestion). OperationsCommander. (n.d.). *Urbanization and Transportation: Navigating the Challenges*. M. E. Journal, "A Review of the Economic and Environmental Challenges of Traffic Congestion in Urban Areas," *Middle East University*, vol. 49, no. 2, 2023. C. Dong, B. N. H. Lee, and A. A. L. Teo, "Urban traffic congestion and its effects on the global economy and environment," *PLoS ONE*, vol. 11, no. 3, p. e0151229, Mar. 2016. P. S. S. Jr., "Challenges of the transportation professional," Purdue University, 1990. UCLA Institute of Transportation Studies. (n.d.). *Traffic Congestion*. UrbanSDK. (n.d.). *The Costs of Traffic Congestion*. Smat. (n.d.). *The Costs of Congestion: Economic Effects*. J. I. Levy, S. M. Chemerynski, and J. A. Sarnat, "The Public Health Costs of Traffic Congestion," IBTTA, 2010. S. T. Ariyawansa and T. M. V. P. Perera, "Economic impact of traffic congestion - estimation and challenges," in *2018 Moratuwa Engineering Research Conference (MERCon)*, 2018. H. C. Frey, N. M. Rouphail, and Z. Zhai, "Link-based emission factors for congested corridors," *Environ. Sci. Technol.*, vol. 42, no. 1, pp. 176-182, Jan. 2008. Omnisight. (n.d.). *Smart City Traffic Management: A Complete Guide*. Transline. (n.d.). *What are Smart Traffic Management Systems? A Comprehensive Guide*. Juniper Research. (n.d.). *How Smart Traffic Management Benefits Cities and Motorists*. Seagate. (n.d.). *How Smart Transportation Improves Traffic Management*. Vitronic. (n.d.). *Smart Cities Through Traffic Technology*. Miovision. (2024, June 14). *The Evolution of Traffic Signal Technology*. P. Kumar, A. Kumar, N. Kumar, and K. Kumar, "An Examination of Intelligent Transportation Systems in India," May 2024. M. A. A. Al-Absi *et al.*, "Intelligent Transportation Systems (ITS): A Review of the Historical Evolution and Future Directions," *Appl. Sci.*, vol. 14, no. 11, p. 4646, 2024. R. A. Gakenheimer, "Intelligent Transportation Systems," in *Encyclopedia of Life Support Systems (EOLSS)*. Eolss Publishers. A. Auer, S. Feese, and S. Lockwood, "History of Intelligent Transportation Systems," Booz Allen Hamilton, Rep. FHWA-JPO-16-329, 2016. U.S. Department of Transportation, "Information Security Analysis of Intelligent Transportation Systems," Rep. JPO-98-009, 1998. "Intelligent transportation system," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Intelligent_transportation_system. P. Patil. (2023, Dec 11). *Building a Three-Tier Application with Docker, Flask, Nginx, and Postgres*. [Online]. Available: https://medium.com/@prajwalpatil0402/building-a-three-tier-application-with-docker-flask-nginx-and-postgres-79bb5b2444d5. IBM. (n.d.). *What is three-tier architecture?*. vFunction. (n.d.). *What is a 3-Tier*

Application?. "Multitier architecture," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Multitier_architecture. B. K. Hebert. (2022, Jan 14). *Three-Tier Architecture Overview and Each Tier Explained*. [Online]. Available: https://dev.to/bkhebert/three-tier-architecture-overview-and-each-tier-explained-3j96. GeeksforGeeks. (n.d.). *Introduction of 3-Tier Architecture in DBMS (Set 2)*. Cloud Devs. (n.d.). *Why use TypeScript with React?*. A. Deving. (2023, Oct 25). *TypeScript with React: Building Type-Safe Frontend Applications*. [Online]. Available: https://alejodeving.medium.com/typescript-with-react-building-type-safe-frontend-applications-b4452bb2452c. Pulsion. (n.d.). *React with TypeScript vs. React with JavaScript: Which is Better?*. Women in Technology. (2023, Nov 27). *Mastering React with TypeScript: Its Benefits and Importance*. [Online]. Available: https://medium.com/womenintechnology/mastering-react-with-typescript-its-benefits-and-importance-85cbc783a85a. J. T. Nomad. (2019, Feb 27). *Why TypeScript is the best way to write Front-End in 2019*. [Online]. Available: https://jackthenomad.com/why-typescript-is-the-best-way-to-write-front-end-in-2019-feb855f9b164. Bacancy Technology. (n.d.). *Why Use React? [Entrepreneurs' Quest]*. Curotec. (n.d.). *Node.js vs. Express.js: What's the Difference?*. PureLogics. (n.d.). *Node.js vs. Express.js: What's the Difference and When to Use Each*. GeeksforGeeks. (n.d.). *The Pros and Cons of Node.js in Web Development*. Radixweb. (n.d.). *Node.js vs Express.js: A Detailed Comparison for 2024*. Fullestop. (n.d.). *Top Benefits of Using Node.js for Enterprise App Development*. Ace Infoway. (n.d.). *Why Node.js is the Best Choice for Building Real-time Applications*. Datamation. (n.d.). *8 Major Advantages of Using MySQL*. Talend. (n.d.). *What is MySQL?*. "Why is MySQL used so often in web development?," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/4779029/why-is-mysql-used-so-often-in-web-development. Sectorlink. (n.d.). *Why You Should Be Using MySQL*. GreenGeeks. (n.d.). *Understanding MySQL and Its Role in Web Design*. Socket.IO. (n.d.). *Introduction*. Authgear. (n.d.). *JWT Authentication: A Secure, Scalable Solution for Modern Applications*. SuperTokens. (n.d.). *What is JWT (JSON Web Token)?*. JWT.IO. (n.d.). *Introduction to JSON Web Tokens*. Descope. (n.d.). *What is a JWT?*. "Why and when should we use JSON Web Tokens?," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/46935135/why-and-when-should-we-use-json-web-tokens. "Why do many people use JWT instead of cookies for auth?," *Reddit*. [Online]. Available: https://www.reddit.com/r/learnprogramming/comments/17igrx8/why_do_many_people_use_jwt_instead_of_cookies_for/. Microsoft. (n.d.). *Tutorial: Add role-based access control to a Node.js web app*. F. Sonani. (n.d.). *Building Role-Based Access Control (RBAC) with JWT in Node.js*. R. Tamang. (2024, Mar 19). *How to Implement Role-Based Access Control (RBAC) in Node.js Applications*. [Online]. Available: https://dev.to/rabindratamang/how-to-implement-role-based-access-control-rbac-in-nodejs-applications-1ed2. T. Cabrel.

(n.d.). *How to Implement Role-Based Access Control in a Node.js API*. D. Ignatovich. (2023, Dec 12). *Implementing Role-Based Access Control (RBAC) in Node.js and React*. [Online].
Available: https://medium.com/@ignatovich.dm/implementing-role-based-access-control-rbac-in-node-js-and-react-c3d89af6f945. Stackademic. (2023, Dec 14). *Mastering Security: Role-Based Access Control in Node.js with JWT*. [Online]. Available: https://blog.stackademic.com/mastering-security-role-based-access-control-in-node-js-with-jwt-1d653f6e35dc. Google. (n.d.). *See your data in real time with Data-driven styling*. Google Maps Platform. (n.d.). *Visualize Data*. QSS Technosoft. (n.d.). *How to Use Google Maps in Leaflet*. Curate Partners. (n.d.). *Leaflet: The Open-Source JavaScript Library for Interactive and Customizable Web Maps*. J. Dougherty and I. M. Ilyankou. (n.d.). *Leaflet Maps with Google Sheets*. Google. (n.d.). *Visualize Your Data on a Custom Map Using Google My Maps*. RF Wireless World. (n.d.). *IoT based Intelligent Traffic System*. "IoT Based Smart Traffic Light Control System," *Scribd*. [Online]. Available:(https://www.scribd.com/document/867701946/IoT-Based-Smart-Traffic-Light-Control-System). M. H. Z. Abadi, S. Azizi, and M. Sheikhalishahi, "A Distributed Multi-Agent Reinforcement Learning Approach for Intelligent Traffic Signal Control," *Sensors*, vol. 5, no. 4, p. 66, 2022. S. S. Priyadharshini *et al.*, "IoT based Traffic Control System using LORA and AI," *International Journal of Next-Generation in Engineering and Technology*, 2024. S. R. Deore, "IoT Based Smart Traffic Signal Monitoring and Control System," *International Research Journal of Engineering and Technology (IRJET)*, vol. 7, no. 5, 2020. "What runs traffic signals at an intersection?," *Reddit*. [Online].
Available: https://www.reddit.com/r/ElectricalEngineering/comments/1j7mm7y/what_runs_traffic_signals_at_an_intersection/. U.S. Department of Transportation, "Verification Using Crowdsourcing," Rep. DOT-32726, 2016. U.S. Department of Transportation, "Crowdsourcing for Operations Applications," 2019. U.S. Department of Transportation. (n.d.). *Examples of Crowdsourcing for Advancing Operations*. A. Vinel, "Trustworthiness of Crowdsourced Data in VANETs," *Sensors*, vol. 19, no. 15, p. 3267, 2019. S. Kim, J. Kim, and W. Jung, "Integrating Crowdsourced Data for Enhanced Road Safety Analysis," *Sustainability*, vol. 16, no. 22, p. 9867, 2024. U.S. Department of Transportation. (2020, Feb 27). *Crowdsourcing for Operations*. D. Nwokike. (2024, May 22). *The best React chart libraries in 2025*. [Online].
Available: https://blog.logrocket.com/best-react-chart-libraries-2025/. Ably. (n.d.). *The top 11 React chart libraries to consider*. OpenReplay. (n.d.). *Top 10 React Chart Libraries in 2025*. Chart.js. (n.d.). *Simple yet flexible JavaScript charting for the modern web*. Microsoft. (n.d.). *Design a RESTful web API*. J. Lee. (2020, Mar 2). *Best practices for REST API design*. [Online].
Available: https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/. S. A. Rahman. (2024, Jan 1). *Mastering REST API Design: Essential Best Practices, Do's, and Don'ts for 2024*. [Online].
Available: https://medium.com/@syedabdullahrahman/mastering-rest-api-design-

essential-best-practices-dos-and-don-ts-for-2024-dd41a2c59133. Ambassador. (n.d.). *7 REST API Design Best Practices*. RESTful API Tutorial. (n.d.). *What is REST?*. GeeksforGeeks. (n.d.). *How to Design Database for Logistics and Transportation?*. "ERD Traffic Management [classic]," *Creately*. [Online]. Available: https://creately.com/diagram/example/ikft3unq1/erd-traffic-management-classic. L. Ragia and M. Deriaz, "The data model for the traffic management database," in *ResearchGate*, 2012. T. Arakawa, "Traffic Database Management System," in *VEHICULAR 2019: The Eighth International Conference on Advances in Vehicular Systems, Technologies and Applications*, 2019. R. Singh. (2023, Dec 13). *Building a Real-Time Traffic Monitoring System with PySpark and SQL*. [Online]. Available: https://ritikaxx.medium.com/building-a-real-time-traffic-monitoring-system-with-pyspark-and-sql-e0815e8e26b2. "What database should I use for traffic monitoring?," *Reddit*. [Online]. Available:(https://www.reddit.com/r/Database/comments/1iusf4m/what-database-should-i-use-for-traffic-monitoring/). Google Cloud. (n.d.). *Authenticating users with a JWT*. Microsoft. (n.d.). *Configure JWT Bearer authentication in ASP.NET Core*. Netguru. (n.d.). *How to Protect API Endpoints Using JWT*. Google Cloud. (n.d.). *Using a service account for authentication*. "Exclusive endpoint for authenticated user in a Rest API using JWT," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/45435998/exclusive-endpoint-for-authenticated-user-in-a-rest-api-using-jwt. GitHub. (n.d.). *Authenticating to the REST API*. Google Maps Platform. (n.d.). *Traffic options*. INRIX. (n.d.). *Real Time Traffic Data [Powered by Artificial Intelligence]*. ArcGIS. (n.d.). *Traffic service*. ArcGIS. (n.d.). *Traffic*. Mapbox. (n.d.). *Traffic Data*. Google Maps Platform. (n.d.). *Traffic Layer*. Catchpoint. (n.d.). *API Architecture*. "react-leaflet-typescript," *CodeSandbox*. [Online]. Available: https://codesandbox.io/s/react-leaflet-typescript-2x3o7. React Leaflet. (n.d.). *React Leaflet*. "@types/react-leaflet examples," *CodeSandbox*. [Online]. Available: https://codesandbox.io/examples/package/@types/react-leaflet. React Leaflet. (n.d.). *Installation*. Leaflet. (n.d.). *Quick Start Guide*. MapTiler. (n.d.). *How to use Leaflet with TypeScript*. "Traffic dashboard," *Shutterstock*. [Online]. Available: https://www.shutterstock.com/search/traffic-dashboard?image_type=illustration. "Traffic Management," *Dribbble*. [Online]. Available: https://dribbble.com/tags/traffic-management. Econolite. (n.d.). *Predictive Traffic Management Systems*. "Traffic dashboard Images," *Freepik*. [Online]. Available: https://www.freepik.com/free-photos-vectors/traffic-dashboard. S. Miah. (2025, Sep 1). *Kemetra -Traffic Management Dashboard*. [Online]. Available:(https://dribbble.com/shots/26469732-Kemetra-Traffic-Management-Dashboard). JMT Technology Group. (n.d.). *Traffic Management Dashboard*. Retool. (n.d.). *Incident Management and Reporting Template*. Glide. (n.d.). *Incident Reporting Template*. incident.io. (n.d.). *The all-in-one AI platform for on-call, incident response, and status pages*. "Incident Reporting," *Dribbble*. [Online]. Available: https://dribbble.com/tags/incident-reporting. Jotform. (n.d.). *Incident Reporting App*. R. Ramirez, "Design and

Development of an Incident Reporting Mobile Application," in *ResearchGate*, 2019. Omnisight. (n.d.). *Smart City Traffic Management: A Complete Guide*. GoodVision. (n.d.). *6 Smart Cities That Get Traffic Control Right*. Digi. (n.d.). *How IoT Can Monitor Traffic*. Blues. (n.d.). *Using Machine Learning at the Edge for Smart City Traffic Management*. S. Nandhini. (2024, Jan 10). *Predictive Analytics in Smart Traffic Management: A Case Study*. [Online]. Available: https://medium.com/@saroknandhini/predictive-analytics-in-smart-traffic-management-a-case-study-ee7bd57fce46. Smat. (n.d.). *Case Studies*. Middleware. (n.d.). *Node.js Performance Monitoring: A Complete Guide*. S. Sharma. (2024, May 22). *Profiling and Benchmarking Node.js Applications*. [Online]. Available: https://dev.to/imsushant12/profiling-and-benchmarking-nodejs-applications-2h2o. A. Kalik. (2024, May 20). *Performance Benchmark: Node.js vs Go*. [Online]. Available: https://itnext.io/performance-benchmark-node-js-vs-go-9dbad158c3b0. R. Salins. (2023, Dec 1). *Scaling Node.js Applications and Performance Benchmarks*. [Online]. Available: https://medium.com/@roysalins94/scaling-node-js-applications-and-performance-benchmarks-dfd156fcdd01. NearForm. (n.d.). *The State of Node.js Performance 2023*. NodeSource. (n.d.). *State of Node.js Performance 2024*. Socket.IO. (n.d.). *Scaling horizontally*. Socket.IO. (n.d.). *Performance tuning*. Socket.IO. (n.d.). *Using multiple nodes*. "scalability issues relating to socket.io," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/9924822/scalability-issues-relating-to-socket-io. Ably. (n.d.). *Scaling Socket.IO: The complete guide*. "Scale socket.io with nodejs for 4000 concurrent connections," *Reddit*. [Online]. Available: https://www.reddit.com/r/node/comments/orw2ej/scale_socketio_with_nodejs_for_4000_concurrent/. LogiNext. (n.d.). *What are Intelligent Transportation Systems (ITS)? A Comprehensive Overview*. M. A. A. Al-Absi *et al.*, "Intelligent Transportation Systems (ITS): A Review of the Historical Evolution and Future Directions," *Appl. Sci.*, vol. 14, no. 11, p. 4646, 2024. U.S. Government Accountability Office, "Intelligent Transportation Systems: Benefits and Challenges," Rep. GAO-23-105740, 2023. S. Narayanaswami, "Artificial intelligence in intelligent transportation systems: a review," *J. Intell. Manag. Sci. Eng.*, vol. 6, no. 1, p. 26, 2023. WeCloudData. (n.d.). *Navigating Your Way: Traffic Prediction with Machine Learning*. A. Kumar, "A Machine Learning Approach for Predictive Analysis of Traffic Flow," *ShodhKosh: Journal of Visual and Performing Arts*, 2024. M. A. A. Al-Absi *et al.*, "Machine Learning-Based Traffic Flow Prediction Using Multisource Data," *Future Internet*, vol. 10, no. 7, p. 155, 2024. F. A. D. D. Amato, G. D. D. Modica, and L. D. O. Tomarchio, "A Two-Level Machine Learning Approach for Traffic Flow Forecasting in Urban Scenarios without Traffic Sensors," *Electronics*, 2024. T. Younas. (2023, Nov 11). *The Role of Machine Learning in Predicting and Managing Traffic Patterns*. [Online]. Available: https://medium.com/@tehreemyounas/the-role-of-machine-learning-in-predicting-and-managing-traffic-patterns-8b8d036358c7. A. Haghighat *et al.*, "Predicting Road Speed using Machine Learning Techniques,"

in *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*, 2024. Highways News. (n.d.). *Q-Free makes dynamic signals secure API documentation available for integration*. Analytics India Magazine. (n.d.). *DMRC Signs MoU with Mappls MapmyIndia to Integrate Delhi Metro Data into Mappls App*. Cor-Sign. (n.d.). *Integrating Public Transportation with Traffic Management*. CGTN. (2025, Oct 31). *China's experience and global vision for people-centered smart cities*. [Online]. Available:((https://news.cgtn.com/news/2025-10-31/China-s-experience-and-global-vision-for-people-centered-smart-cities-1HVmhDHFs3e/p.html)). U.S. Department of Transportation. (n.d.). *Linking Planning and Operations*. Data-Smart City Solutions. (n.d.). *Leveraging Data Integration to Revolutionize Urban Transportation*. University of Canterbury. (n.d.). *IEEE Style*. Concordia University Library. (n.d.). *Citing - IEEE*. IJSSST. (n.d.). *IEEE Citation Style Guide*. Wordvice. (n.d.). *IEEE Citation Examples & Guidelines*. Scribbr. (n.d.). *IEEE Citation | Quick Guide & Examples*. IEEE Author Center. (n.d.). *IEEE Editorial Style Manual*.

1. INTRODUCTION

<Contents, Times New Roman 12, Line spacing 1.15>

7. REFERENCES

<Contents, Times New Roman 12, Line spacing 1.15>

<< IEEE, Harvard Format >>

1. Apruzzese, G., Laskov, P., Montes de Oca, E., Mallouli, W., Brdalo Rapa, L., Grammatopoulos, A.V. and Di Franco, F., 2023. The role of machine learning in cybersecurity. *Digital Threats: Research and Practice*, *4*(1), pp.1-38.
2. Dasgupta, D., Akhtar, Z. and Sen, S., 2022. Machine learning in cybersecurity: a comprehensive survey. *The Journal of Defense Modeling and Simulation*, *19*(1), pp.57-106.

APPENDIX A – Sample Code