A

Mini Project Report

On

SMART CITY INFRASTRUCTURE MONITORING USING IOT AND CLOUD SERVICES

Submitted to JNTU HYDERABAD

In Partial Fulfilment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

Submitted
By

P. Niharika	**(228R1A1243)**
K. Mallikarjun	**(228R1A1224)**
S. Almas	**(228R1A1253)**
K. Rithwik	**(228R1A1228)**

Under the Esteemed guidance of

Mr. K. ANIL

Assistant Professor, Department of IT

Department of Information Technology

CMR ENGINEERING COLLEGE
(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)

(Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401)

(2024-2025)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA , Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)

(Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401)

Department of Information Technology

CERTIFICATE

This is to certify that the project entitled **"SMART CITY INFRASTRUCTURE MONITORING USING IOT AND CLOUD SERVICES"** is a bonafide work carried out by

P. Niharika	**(228R1A1243)**
K. Mallikarjun	**(228R1A1224)**
S. Almas	**(228R1A1253)**
K. Rithwik	**(228R1A1228)**

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **INFORMATION TECHNOLOGY** from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision.

The results presented in this project have been verified and are found to be satisfactory. The results embodied in this project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	Head of the Department
Mr. K . ANIL	**Dr. MADHAVI PINGILI**
Assistant Professor	Professor & HOD
Department of IT	Department of IT
CMREC, Hyderabad	CMREC, Hyderabad

DECLARATION

This is to certify that the work reported in the present project entitled **"SMART CITY INFRASTRUCTURE MONITORING USING IOT AND CLOUD SERVICES"** is a record of bonafide work done by us in the Department of Information Technology, CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

P. Niharika	**(228R1A1243)**
K. Mallikarjun	**(228R1A1224)**
S. Almas	**(228R1A1253)**
K. Rithwik	**(228R1A1228)**

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, HOD, **Department of IT, CMR Engineering College** for their constant support.

We are extremely thankful to **Mr. K . ANIL,** Assistant Professor, Internal Guide, Department of IT, for his constant guidance, encouragement, and moral support throughout the project.

We will be failing in duty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We express our thanks to all staff members and friends for all the help and coordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided us through every step.

P. Niharika	**(228R1A1243)**
K. Mallikarjun	**(228R1A1224)**
S. Almas	**(228R1A1253)**
K. Rithwik	**(228R1A1228)**

CONTENTS

ABSTRACT

This project focuses on developing a smart city traffic monitoring system that leverages AWS services to enable real-time environmental and traffic analysis. IoT sensors upload air quality index (AQI), noise level, and humidity data to Amazon S3 , where it is processed by an AWS Lambda function that evaluates environmental thresholds and dynamically updates traffic signals. When hazardous conditions are detected, Amazon SNS triggers SMS/email alerts to relevant authorities and stakeholders. The system integrates with a web-based dashboard that visualizes junction signals, vehicle movement, and route planning based on current AQI levels, allowing users to monitor traffic flow and environmental health in urban areas. The AQI values play a key role in deciding how traffic should be managed in different pollution conditions. For instance, if the air quality is good, a green traffic signal is shown; if the air quality is moderate, a yellow signal is displayed; and if the air is heavily polluted, the system switches the dashboard to display a red signal. These signals are not controlling real-world traffic lights but are instead visually represented on an interactive dashboard, which reflects live pollution data and the corresponding traffic signal in real time. The dashboard provides a user-friendly interface that helps users or city officials easily interpret current air conditions and understand how traffic would ideally be managed in response. This solution demonstrates how cloud-integrated IoT systems can be used not just for environmental monitoring, but also for visual decision-making in urban infrastructure. By combining sensor-based pollution detection with cloud storage and serverless processing, the project ensures a scalable, cost-effective, and real-time approach to smart city development. It serves as a foundational model for future smart systems where environmental data can directly influence urban operations like traffic flow management, ultimately contributing to cleaner air and more efficient city planning.

Keywords: Smart City, IoT, Air Quality Monitoring, AQI, MQTT, AWS S3, AWS Lambda, Real-Time Data, Traffic Signal Control, Cloud Computing, Environmental Monitoring, Pollution Detection.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction

As urban populations rapidly increase, cities are facing escalating challenges in managing traffic congestion, environmental degradation, and public health risks. Traditional traffic management systems often rely on fixed signal patterns and manual intervention, which are not capable of adapting to real-time environmental or traffic conditions. This leads to higher travel times, fuel wastage, and increased pollution levels. Factors like Air Quality Index (AQI)**,** noise pollution, and humidity play a critical role in urban sustainability, yet they are often overlooked in existing traffic infrastructure.

To overcome these limitations, modern cities are moving toward smart infrastructure using Internet of Things (IoT) devices and cloud-based platforms. Real-time monitoring of environmental parameters and automated decision-making is essential for responsive urban governance. According to the World Health Organization, over 4.2 million deaths each year are linked to poor air quality, while Indian cities alone lose over ₹60,000 crores annually due to traffic inefficiencies and pollution. These statistics highlight the urgency of adopting intelligent, adaptive systems.

Smart City Infrastructure Monitoring and Traffic Control System that leverages IoT-enabled data collection and AWS cloud services to create a scalable, responsive, and environmentally aware traffic management system. Using simulated IoT sensor data, the system monitors six city junctions and collects real-time data on AQI, noise levels, and humidity. This data is stored in Amazon S3, processed via AWS Lambda, and alerts are issued using Amazon SNS when pollution thresholds are exceeded.

The architecture is serverless and event-driven. When a new sensor dataset (in .json format) is uploaded to S3, it automatically triggers a Lambda function that processes the data. If hazardous AQI or noise levels are detected, SNS sends real-time email alerts to authorities or system users. Additionally, the dashboard uses API Gateway and Lambda to fetch and display live environmental data, adapting traffic signals dynamically based on pollution levels.

A web-based dashboard built using HTML, CSS, and JavaScript visualizes all junction data. It dynamically displays AQI signal colors (green/yellow/red), supports null or missing values, and rotates between junctions to provide real-time updates.The use of event-driven functions helps in responding immediately to environmental changes, while S3 triggers and API Gateway provide seamless integration between backend processing and frontend display. This architecture also keeps the system highly available, modular, and easy to maintain.

This project demonstrates a complete, cloud-based, and low-cost prototype for real-time traffic and environmental management — all within the AWS Free Tier. It serves as a foundation for future enhancements like real-time sensor integration, predictive analytics, and public health alert systems.

The project is designed using a fully serverless architecture, enabling it to scale automatically without requiring manual server provisioning. Services like AWS Lambda, Amazon S3, API Gateway, and SNS are part of the Platform as a Service (PaaS) model, allowing rapid development and cost-efficient deployment. The use of event-driven functions helps in responding immediately to environmental changes, while S3 triggers and API Gateway provide seamless integration between backend processing and frontend display. This architecture also keeps the system highly available, modular, and easy to maintain.

1.2 Project Objectives

The primary aim of this project is to design and implement an intelligent traffic monitoring system for smart cities using IoT and AWS technologies. Deploy IoT sensors at strategic traffic junctions to collect continuous data on air quality index (AQI), noise levels, humidity, and temperature. Use Amazon S3 for secure and scalable storage of sensor data. Implement AWS Lambda functions to automatically process incoming data, evaluate environmental thresholds, and take necessary actions without human intervention. The use of event-driven functions helps in responding immediately to environmental changes, while S3 triggers and API Gateway provide seamless integration between backend processing and frontend display.

1.3 Purpose of the Project

The purpose of this project is to develop an intelligent, automated system that enhances urban traffic management and environmental monitoring by leveraging IoT technology and cloud-based services from Amazon Web Services (AWS). In modern cities, increasing traffic congestion, air pollution, and noise levels have become major concerns affecting public health, quality of life, and overall urban sustainability. Implement AWS Lambda functions to automatically process incoming data, evaluate environmental thresholds, and take necessary actions without human intervention. Integrating real-time data collection from IoT sensors to monitor key environmental parameters such as air quality, noise, humidity, and temperature. Enabling smart decision-making through automated analysis and response using serverless computing (AWS Lambda), without the need for constant human intervention.

1.4 Existing System with Disadvantages

Smart city infrastructure monitoring is essential for maintaining urban safety, sustainability, and efficiency. However, most existing infrastructure monitoring systems in traditional cities are limited in scope, outdated in technology, and often lack integration with modern cloud and IoT solutions. Data is often used only for reporting purposes and not for predictive analysis or real-time decision-making due to the absence of cloud-based processing and analytics.These systems primarily rely on manual inspections, isolated monitoring tools, and non-scalable infrastructures, which are not suitable for handling the dynamic needs of modern urban environments.Manual or periodic monitoring of infrastructure (e.g., air quality, water levels, waste management, traffic signals).Use of localized, non-networked sensors or devices. Centralized control centers with minimal automation. Data stored in on-premise servers with limited accessibility and processing power.

Disadvantages of the Existing System:

- **Lack of Real-Time Data Collection:** Infrastructure metrics (e.g., pollution levels, traffic congestion, or structural health) are not collected in real time, leading to delayed detection of faults or hazards.
- **High Dependence on Manual Monitoring:** Infrastructure inspections and environmental readings are often conducted manually, making them time-consuming, labor-intensive, and prone to human error.
- **Poor Integration and Scalability:** Existing systems lack integration across different urban domains (traffic, pollution, public safety), and scaling the system to cover more regions or parameters is expensive and inefficient.
- **Limited Analytical Capabilities:** Data is often used only for reporting purposes and not for predictive analysis or real-time decision-making due to the absence of cloud-based processing and analytics.
- **Inadequate Alert Mechanisms:** Alerts for critical infrastructure issues (e.g., excessive pollution, noise, or structural faults) are either delayed or not automated, increasing risk to public health and safety.
- **Restricted Accessibility:** Data is usually confined to government offices or control centers, limiting transparency and public awareness. Citizens and stakeholders cannot easily access or respond to live city conditions.
- **High Maintenance and Operational Costs:** On-premise systems require regular maintenance, upgrades, and manual oversight, leading to higher costs over time compared to cloud-based, automated alternatives.

1.5 Proposed System with Feature

To overcome the limitations of traditional infrastructure monitoring systems, the proposed solution leverages IoT (Internet of Things) devices in combination with cloud computing services, specifically Amazon Web Services (AWS), to build an intelligent, real-time, scalable, and automated Smart City Infrastructure Monitoring System. Serverless computing with AWS Lambda enables automatic processing and threshold evaluation of sensor data without the need for dedicated servers or manual intervention.When thresholds (e.g., hazardous AQI or noise levels) are exceeded, Amazon SNS (Simple Notification Service) triggers automated alerts via SMS or email to city officials, emergency responders, and other stakeholders. This system aims to enhance the monitoring, analysis, and management of urban infrastructure such as air quality, traffic, noise levels, humidity, waste, and water systems through connected sensors and cloud-based processing.

- **Real-Time Monitoring with IoT Sensors**: Smart sensors are deployed across the city to collect live data on air quality (AQI), noise pollution, humidity, temperature, traffic density, and other infrastructure parameters.
- **Cloud-Based Data Storage and Processing:** Sensor data is transmitted to Amazon S3, where it is securely stored and made available for both real-time and historical analysis.
- **Automated Data Analysis using AWS Lambda:** Serverless computing with AWS Lambda enables automatic processing and threshold evaluation of sensor data without the need for dedicated servers or manual intervention.
- **Smart Alerting System via Amazon SNS:** When thresholds (e.g., hazardous AQI or noise levels) are exceeded, Amazon SNS (Simple Notification Service) triggers automated alerts via SMS or email to city officials, emergency responders, and other stakeholders.
- **Dynamic Traffic Signal Adjustment (if applicable):** The system can adjust traffic signal patterns dynamically based on real-time traffic data and pollution levels to reduce congestion and emissions.
- **Interactive Web-Based Dashboard:** A centralized dashboard visualizes data in real time, showing sensor readings, traffic flows, environmental health indicators, and alerts. It is accessible to both administrators and citizens. system can easily scale to accommodate more sensors.
- **Scalable and Cost-Effective Infrastructure:** By using AWS services such as Lambda, S3, SNS, and DynamoDB, the system can easily scale to accommodate more sensors and additional smart city features without major infrastructure changes.

Advantages:

Each junction provides environmental data points such as the Air Quality Index (AQI), noise level (in decibels), and humidity (percentage). It also provide optimal routes to travel from one junction to other junction. Traffic signals changes dynamically when AQI values changes also visualizes data in real time, showing sensor readings, traffic flows, environmental health indicators, and alerts. It is accessible to both administrators and citizens. The system can easily scale to accommodate more sensors and additional smart city features without major infrastructure changes.

- **Real-Time Monitoring and Decision Making:**

  -Ensures quick response to faults, environmental hazards, or abnormal conditions through continuous data streaming and automated logic.
- **Scalability and Flexibility:**

  -Cloud-native architecture allows easy scaling of the system across more locations and supports integration of additional sensors or data types.
- **Automation and Efficiency**:

  -Reduces reliance on manual operations and improves efficiency through serverless automation and smart alerts.
- **Improved Urban Safety and Sustainability:**
  - Enables proactive environmental and infrastructure management to ensure public safety and reduce pollution and energy waste.
- **Cost-Effective Maintenance:**
  - AWS pay-as-you-go services reduce capital expenditures on hardware and IT infrastructure while improving operational efficiency.
- **Transparency and Public Engagement:**
  - The system's dashboard and alerts can inform and engage citizens, promoting more responsible behavior and transparency.
- **Predictive and Preventive Maintenance:**
  - Historical data analytics help predict equipment or infrastructure failures, allowing for scheduled preventive maintenance.
- **Data-Driven Governance:**

  The collected datasets can be analyzed to support policy-making, such as noise zoning regulatios, green space development, or optimized traffic signal timing strategies.

1.6 Input and Output Design

Input Design

The input design of this project focuses on how environmental sensor data is collected and structured before being processed by the system. In this project, the data is simulated and formatted as a JSON file that represents readings from six different traffic junctions. Each junction provides environmental data points such as the Air Quality Index (AQI), noise level (in decibels), and humidity (percentage). These datasets are uploaded to an Amazon S3 bucket in real time or at set intervals. The system is designed to accept partial or incomplete data — if values such as noise or humidity are missing or set as null, the system still processes the file without error. The use of the `.json` format ensures consistency, machine-readability, and easy integration with AWS Lambda functions. Each upload event acts as a trigger, launching the backend processing pipeline without the need for manual intervention.

Objectives

The primary objective of the input design is to ensure that the system receives structured, reliable, and readable environmental data from various traffic junctions. Since the system relies on sensor-based inputs, it is crucial that the format remains consistent and the data is organized in a way that can be easily parsed and validated. By using `.json` as the data format, the design ensures that the structure remains both human-readable and machine-processable.

Another key objective is to support real-time and automated data ingestion. The system is built to respond instantly to new data uploads in Amazon S3 using event-based triggers. This eliminates the need for manual intervention or periodic polling, making the solution scalable and efficient for live environments. As soon as new data is uploaded, a Lambda function is automatically triggered, starting the processing and alerting workflow.

The input design also aims to provide tolerance for incomplete or noisy data. In real-world applications, IoT data can be inconsistent due to network issues, sensor failure, or power interruptions. The system is therefore designed to accept `null`, negative, or missing values without crashing or rejecting the entire dataset. Instead, it processes the available data while safely ignoring or flagging the missing parts. Finally, the design encourages simplicity and ease of use input process remains straightforward..

Output Design

The output design defines how the system displays processed data and communicates critical alerts. The primary output is a web-based dashboard created using HTML, CSS, and JavaScript. This dashboard dynamically fetches data through AWS API Gateway, which connects to a Lambda function that retrieves the latest sensor data from S3. The dashboard displays each junction's AQI, noise level, and humidity values in real time and uses colored traffic signals to visually represent the severity of the AQI — green for good, yellow for moderate, and red for hazardous levels. The dashboard also rotates through each junction at fixed intervals, ensuring up-to-date monitoring. In addition to visual output, the system sends alerts through Amazon SNS when AQI exceeds 300 or noise surpasses 80 dB. These alerts are delivered to authorized recipients via email, enabling quick action when environmental conditions become critical.

Objectives

The primary objective of the output design is to deliver a comprehensive, clear, and intuitive presentation of complex environmental data, enabling stakeholders to make timely and informed decisions with ease. This dashboard serves as an interactive and real-time visualization platform, consolidating multiple critical environmental metrics temperature, and humidity—across six strategically important city junctions.

To enhance user experience and accessibility, the dashboard employs easily recognizable visual indicators like colored traffic light signals (green, yellow, red) to represent varying pollution severity levels at each junction. This approach ensures that even users without technical expertise can instantly understand the environmental status and respond accordingly. Real-time data updates guarantee that the dashboard reflects current conditions, providing a dynamic overview that supports rapid situational awareness and decision-making.

Another crucial objective is to implement an automated alerting mechanism that activates when environmental metrics exceed predefined safety thresholds. Leveraging Amazon Simple Notification Service (SNS), the system promptly sends notifications to designated decision-makers, environmental agencies, and emergency responders. This immediate alerting capability is vital to reducing the lag between detection and action, thereby minimizing public health risks and enabling swift intervention measures. The dashboard is engineered to gracefully handle incomplete or partial data sets without causing system errors or misleading representation.

2. LITERATURE SURVEY

Jeremy Reed; Ali Tosun; Turgay Korkmaz; Benjamin Topolosky. " IoT Device Classification in Edge-Cloud Models" Publisher: IEEE 20 May 2025. The paper investigates security vulnerabilities in edge-cloud-based IoT systems. They propose a novel side-channel attack that uses DNS request patterns and AWS cloud endpoints to classify IoT devices, even when data is encrypted. This technique reveals device roles within a network and exposes systems to physical and cyber threats, emphasizing the need for robust security mechanisms in IoT-cloud architectures. The study demonstrates how attackers can exploit non-payload metadata to extract sensitive information. It underscores the limitations of relying solely on encryption for IoT security. The findings call for improved traffic anonymization and privacy-aware network designs. The proposed solution uses live data to dynamically reroute traffic, minimizing idle time and fuel usage.

S Shahul Hammed; C. Preethi; K Haripriya; S. Pavalarajan; M.R. Gowtham; N. Akshaay Krithick. "Traffic and Pollution Control Using IoT-Enabled Smart Routing Algorithm". Publisher: IEEE 25 April 2025. This paper highlights address urban congestion and pollution by introducing a smart routing algorithm powered by IoT-enabled real-time data. The system collects traffic, environmental, and driving pattern data to make eco-friendly decisions. The paper also discusses deployment challenges such as scalability, data privacy, and security, highlighting the need for sustainable urban planning with IoT integration. The proposed solution uses live data to dynamically reroute traffic, minimizing idle time and fuel usage. It supports smart city goals by improving both environmental health and transportation efficiency. The authors recommend further research into data fusion and cloud-based decision-making platforms.

P. Chinnasamy; Patan Feridoz Khan; C Sasikala; Mavalluru Swathi; M Prema Kumar; Ajay Kumar. "Industrial IoT Smart Manufacturing Systems based on Cloud Computing Energy Efficiency Analysis".Publisher: IEEE 02 April 2025. The paper proposes an IoT-based smart manufacturing system using cloud computing and machine learning to improve energy efficiency. explore how cloud-based IoT and machine learning can enhance energy efficiency in industrial environments. The proposed RMCLR algorithm analyzes energy usage and optimizes processes in real-time, demonstrating significant performance improvements. This approach supports real-time monitoring and predictive adjustments in manufacturing workflows. The paper highlights how Industry 4.0 technologies can reduce operational costs and carbon footprints. Results show improved productivity and energy savings across multiple test scenarios.

Zeinab Nazemi Absardi ; Reza Javidan."A QoS-Aware Traffic Management Policy for IoT-enabled Smart City Applications based on Cloud Computing". Publisher: IEEE 21 November 2025: This paper introduces a QoS-aware traffic management policy for smart cities using edge and cloud computing. The propose a traffic management system that considers Quality of Service (QoS) parameters such as delay, energy consumption, and data throughput. By leveraging both edge and cloud computing, the system handles massive IoT data more efficiently. The approach significantly reduces energy usage and processing time, improving responsiveness in smart city environments. The policy supports critical applications like emergency response and congestion prediction.

S. Arul; P. Kavitha; S. Kamalakkannan. "Innovative Solutions for Sustainable IoT in Smart City Infrastructure".Publisher: IEEE 24 October 2024. The paper discuss how IoT-based video surveillance can improve safety and public services. The paper integrates technologies such as deep learning, blockchain, and edge-cloud models to enhance infrastructure monitoring. The system supports applications like traffic control, crime prevention, and emergency detection. Blockchain ensures data integrity and security, while deep learning boosts real-time object recognition accuracy. The approach promotes sustainability by optimizing resource use and minimizing human intervention.

Ahgaly Subbiah. "AI-Enhanced IoT System for Efficient Traffic Management: Leveraging Black Data in Smart Cities". Publisher: IEEE 14 February 2025: This paper presents an intelligent system that uses unused "black data" from speed cameras and sensors. By integrating Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs), the system processes data at the edge, reducing cloud burden. It filters traffic data to prioritize relevant insights, enabling real-time traffic optimization. The use of edge AI accelerates decision-making and enhances scalability. The paper shows improved traffic prediction and congestion handling capabilities.

Kariuki Pius Muriuki; Japheth Odiwour Okello; Joan Chepkoech. "Advanced Intelligent Traffic Management System(AITMS)". Publisher:IEEE 28 November 2024 .The AITMS is a smart traffic system using Generative AI and IoT to manage urban congestion. It design a smart traffic system powered by Generative AI and IoT. The system predicts traffic flow, adjusts signal timings, and provides real-time driver alerts to reduce congestion. It continuously learns from historical and live data, improving over time. Generative AI is used to simulate and evaluate traffic scenarios, enhancing adaptability.

A.C. Santha Sheela ; B.Shamreen Ahamed; D. Poornima; Charles Sagayaraj. "Integration of Wireless Sensor Networks with IoT in Smart Transportation Systems and Traffic Management".

IEEE 21 February 2024. This paper focus on combining Wireless Sensor Networks (WSNs) with IoT to improve traffic monitoring and urban mobility. The system collects real-time data on vehicle flow, road usage, and environmental conditions. It promotes eco-friendly transport by informing city planners and enabling smart route decisions. The authors address challenges in data accuracy, network security, and system scalability. The architecture is suitable for both small-scale trials and full-city deployments.

Vijay Gurbade; Prateek Verma; Swapnil Gundewar; Vijendra Singh Bramhe "Building a Data Lake for Power BI in the Cloud: A Review on Utilizing Cloud Storage Services for Large Datasets" Publisher: IEEE 20 January 2025. The paper explores the use of cloud-based data lakes integrated with Power BI to manage and analyze large, diverse datasets efficiently. Unlike traditional data warehouses, data lakes store raw, structured, and unstructured data without requiring a fixed schema. Cloud platforms like Azure Data Lake Storage and Amazon S3 offer scalability, cost-effectiveness, and robust security for handling massive data volumes. Integrating Power BI with cloud data lakes enables real-time analytics, advanced visualizations, and self-service business intelligence.

Anurag Pal; Prince Jay Shankar; Ananya Das; P Yellamma. Efficient Data Storage Algorithm for IoT in Cloud-Distributed Environments. Publisher: IEEE 11 April 2024. It introduces the use of the Hadoop Distributed File System (HDFS) to optimize data access and storage within cloud-based IoT frameworks. By leveraging hash values, the method enhances metadata configuration and distribution, leading to better data block allocation. The approach is validated through simulations and real-world testing, showing improvements in efficiency, reduced latency, and system reliability. Security features like encryption, access control, and authentication are also integrated. Overall, the study demonstrates a robust and scalable solution for modern IoT-cloud ecosystems.

Anurag Pal; Prince Jay Shankar; Ananya Das; P Yellamma. Efficient Data Storage Algorithm for IoT in Cloud-Distributed Environments. Publisher: IEEE 02 April 2025. This paper proposes an AI-enabled Multi-Factor Authentication (MFA) system to enhance security in both private and public cloud environments. Traditional MFA approaches are no longer sufficient against modern cyber threats, making AI integration essential. The framework utilizes behavioral analysis, anomaly detection, and contextual data to evaluate authentication attempts dynamically.

3. SOFTWARE REQUREIMENTS ANALYSIS

3.1 Problem Statement

This project aims to develop a Smart City Infrastructure Monitoring and Traffic Control system that uses IoT sensor data to dynamically manage traffic signals, monitor environmental conditions, and alert authorities about hazardous pollution or noise levels, thus enhancing city livability and traffic efficiency.
Real-time monitoring and control of traffic and environmental parameters such as Air Quality Index (AQI), noise, humidity, and traffic flow are crucial for effective city management. Manual monitoring is inefficient and prone to delays, causing slower responses to pollution spikes or traffic jams.

3.2 Modules and Their Functionalities

The Smart City Infrastructure Monitoring System is divided into several key modules, each responsible for specific functionalities. These modules work together to ensure efficient data collection, processing, alerting.

1. IoT Sensor Data Collection Module

The IoT Sensor Data Collection Module is responsible for capturing accurate and real-time environmental and traffic-related data across the city. Sensor nodes equipped with a variety of devices such as air quality sensors (for example, MQ-135 or PMS5003), noise sensors with microphones to measure sound levels, humidity and temperature sensors like the DHT22, and traffic flow sensors such as infrared or ultrasonic vehicle counters are strategically deployed at key junctions and roads. These sensors continuously monitor parameters like AQI, noise pollution, humidity, and traffic density. Alternatively, sensors can send HTTP POST requests directly to an API Gateway endpoint. Each sensor reading includes essential metadata such as timestamps, sensor ID, and geo location to enable precise tracking and analysis.

2. Data Ingestion and Processing Module

The Data Ingestion and Processing Module forms the backbone of the backend system, responsible for receiving, validating, and storing incoming data streams. AWS API Gateway acts as a secure RESTful interface that accepts data sent either from the IoT sensors or the frontend applications. It also enforces authentication and throttling policies to maintain security and service stability. Upon receiving new data, AWS Lambda functions are triggered to process the information. Additionally, the Lambda functions invoke alert checks to detect any threshold breaches. Processed data is stored in two main repositories: Amazon DynamoDB holds the most recent sensor readings to enable rapid queries and support real-time dashboard .

3. Alert Generation Module

The Alert Generation Module continuously monitors incoming sensor data to identify any violations of predefined environmental or traffic thresholds. Lambda functions implement this monitoring by comparing real-time readings against established limits—for instance, an AQI value exceeding 150 or noise levels rising above 70 decibels. When such thresholds are crossed, Amazon Simple Notification Service (SNS) is used to automatically send alerts to subscribed stakeholders via email or SMS, ensuring that both city authorities and citizens receive timely warnings about hazardous conditions.

4. Data Visualization and Control Module

The Data Visualization and Control Module encompasses the frontend components that provide an interactive and user-friendly interface for monitoring and managing city infrastructure. Developed using React.js, the web application enables real-time UI updates and dynamic interactions. It presents an interactive city map overlay showing sensor locations and their latest readings, current traffic light statuses, and animated vehicles that visualize traffic flow intensity and congestion levels. Users can filter sensors by type, location, or alert status to focus on areas of interest and analyze historical data through charts and trend graphs.

3.3 Functional Requirements

The Smart City Monitoring System must reliably collect data from IoT sensor nodes deployed across various traffic junctions. These sensors capture real-time data such as air quality index (AQI), humidity, noise levels, and traffic flow. The data is transmitted at configurable intervals, typically every 10 seconds, and includes metadata such as sensor ID, location, and timestamps. Communication is secured via MQTT with TLS encryption or HTTPS POST to AWS API Gateway. Upon receiving data, AWS Lambda functions process and clean the data, check for anomalies, and store it in DynamoDB for live querying and S3 for historical storage. If a threshold is exceeded, such as high pollution or excessive noise, the system generates an alert through Amazon SNS and notifies users via email or SMS. The frontend dashboard, built using React.js, updates every few seconds to reflect live data, visualizing sensor statuses, traffic lights, and vehicle movement. General users can monitor conditions, while city officials can control traffic signals and view detailed reports.

Sensor Nodes: Collect AQI, noise, humidity, and traffic data in real time.

Transmission: Sends data via MQTT (AWS IoT Core) or HTTP POST (API Gateway).

Security: Uses TLS encryption and authentication tokens.
API Gateway: Receives incoming sensor data and routes to AWS Lambda.
AWS Lambda:Validates and filters data, Detects anomalies and triggers alerts,Formats and stores data.
Amazon DynamoDB: Stores current sensor readings for quick frontend access.
Amazon S3: Archives data logs in structured formats (JSON/CSV).
Amazon SNS: Sends alert messages with sensor type, value, and location.

3.4 Non-Functional Requirements

The system must support high scalability to handle hundreds or even thousands of sensor nodes without performance bottlenecks. AWS services such as Lambda and DynamoDB must auto-scale as the sensor network grows. The system should remain highly available with minimal downtime, using retry mechanisms to handle intermittent data transmission failures from sensors. Cloud services with at least 99.9% uptime ensure reliability. Security is critical, with all data transmissions encrypted and strict IAM roles used for access control. Stored data in DynamoDB and S3 must also be encrypted. The system should offer low-latency performance, ensuring that live sensor data appears on the dashboard within 2 seconds and alerts are delivered within 1 minute. Maintainability is ensured through modular development, clear APIs, and monitoring via AWS CloudWatch. The frontend should be responsive and intuitive, accessible from both desktop and mobile devices. Lastly, the solution should be cloud-native and portable, allowing easy replication or migration to other cities or cloud regions.

Scalability: Supports 100s to 1000s of sensors., Lambda and DynamoDB auto-scale with demand.
Reliability & Availability: Retry logic on sensor data failures and also uses AWS services with 99.9% uptime SLAs.
Security: TLS-secured MQTT/HTTPS communications, and also IAM policies restrict resource access.
Performance: Dashboard refresh latency < 2 seconds,Alerts delivered via SNS within 60 seconds.
Maintainability: Code modularity with serverless functions and Monitoring and logs via AWS CloudWatch.
Usability: Frontend responsive across devices.

3.5 Feasibility Study

The feasibility study for the web-based traffic control assesses the viability of the project from various perspectives to ensure that it can be successfully developed and implemented. The study covers three main areas: technical feasibility, operational feasibility, and economic feasibility.

Technical Feasibility

The proposed system is technically feasible due to the use of reliable and well-documented AWS services such as AWS IoT Core, Lambda, API Gateway, DynamoDB, and S3. These cloud-native services ensure scalability, flexibility, and seamless integration. On the frontend, React.js combined with modern JavaScript libraries enables the creation of a responsive and dynamic user interface that can effectively visualize real-time sensor data. The hardware components, including air quality sensors, noise detectors, humidity sensors, and traffic counters, are widely available and compatible with standard communication protocols like MQTT and HTTP, making integration straightforward.

Economic Feasibility

Economically, the project is feasible and cost-effective, particularly in the early stages, as most of the AWS services used fall under the AWS Free Tier, significantly reducing infrastructure costs during prototyping. The use of open-source technologies like React.js and supporting JavaScript libraries further reduces software licensing expenses.

Operational Feasibility

Operationally, the system enhances efficiency by automating data collection, alerting, and traffic monitoring tasks that would otherwise require human oversight. The use of real-time alerts helps city officials respond quickly to environmental and traffic issues, improving safety and responsiveness. The interactive frontend dashboard provides a centralized view of sensor data, making daily monitoring simple and accessible.

Schedule Feasibility

From a scheduling perspective, the project is structured to allow phased implementation. A basic prototype involving a limited number of sensors can be developed and tested within four to six weeks. This initial phase focuses on sensor integration, data collection, and storage in the cloud. Subsequent phases include the development of the alert system and real-time frontend dashboard.

4. SOFTWARE AND HARDWARE REQUIREMENTS

4.1 Software Requirements

The software stack includes AWS cloud services for backend operations and React.js for frontend visualization. AWS IoT Core manages sensor connections, API Gateway handles REST requests, and AWS Lambda processes data. DynamoDB stores live data, while S3 is used for backups. SNS sends notifications. The frontend uses React.js with mapping and charting libraries to display real-time data. Development is done using VS Code and managed with Git.

Component	**Specification**
Operating System:	Windows 10 / Ubuntu
Coding Language:	Python, JavaScript (React)
Tool:	Visual Studio Code (VS Code)
Server:	AWS Lambda, API Gateway
Frontend Framework:	React.js
Database:	Amazon DynamoDB
File Storage:	Amazon S3
Notifications:	Amazon SNS

4.2 Hardware Requirements

The hardware setup involves sensor nodes to detect air quality, noise, humidity, and traffic flow. These are powered by microcontrollers like Arduino or ESP32. Communication is enabled through Wi-Fi or cellular modules. Outdoor units may use batteries or solar panels. This also uses machines for development and testing are done using laptops or PCs with simulated inputs if physical sensors. The collected data is transmitted to the cloud using lightweight protocols like MQTT for real-time monitoring. Protective enclosures are used to shield the sensors from dust, heat, and rain in outdoor environments.

Component	**Specification**
Connectivity Modules:	Wi-Fi (ESP8266/ESP32)
Development Machines	:PCs or laptops
Testing Equipment:	Virtual sensor simulators, mock traffic generators
Connection Interface:	MQTT protocol for IoT Core data transmission

5 SOFTWARE DESIGN

5.1 System Architecture

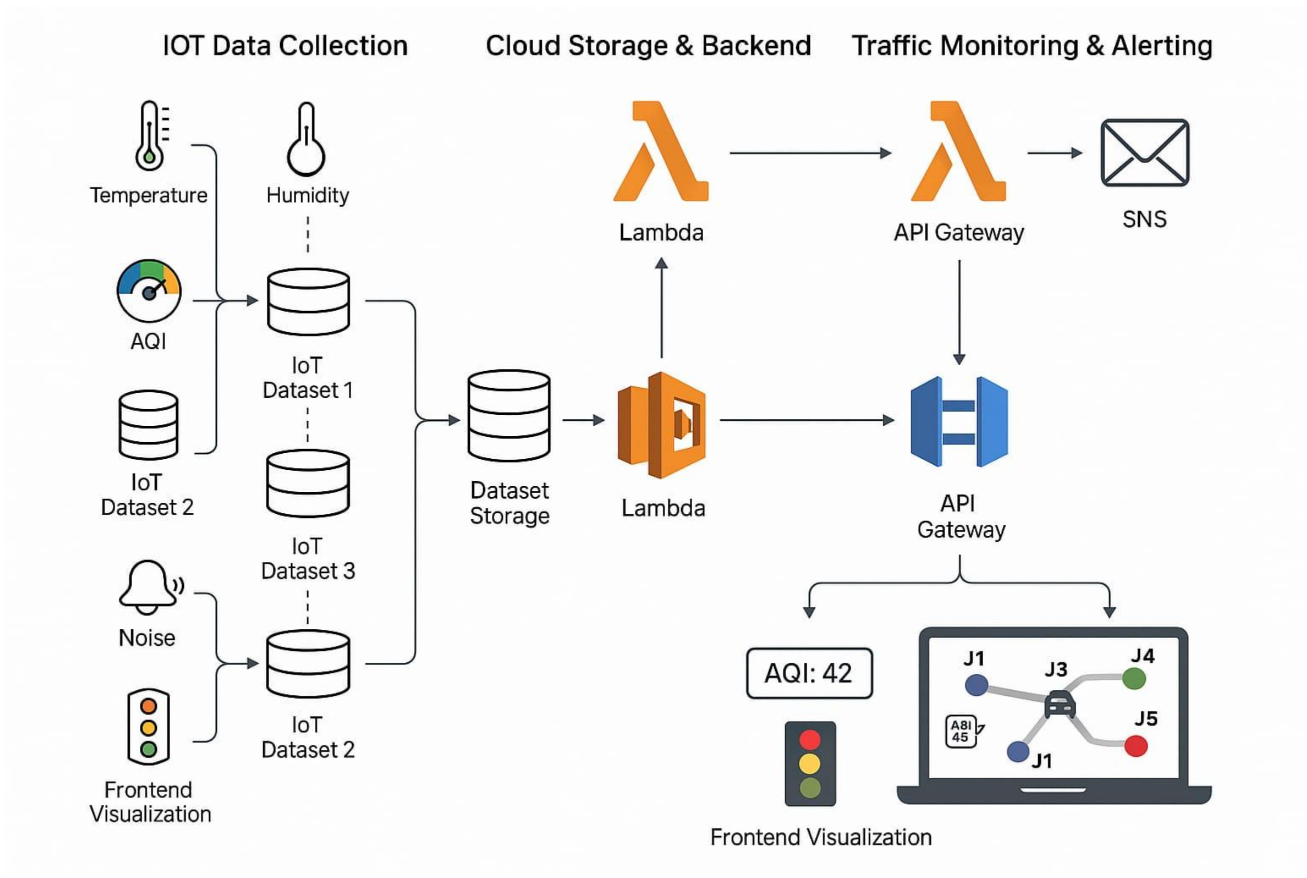


Figure: 5.1 System Architecture

The system architecture begins with IoT sensors deployed at key city locations to collect data on air quality, noise, humidity, and traffic. These sensors send real-time readings using the MQTT protocol to AWS IoT Core. The incoming data is then stored in Amazon S3 for long-term archival and analysis. A Lambda function is triggered to process and validate the data, checking for anomalies and threshold violations. If a critical condition is detected, Lambda sends notifications via Amazon SNS to subscribed users. Processed data is also routed through Amazon API Gateway to make it securely accessible to the frontend. The React.js-based frontend fetches this data using Axios or Fetch API. It displays the information on an interactive dashboard with live sensor readings, maps, and traffic signals. The architecture ensures low latency and scalable performance using serverless components. This end-to-end flow supports real-time monitoring, alerts, and control of smart city infrastructure.

5.2 Dataflow Diagram

The **dataflow information** for your Smart City Infrastructure Monitoring system:

1. **IoT Sensors** (Air Quality, Noise, Humidity, Traffic) collect real-time environmental and traffic data from various junctions and roads in the city.
2. The data is transmitted securely using **MQTT protocol** to **AWS IoT Core**, which acts as the initial ingestion point.
3. From IoT Core, data is stored in **Amazon S3** for logging and archival purposes.
4. **AWS Lambda** functions are triggered to process the incoming data — cleaning, formatting, and performing threshold checks.
5. If thresholds are exceeded (e.g., high AQI or noise), **Amazon SNS** sends alerts via email or SMS to subscribed users.
6. Processed data is made accessible via **API Gateway**, exposing RESTful APIs.
7. The **Frontend Dashboard**, built using **React.js**, periodically fetches this data using `fetch()`,visualizing it with maps and charts for real-time monitoring and decision-making.

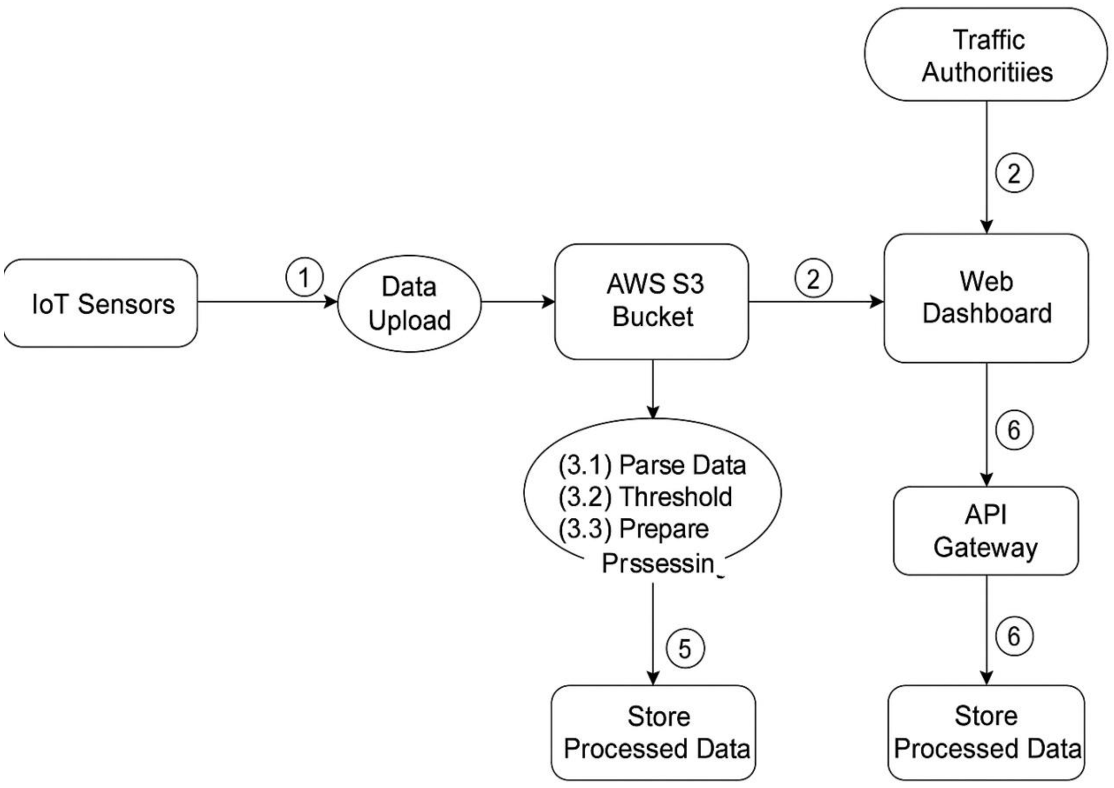


Figure: 5.2 Dataflow Diagram

5.3 UML Diagrams

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and the UML 1.0 specification draft was proposed to the OMG in January 1997.

There are several types of UML diagrams, each of which serves a different purpose regardlessof Behavioral whether it is being designed before the implementation or after (as part of documentation). UMLis directly related to object-oriented analysis and design. After some standardization, UML hasbecome an OMG standard. The two broadest categories that encompass all other types are:

- UML diagram and
- Structural UML diagram.

  As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas other describe the behavior of the system, its actors, and its building components.

Goals: The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language to develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

The different types are as follows:

- Use case Diagram
- Class diagram
- Sequence diagram
- Activity diagram
- Collaboration diagram

Use Case Diagram

A use case diagram is a high-level visual representation that captures the functional aspects of a system by showing the interactions between external actors and the various use cases or services the system provides. It plays a critical role in modeling the system's intended behavior and helps stakeholders understand what the system is supposed to do without going into technical implementation details. In a use case diagram, actors—typically users or other systems—are represented as stick figures, while the system's functionalities are depicted as ovals or ellipses known as use cases. The relationships between actors and use cases are illustrated through connecting lines or arrows, sometimes labeled with terms like "includes" or "extends" to show how use cases relate to one another.

For the Smart City Infrastructure Monitoring and Traffic Control project, the use case diagram outlines how different actors interact with the system. For example, IoT sensor nodes (acting as system actors) collect data such as air quality, noise, and humidity, and send it to AWS cloud services.

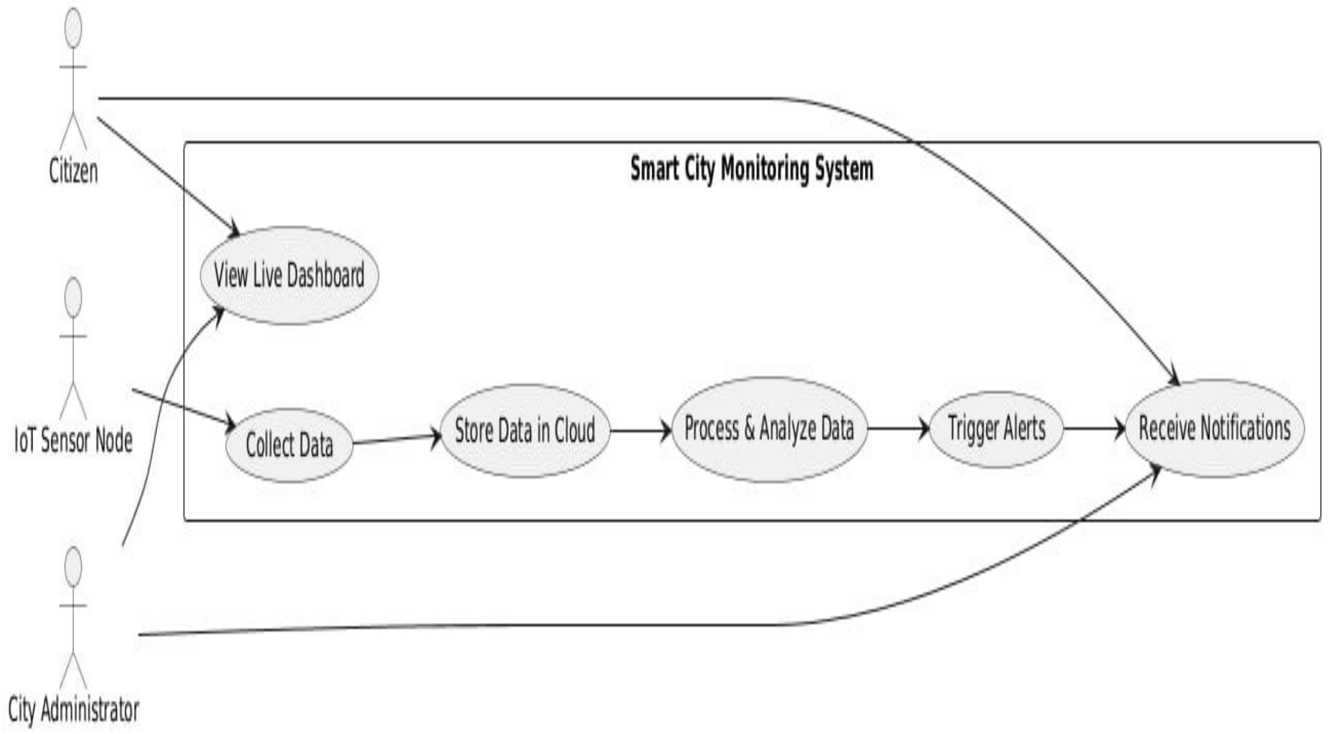


Figure 5.3.1 Use Case Diagram

Class Diagram

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

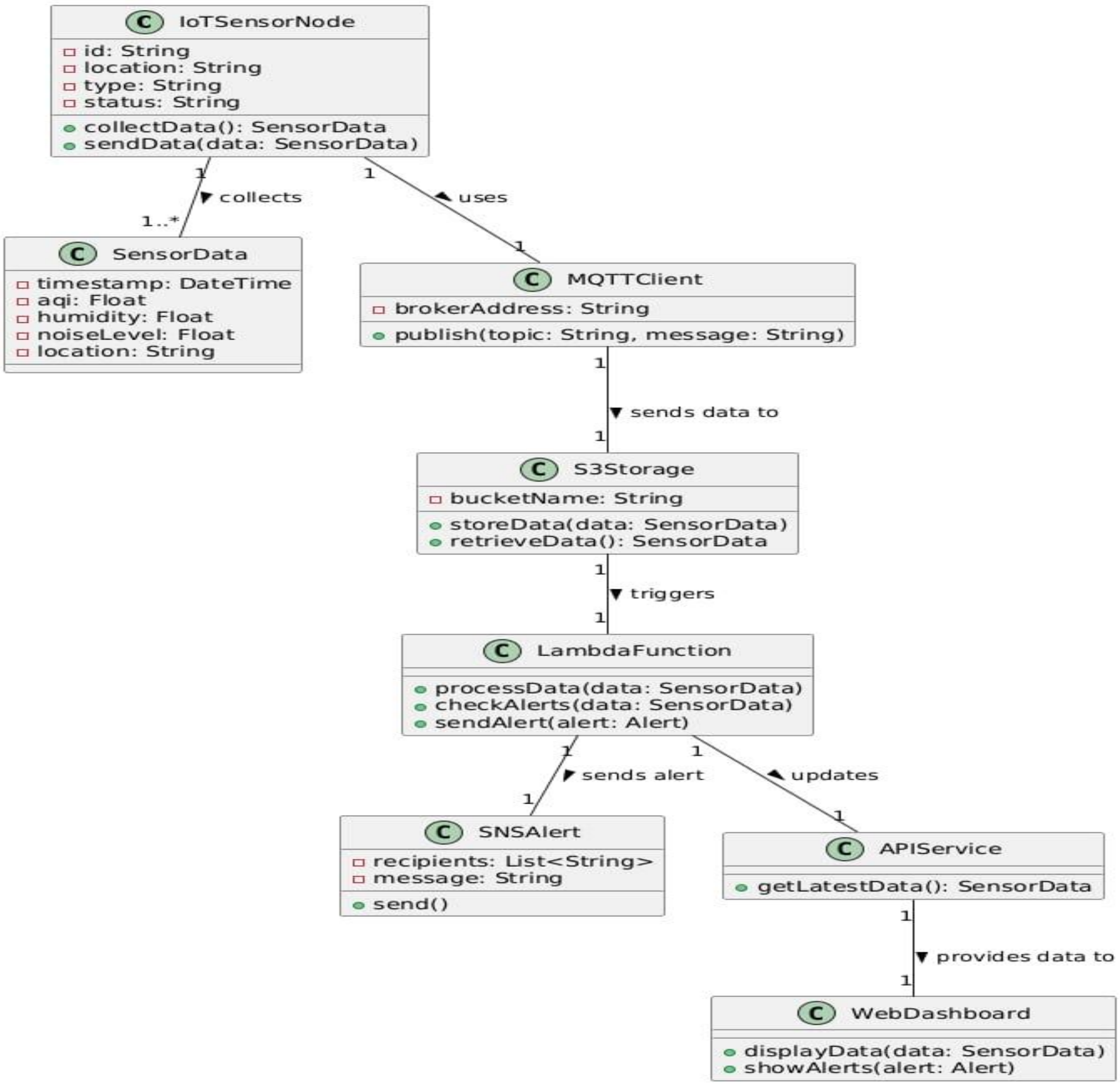


Figure 5.3.2 Class Diagram

Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

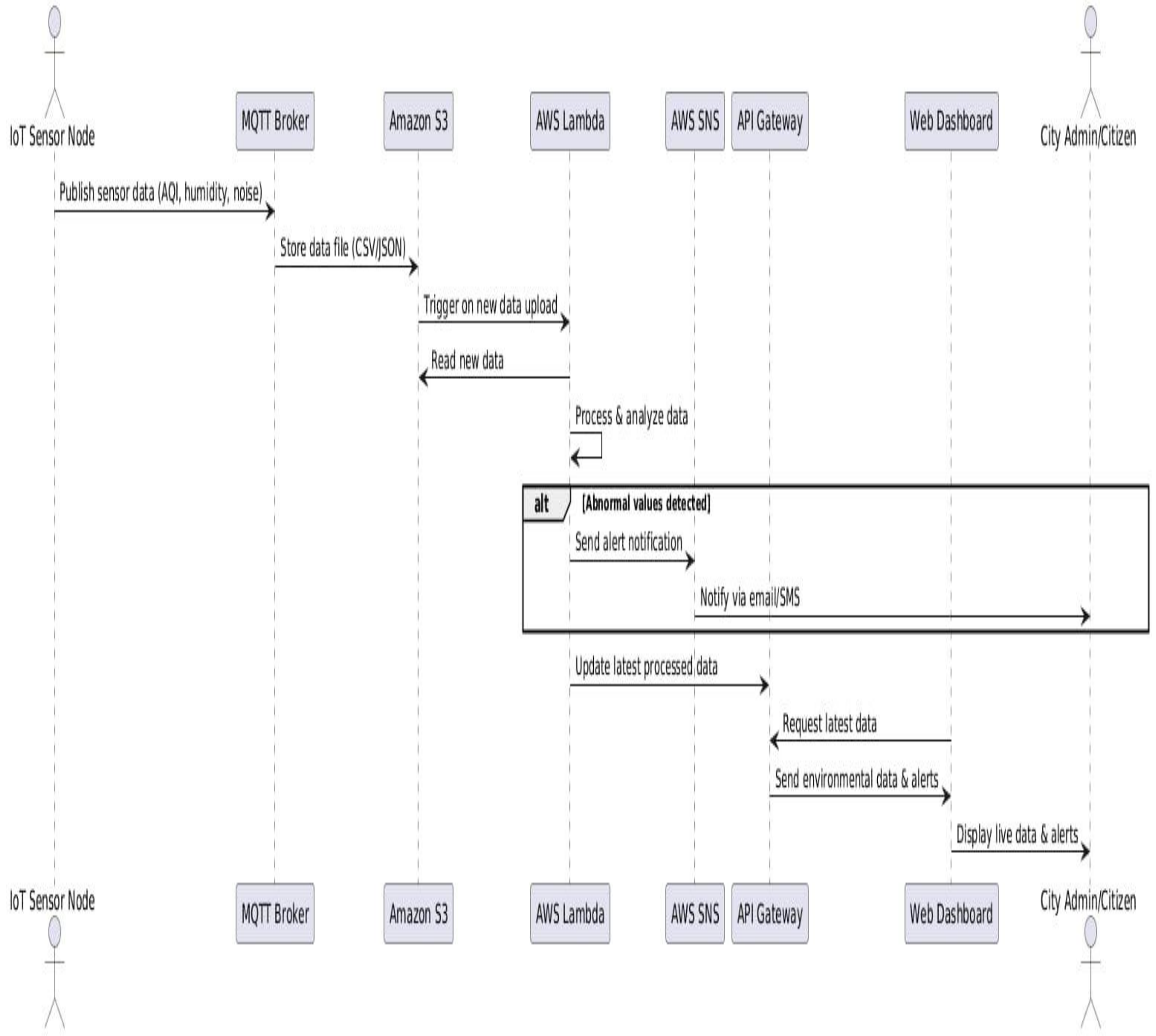
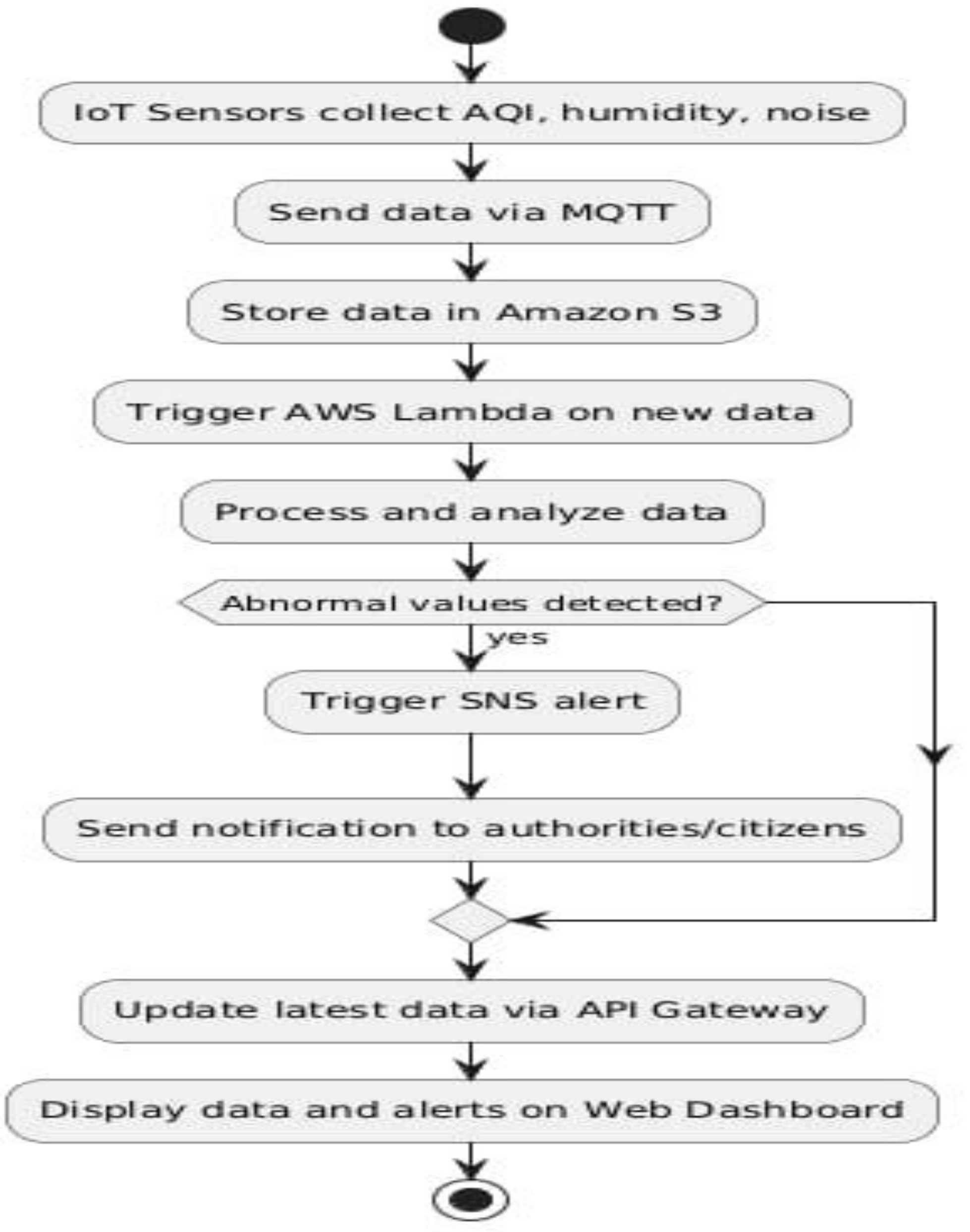

IoT Sensor Node
MQTT Broker
Amazon S3
AWS Lambda
AWS SNS
API Gateway
Web Dashboard
City Admin/Citizen
Publish sensor data (AQI, humidity, noise)
Store data file (CSV/JSON)
Trigger on new data upload
Read new data
Process & analyze data
alt
[Abnormal values detected]
Send alert notification
Notify via email/SMS
Update latest processed data
Request latest data
Send environmental data & alerts
Display live data & alerts

Figure 5.3.3 Sequence Diagram

Activity Diagram

The activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all types of flow control by using different elements such as fork, join, etc.



IoT Sensors collect AQI, humidity, noise
Send data via MQTT
Store data in Amazon S3
Trigger AWS Lambda on new data
Process and analyze data
Abnormal values detected?
yes
Trigger SNS alert
Send notification to authorities/citizens
Update latest data via API Gateway
Display data and alerts on Web Dashboard

Figure 5.3.4 Activity Diagram

Collaboration Diagram

The collaboration starts with IoT sensors deployed at various city junctions that continuously collect environmental data such as air quality (AQI), humidity, and noise levels. These sensor readings are aggregated into structured datasets and sent to an AWS S3 bucket for centralized cloud storage. Once the data is stored, AWS Lambda functions are triggered automatically to process and analyze the incoming datasets, checking for abnormal conditions or threshold violations. Upon detecting any critical values, Lambda invokes Amazon SNS to send instant notifications to administrators or stakeholders, enabling prompt action. Additionally, the processed data is exposed through AWS API Gateway, which allows the frontend application to fetch and display real-time insights and alerts on an interactive web dashboard. This interconnected collaboration between hardware (sensors), cloud services (S3, Lambda, SNS, API Gateway), and the user interface ensures automated, scalable, and efficient monitoring and decision-making, significantly enhancing the management of city infrastructure and environmental health.

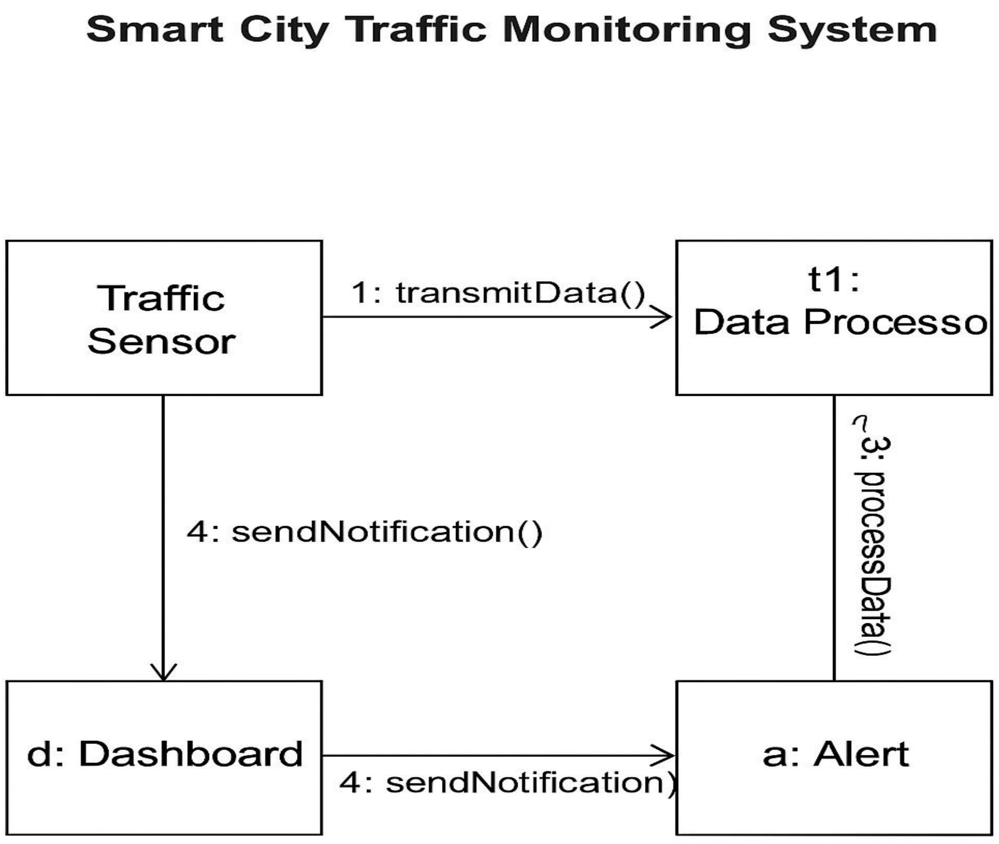


Figure 5.3.5 Collaboration Diagram

6 CODING AND ITS IMPLEMENTATION

6.1 Source Code

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>□ Smart City Traffic Monitoring Dashboard</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap @5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

  <!-- Bootstrap Icons -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons @1.10.5/font/bootstrap-icons.css"
rel="stylesheet">

  <!-- Custom Styles -->
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <header class="text-center py-4 bg-primary text-white">
    <h1>□ Smart City Traffic Monitoring Dashboard</h1>
    <p>Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning</p>
  </header>

  <main class="py-4">
    <div class="container-fluid">
      <div class="row">
        <div class="col-md-8">
          <div class="map-container">
            <svg id="trafficMap" viewBox="0 0 800 600"></svg>
          </div>
        </div>
        <div class="col-md-4">
          <div class="stats-panel">
            <div class="card mb-4 shadow-sm">
              <div class="card-body">
                <h5 class="card-title">□ Live Metrics</h5>
                <ul id="junctionStatus" class="list-group list-group-flush"></ul>
              </div>
            </div>
            <div class="card shadow-sm">
              <div class="card-body">
                <h5 class="card-title">□ Total Vehicles: <span id="totalVehicles">0</span></h5>
                <form class="route-form">
                  <div class="mb-3">
                    <label for="datasetSelect" class="form-label">Select Dataset</label>
                    <select id="datasetSelect" class="form-select"></select>
                  </div>
                  <div class="mb-3">
                    <label for="startJunction" class="form-label">Start Junction</label>
                    <select id="startJunction" class="form-select"></select>
                  </div>
                  <div class="mb-3">
                    <label for="endJunction" class="form-label">End Junction</label>
                    <select id="endJunction" class="form-select"></select>
                  </div>
                  <button type="button" onclick="findAndDrawPath()" class="btn btn-primary w-100">Find
Optimal Route</button>
                  <div id="routeResult" class="mt-3"></div>
                </form>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </main>

  <footer class="bg-light text-center py-3 mt-5 border-top">
    <p class="mb-0">© 2025 Smart Traffic Management System • Real-Time Traffic Monitoring</p>
  </footer>

  <!-- Script -->
  <script src="script.js"></script>
</body>
</html>
```

Styles.css

```
:root {
  --primary: #00f3ff;
  --secondary: #ff007a;
  --bg: #01010f;
  --card-bg: rgba(255, 255, 255, 0.05);
  --text: #ffffff;
  --accent: #fff;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: var(--bg);
  color: var(--text);
  margin: 0;
  padding: 0;
}

header {
  background: linear-gradient(45deg, var(--primary), var(--secondary));
  color: transparent;
  -webkit-background-clip: text;
  background-clip: text;
  text-align: center;
  padding: 2rem 1rem 1rem;
  position: relative;
  border-bottom: 1px solid var(--primary);
}

.map-container {
  position: relative;
  height: 80vh;
  background: #0d0d1f;
  border-radius: 15px;
  overflow: hidden;
  box-shadow: 0 0 30px rgba(0, 255, 255, 0.1);
}

svg {
  width: 100%;
  height: 100%;
}

.card {
  background-color: var(--card-bg);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 15px;
  padding: 1.5rem;
  backdrop-filter: blur(10px);
  transition: all 0.3s ease;
}

.card:hover {
  transform: translateY(-5px);
  box-shadow: 0 0 20px rgba(0, 255, 255, 0.2);
}

.route-form {
  background: var(--card-bg);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 15px;
  padding: 1.5rem;
  backdrop-filter: blur(10px);
}

.form-label {
  color: var(--primary);
  font-weight: 600;

}
select.form-select {
  background-color: #1a1a2e;
  border: 1px solid rgba(255, 255, 255, 0.1);
  color: white;

}

footer {
  text-align: center;
  padding: 1rem;
  color: #ccc;
}

.alert {
  margin-top: 1rem;
}

.junction {
  cursor: pointer;
}

.junction:hover {
  transform: scale(1.1);
  z-index: 10;
}
```

Script.js

```
const JUNCTION_RADIUS = 14;
const VEHICLE_SPEED = 0.5; // pixels per frame
const DATASET_SWITCH_INTERVAL = 5000;

// Static junction definitions (positions remain unchanged)
const junctions = [
  { id: "J1", x: 100, y: 100, aqi: 0 },
  { id: "J2", x: 300, y: 100, aqi: 0 },
  { id: "J3", x: 500, y: 100, aqi: 0 },
  { id: "J4", x: 100, y: 300, aqi: 0 },
  { id: "J5", x: 300, y: 300, aqi: 0 },
  { id: "J6", x: 500, y: 300, aqi: 0 }
];

const roads = [
  { from: "J1", to: "J2" }, { from: "J2", to: "J3" },
  { from: "J1", to: "J4" }, { from: "J2", to: "J5" }, { from: "J3", to: "J6" },
  { from: "J4", to: "J5" }, { from: "J5", to: "J6" }
];

let currentDatasetIndex = 0;
let datasetKeys = ["Live Dataset - Fetched from API"];
let currentAQIData = [];

let junctionSignals = {};
let activeVehicles = [];
```

```
const trafficMap = document.getElementById("trafficMap");
const apiEndpoint = "https://d3wui4o2xg.execute-api.ap-south-1.amazonaws.com/prod/process-data ";


// Function to fetch data from the API Gateway
async function fetchAQIData() {
  try {
    const response = await fetch(apiEndpoint);
    if (!response.ok) throw new Error("Failed to fetch data");

    const result = await response.json();

    // Assuming the API returns an array of objects similar to datasets
    currentAQIData = result || [];
    drawMap();
  } catch (error) {
    console.error("Error fetching data:", error);
    alert("Could not load live dataset from API.");
  }
}

function applyAQIData(aqiData) {
  aqiData.forEach(d => {
    const j = junctions.find(j => j.id === d.id);
    if (j) {
      j.aqi = d.aqi !== undefined ? Number(d.aqi) : null;
      j.noiseLevel = d.noiseLevel !== undefined ? Number(d.noiseLevel) : null;
      j.humidity = d.humidity !== undefined ? Number(d.humidity) : null;
    }
  });
}

function assignSignals() {
  junctions.forEach(j => {
    let hasNegative = false;
    if (
      j.aqi !== null && j.aqi < 0 ||
      j.noiseLevel !== null && j.noiseLevel < 0 ||
      j.humidity !== null && j.humidity < 0
    ) {
      hasNegative = true;
    }

    if (hasNegative) {
      junctionSignals[j.id] = "red";
    } else if (j.aqi === null || j.aqi === 0|| j.aqi >0 && j.aqi<=100) {
      junctionSignals[j.id] = "green"; // Safe if 0 or unknown
    } else if (j.aqi >=100 && j.aqi <= 200) {

      junctionSignals[j.id] = "yellow";
    } else {

      junctionSignals[j.id] = "red";
    }
  });
}


function getSignalColor(signal) {
  return signal === "green" ? "#4caf50" :
         signal === "yellow" ? "#ffeb3b" : "#f44336";
}
function getAQIColor(aqi) {
  if (aqi === null) return "#fff";
  if (aqi <= 50) return "#4caf50";
  if (aqi <= 100) return "#ffeb3b";
  if (aqi <= 150) return "#ff9800";
  if (aqi <= 200) return "#f44336";
  if (aqi <= 300) return "#9c27b0";
  return "#d32f2f";
}

function getNoiseColor(noiseLevel) {
  if (noiseLevel === null) return "#fff";
  if (noiseLevel <= 50) return "#4caf50";
  if (noiseLevel <= 70) return "#ffeb3b";
  if (noiseLevel <= 80) return "#ff9800";
  if (noiseLevel <= 90) return "#f44336";
  return "#d32f2f";
}

function getHumidityColor(humidity) {
  if (humidity === null) return "#fff";
  if (humidity <= 30) return "#ff9800";
  if (humidity <= 60) return "#4caf50";
  if (humidity <= 80) return "#ffeb3b";
  return "#f44336";
}

function getAlertStatus(junction) {
  const alerts = [];
  if (junction.aqi !== null && junction.aqi < 0) {
    alerts.push({ type: "AQI", level: "Invalid", value: junction.aqi });
  }
  if (junction.noiseLevel !== null && junction.noiseLevel < 0) {
    alerts.push({ type: "Noise", level: "Invalid", value: junction.noiseLevel });
  }
  if (junction.humidity !== null && junction.humidity < 0) {
    alerts.push({ type: "Humidity", level: "Invalid", value: junction.humidity });
  }

  return alerts;
}
function updateMetrics() {
  const panel = document.getElementById("junctionStatus");
  panel.innerHTML = "";
  junctions.forEach(j => {
    const signal = junctionSignals[j.id];
    const badgeClass = signal === 'green'
      ? 'success'
      : signal === 'yellow' ? 'warning text-dark' : 'danger';

    const alerts = getAlertStatus(j);
    const li = document.createElement("li");
    li.className = "list-group-item d-flex justify-content-between align-items-center";
    let metricString = `<strong>${j.id}</strong>`;
    if (j.aqi !== null) metricString += ` - AQI: ${j.aqi}`;
    if (j.noiseLevel !== null) metricString += `, Noise: ${j.noiseLevel} dB`;
    if (j.humidity !== null) metricString += `, Humidity: ${j.humidity}%`;
    if (alerts.length > 0) {
      const alertText = alerts.map(a => `${a.type}: ${a.level}`).join(", ");
      metricString += `<br><span class="text-danger">□ Alert: ${alertText}</span>`;
    }
    li.innerHTML = `
      <span>${metricString}</span>
      <span class="badge bg-${badgeClass}">
        ${signal.toUpperCase()}
      </span>
    `;
    panel.appendChild(li);
  });
  document.getElementById("totalVehicles").textContent = activeVehicles.length;
}

class Vehicle {
  constructor(fromId, toId) {
    this.from = junctions.find(j => j.id === fromId);
    this.to = junctions.find(j => j.id === toId);
    this.progress = 0;
    this.totalDistance = Math.hypot(this.to.x - this.from.x, this.to.y - this.from.y);
    this.speed = VEHICLE_SPEED * (junctionSignals[this.to.id] === "yellow" && this.to.aqi <= 200 ?
0.5 : 1);
    this.element = null;
    this.animationFrame = null;
  }

  start() {
    this.element = document.createElementNS("http://www.w3.org/2000/svg", "rect");
    this.element.setAttribute("x", this.from.x);
    this.element.setAttribute("y", this.from.y);
    this.element.setAttribute("width", "10");
    this.element.setAttribute("height", "6");
    this.element.setAttribute("fill", "orange");

    trafficMap.appendChild(this.element);
    this.animate();
  }

  animate() {
```

```
const dx = this.to.x - this.from.x;
const dy = this.to.y - this.from.y;
const stepX = this.speed * (dx / this.totalDistance);
const stepY = this.speed * (dy / this.totalDistance);
const moveStep = () => {
  if (this.progress >= this.totalDistance) {

    if (junctionSignals[this.to.id] === "red" || this.to.aqi > 200) {
      this.element.setAttribute("fill", "#555");
      const newPath = findSafePath(this.to.id, "J6"); // Try rerouting
      if (newPath && newPath.length > 1) {
        const nextJunction = newPath[1];
        const newVehicle = new Vehicle(this.to.id, nextJunction);
       newVehicle.start();
        activeVehicles.push(newVehicle);
      } else {
        setTimeout(() => {
          if (trafficMap.contains(this.element)) {
            trafficMap.removeChild(this.element);
          }
        }, 5000);
      }
    } else {
      trafficMap.removeChild(this.element);
    }
    return;
  }
  if (junctionSignals[this.to.id] === "red" || this.to.aqi > 200) {
    cancelAnimationFrame(this.animationFrame);
    this.element.setAttribute("fill", "#555");
    const newPath = findSafePath(this.to.id, "J6");
    if (newPath && newPath.length > 1) {
      const nextJunction = newPath[1];
      const newVehicle = new Vehicle(this.to.id, nextJunction);
      newVehicle.start();
      activeVehicles.push(newVehicle);
    } else {
      setTimeout(() => {
        if (trafficMap.contains(this.element)) {
          trafficMap.removeChild(this.element);
        }
      }, 5000);
    }
    return;
  }
  this.progress += this.speed;
  const newX = this.from.x + (dx * this.progress / this.totalDistance);

  const newY = this.from.y + (dy * this.progress / this.totalDistance);
  this.element.setAttribute("x", newX);
  this.element.setAttribute("y", newY);
  this.animationFrame = requestAnimationFrame(moveStep);
```

```
    };
    moveStep();
  }

  stop() {
    cancelAnimationFrame(this.animationFrame);
  }

}

function drawMap() {
  if (currentAQIData.length === 0) {
    console.warn("No AQI data loaded yet.");
    return;
  }
  applyAQIData(currentAQIData);
  assignSignals();

  // Redraw map and vehicles
  trafficMap.innerHTML = `
    <defs>
      <marker id="arrowhead" markerWidth="10" markerHeight="7" refX="5" refY="3.5"
orient="auto">
        <polygon points="0 0, 10 3.5, 0 7" fill="#000"/>
      </marker>
    </defs>
  `;

  roads.forEach(r => {
    const from = junctions.find(j => j.id === r.from);
    const to = junctions.find(j => j.id === r.to);
    const roadLine = document.createElementNS("http://www.w3.org/2000/svg", "line");
    roadLine.setAttribute("x1", from.x);
    roadLine.setAttribute("y1", from.y);
    roadLine.setAttribute("x2", to.x);
    roadLine.setAttribute("y2", to.y);
    roadLine.setAttribute("stroke", getRoadColor(r.from, r.to));
    roadLine.setAttribute("stroke-width", "4");
    trafficMap.appendChild(roadLine);
  });

  activeVehicles.forEach(v => v.stop());
  activeVehicles = [];

  junctions.forEach(jFrom => {
    if (junctionSignals[jFrom.id] === "red") return;

    const outgoingRoads = roads.filter(r => r.from === jFrom.id);
    outgoingRoads.forEach(r => {
      const jTo = junctions.find(j => j.id === r.to);
      const targetSignal = junctionSignals[jTo.id];
      if (
        targetSignal !== "red" &&
      (jTo.aqi === null || jTo.aqi <= 200)
    ) {
      const vehicle = new Vehicle(jFrom.id, jTo.id);
      vehicle.start();
      activeVehicles.push(vehicle);
    }
  });

});

junctions.forEach(j => {
  const alerts = getAlertStatus(j);
  const signalColor = alerts.some(a => a.level === "Invalid")
    ? "#f44336"
    : getSignalColor(junctionSignals[j.id]);

  const circle = document.createElementNS("http://www.w3.org/2000/svg", "circle");
  circle.setAttribute("cx", j.x);
  circle.setAttribute("cy", j.y);
  circle.setAttribute("r", "14");
  circle.setAttribute("fill", signalColor);
  trafficMap.appendChild(circle);

  const label = document.createElementNS("http://www.w3.org/2000/svg", "text");
  label.textContent = `${j.id} (${junctionSignals[j.id]})`;
  label.setAttribute("x", j.x - 10);
  label.setAttribute("y", j.y + 30);
  label.setAttribute("font-size", "12");
  label.setAttribute("fill", "#fff");
  trafficMap.appendChild(label);

  const aqiText = document.createElementNS("http://www.w3.org/2000/svg", "text");
  aqiText.textContent = `AQI: ${j.aqi !== null ? j.aqi : "-"}`;
  aqiText.setAttribute("x", j.x + 15);
  aqiText.setAttribute("y", j.y + 25);
  aqiText.setAttribute("font-size", "12");
  aqiText.setAttribute("fill", getAQIColor(j.aqi));
  trafficMap.appendChild(aqiText);

  if (j.noiseLevel !== null) {
    const noiseText = document.createElementNS("http://www.w3.org/2000/svg", "text");
    noiseText.textContent = `Noise: ${j.noiseLevel} dB`;
    noiseText.setAttribute("x", j.x + 15);
    noiseText.setAttribute("y", j.y + 40);
    noiseText.setAttribute("font-size", "12");
    noiseText.setAttribute("fill", getNoiseColor(j.noiseLevel));

    trafficMap.appendChild(noiseText);
  }
  if (j.humidity !== null) {
    const humidityText = document.createElementNS("http://www.w3.org/2000/svg", "text");
    humidityText.textContent = `Humidity: ${j.humidity}%`;
```

```
      humidityText.setAttribute("x", j.x + 15);
      humidityText.setAttribute("y", j.y + 55);
      humidityText.setAttribute("font-size", "12");
      humidityText.setAttribute("fill", getHumidityColor(j.humidity));
      trafficMap.appendChild(humidityText);
    }


    if (getAlertStatus(j).length > 0) {
      const warning = document.createElementNS("http://www.w3.org/2000/svg", "text");
      warning.textContent = "□";
      warning.setAttribute("x", j.x + 20);
      warning.setAttribute("y", j.y - 20);
      warning.setAttribute("font-size", "14");
      warning.setAttribute("fill", "#ff4d4d");
      trafficMap.appendChild(warning);

    }
  });
  updateMetrics();
}

function populateDatasets() {
  const select = document.getElementById("datasetSelect");
  datasetKeys.forEach((name) => {
    const opt = document.createElement("option");
    opt.value = name;
    opt.textContent = name;
    select.appendChild(opt);
  });

  select.onchange = () => {
    fetchAQIData(); // Re-fetch data when dataset changes (if needed)
  };
}

function getRoadColor(fromId, toId) {
  const fromSignal = junctionSignals[fromId];
  const toSignal = junctionSignals[toId];
  const fromAQI = junctions.find(j => j.id === fromId)?.aqi;
  const toAQI = junctions.find(j => j.id === toId)?.aqi;

  if (
    fromSignal !== "red" &&
    toSignal !== "red" &&

    (fromAQI === null || fromAQI <= 200) &&
    (toAQI === null || toAQI <= 200)
  ) {
    return "#4caf50"; // Green
  } else {
    return "#f44336"; // Red
  }
```

```
}

function findSafePath(start, end) {
  const graph = {};
  roads.forEach(r => {
    if (!graph[r.from]) graph[r.from] = [];
    graph[r.from].push(r.to);
  });

  const queue = [{ node: start, path: [start] }];
  const visited = new Set([start]);

  while (queue.length > 0) {
    const { node, path } = queue.shift();
    if (node === end) return path;

    const neighbors = graph[node] || [];

    for (let neighbor of neighbors) {
      const neighborNode = junctions.find(j => j.id === neighbor);
      const neighborAQI = neighborNode?.aqi;
      const neighborSignal = junctionSignals[neighbor];

      if (
        neighborAQI > 200 ||
        neighborSignal === "red" ||
        visited.has(neighbor)
      ) continue;

      visited.add(neighbor);
      queue.push({ node: neighbor, path: [...path, neighbor] });
    }
  }

  return null;
}

function drawRoute(path) {
  trafficMap.querySelectorAll(".highlight-path").forEach(el => el.remove());
  for (let i = 0; i < path.length - 1; i++) {
    const from = junctions.find(j => j.id === path[i]);
    const to = junctions.find(j => j.id === path[i + 1]);
    const line = document.createElementNS("http://www.w3.org/2000/svg", "line");
    line.setAttribute("x1", from.x);
    line.setAttribute("y1", from.y);
    line.setAttribute("x2", to.x);
    line.setAttribute("y2", to.y);
    line.setAttribute("stroke", "blue");
    line.setAttribute("stroke-width", "4");
    line.setAttribute("class", "highlight-path");
    trafficMap.appendChild(line);

  }
```

```
}
function findAndDrawPath() {
  const start = document.getElementById("startJunction").value;
  const end = document.getElementById("endJunction").value;
  const resultDiv = document.getElementById("routeResult");

  if (!start || !end || start === end) {

    resultDiv.innerHTML = '<div class="alert alert-warning">Select valid start and end junctions.</div>';
    return;
  }

  const path = findSafePath(start, end);
  if (!path) {
    resultDiv.innerHTML = '<div class="alert alert-danger">□ No safe route due to red signals or
hazardous AQI.</div>';
    return;

  }
  resultDiv.innerHTML = `<div class="alert alert-success">□ Optimal Safe Route: ${path.join(" →
")}</div>`;
  drawRoute(path);
}

function populateDropdowns() {
  const start = document.getElementById("startJunction");
  const end = document.getElementById("endJunction");
  junctions.forEach(j => {
    const opt = document.createElement("option");
    opt.value = j.id;
    opt.textContent = j.id;
    start.appendChild(opt.cloneNode(true));
    end.appendChild(opt);
  });
}

window.onload = () => {
  fetchAQIData(); // Fetch initial data on page load
  populateDropdowns();
  populateDatasets();
  setInterval(fetchAQIData, DATASET_SWITCH_INTERVAL); // Refresh dataset periodically
};
```

Lambda Function

```
import boto3
import json
def lambda_handler(event, context):
    s3 = boto3.client('s3')

    sns = boto3.client('sns')
    bucket_name = 's3-smartcitytraffic'
    file_key = 'dataset_2__partial_data.json'
```

```
sns_topic_arn = 'arn:aws:sns:ap-south-1:273354649801:smarttraffic'
headers = {

    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Headers": "Content-Type",
    "Access-Control-Allow-Methods": "GET"
}


try:
    response = s3.get_object(Bucket=bucket_name, Key=file_key)
    data = json.loads(response['Body'].read().decode('utf-8'))

    # Check if any junction has AQI > 150
    alert_junctions = []
    for junction in data:
        aqi = junction.get('aqi')

        if aqi is not None and aqi > 150:
            alert_junctions.append(junction['id'])
    if alert_junctions:

        message = f"Alert: High AQI detected at junction(s): {', '.join(alert_junctions)}"
        sns.publish(
            TopicArn=sns_topic_arn,
            Message=message,
            Subject='Smart Traffic Alert: High AQI Detected'
        )
    return {
        'statusCode': 200,
        'headers': headers,
        'body': json.dumps(data)
    }
except Exception as e:
    return {
        'statusCode': 500,
        'headers': headers,
        'body': json.dumps({'error': str(e)})
```

6.2 Implementation

6.2.1 Web Technologies Used

For the frontend development of the Smart City Infrastructure Monitoring and Traffic Control system, **HTML** is used to create the basic structure and layout of the web pages, ensuring that all elements like maps, sensor data panels, and control buttons are properly organized. **CSS** is employed to style these elements, making the dashboard visually appealing and responsive across different devices, from desktops to mobile screens. **JavaScript** powers the dynamic aspects of the frontend by enabling real-time data fetching from AWS API Gateway using libraries such as **Axios** or the native **fetch()** API. It also handles updating traffic lights, animating vehicle flows, and displaying sensor alerts interactively. Additionally, popular JavaScript libraries like **React.js** can be incorporated to build a reactive user interface that efficiently updates when new data arrives, while mapping libraries such as **Leaflet** or **Google Maps API** help in rendering interactive city maps. Charting libraries like **Chart.js** or **D3.js** are often used to visualize trends and historical data in graphs, enhancing data comprehension for users and city officials alike..

6.2.2 Modules Used in Implementation

Traffic Signal Simulation Module: This module determines the traffic signal color at each junction based on the AQI (Air Quality Index). If the AQI is below or equal to 100, the signal is set to green indicating safe air quality. For AQI between 101 and 200, a yellow signal is used to indicate caution. AQI values above 200 or invalid readings such as negative values or null are considered hazardous and represented by a red signal. These colors are dynamically applied to the junction nodes on the dashboard.

Vehicle Movement Simulation: Vehicles are programmatically generated and move from one junction to another based on the current environmental and traffic conditions. Each vehicle evaluates the AQI and signal of its destination before proceeding. If a junction is unsafe (AQI > 200 or red signal), the vehicle attempts to find an alternate safe route using a pathfinding algorithm. This module simulates realistic traffic flow under changing conditions.

Environmental Data Visualization: The dashboard displays real-time environmental metrics like AQI, noise levels, and humidity for each junction. These metrics are color-coded to provide intuitive visual feedback. For example, safe noise levels and humidity ranges are shown in green, while critical values are marked in red. Alerts are triggered for any negative or invalid sensor data.

Dataset Switching Mechanism: Multiple datasets representing different traffic and environmental scenarios are preloaded into the system. Users can select a dataset from a dropdown menu to simulate conditions like normal traffic, partial data loss, or extreme pollution. Upon selection, the dashboard updates the junctions, metrics, and traffic simulation accordingly in real time.

Route Planning Tool: Users can input a start and end junction to find the optimal route avoiding heavily polluted or blocked areas. The system employs a simple pathfinding logic to determine the safest path considering current AQI and signal status at each junction. The resulting route is visually highlighted on the map.

JavaScript Functions: The JavaScript file includes several key functions such as:

- assignSignals(): Assigns signal color to each junction.
- updateMetrics(): Updates the junction metrics panel.
- Vehicle class: Manages vehicle behavior, animation, and rerouting.
- findSafePath(): Calculates alternative paths avoiding hazardous junctions.
- drawMap(): Renders the junctions, roads, signals, vehicles, and live data to the dashboard.

This modular and interactive approach ensures that the smart city dashboard effectively simulates real time traffic conditions, facilitates decision-making, and provides an engaging experience for users and stakeholders.

7. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

7.1. Types Of Test

Unit testing

Unit testing involves testing individual components or functions of the application in isolation to ensure they work as expected. These are typically the smallest parts of a system, like a function or method.

Uses:

It is used to catch errors early in development and ensure that each function performs correctly on its own. Developers often automate unit tests to quickly validate logic as code evolves.

Application:

Unit testing is used to verify small components like the Lambda function that processes sensor data, logic for determining traffic light status based on congestion level, or the vehicle animation controller. For example, a function that maps AQI values to alert colors would be tested in isolation to ensure accuracy.

Integration Testing

Integration testing checks if different modules or services within the system work together correctly. It ensures data flows properly from one part of the application to another.

Uses:

It’s essential when a system involves communication between components such as APIs, databases, and frontend interfaces. It identifies issues in data exchange or interface mismatches.

Application:

Integration testing is used to ensure that sensor data is correctly fetched from IoT devices, passed through API Gateway, processed by AWS Lambda, and then displayed on the frontend dashboard. For instance, when noise levels are detected above a threshold, the alert banner must show up and also trigger an SNS alert—this end-to-end flow is tested through integration tests.

Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manual. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. Functional testing verifies that the software behaves according to the defined functional requirements.

Uses:

This testing checks whether features like data input, button clicks, data updates, and visual feedback perform as intended.

Application:

Functional tests in our project include checking whether AQI updates every few seconds, whether traffic lights change according to rules, and if the vehicle animations sync with signal lights. These ensure that user-facing features operate smoothly.

System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing involves testing the complete and integrated software system to ensure it meets the specified requirements.

Uses:

This is a high-level test that validates the system as a whole. It covers functional and non-functional aspects like performance, usability, and reliability.

Application:

In our system, this includes testing the entire functionality—sensor data flow, data display, alert generation, traffic signal logic, and vehicle animations. For example, when AQI is high and traffic is heavy, the system should display an alert, change the signal light to red, and slow down the vehicle animations—this full flow is validated during system testing.

User Acceptance Testing (UAT)

User Acceptance Testing is the final testing phase, where actual users verify that the system meets their needs and functional expectations.

Uses:

It is used to ensure the software is usable and provides value to the end users before going live or being finalized for delivery.

Application:

In our project, UAT is performed by mentors or faculty to ensure that the dashboard is user-friendly, data is clear and updated in real-time, and alerts or automation work correctly. It also checks that the Lambda

+ API Gateway architecture can return data within acceptable limits (e.g., under 200ms response time).This includes checking that traffic signals behave properly and alerts are triggered at correct thresholds based on AQI, noise, or humidity.

Performance Testing

Performance testing checks how the system performs under load, including response time, scalability, and resource usage.

Uses:

It's used to identify bottlenecks and ensure the system can handle high-frequency data updates or multiple users without crashing.

Application:

For our smart city dashboard, performance testing verifies that the real-time data updates from multiple sensors (e.g., AQI, traffic, noise) do not slow down the dashboard. It also checks that the Lambda + API Gateway architecture can return data within acceptable limits (e.g., under 200ms response time).

Compatibility and UI Testing

UI testing checks the frontend design and behavior, while compatibility testing ensures the software works across different devices, browsers, or screen sizes.

Uses:

It's used to validate layout, responsiveness, accessibility, and cross-platform consistency. Identify bottlenecks and ensure the system can handle high-frequency data updates or multiple users without crashing.

Application:

Dashboard must display correctly on desktops, tablets, and mobiles. This testing ensures that visualizations, buttons, text, and alerts scale properly and remain usable regardless of device or screen resolution.

7.2. Test Cases:

S.no	Test Case	Excepted Result	Result	Remarks
1	Display Dashboard	The dashboard with map and metrics should be visible on load.	Pass	Dashboard loads and displays as expected.
2	Dataset Switching	When a dataset is selected, junction data should update accordingly.	Pass	AQI, noise, and humidity values update correctly.
3	Invalid Dataset Values	If dataset contains AQI < 0 or missing data, alert icons should appear.	Pass	Alert □ icon is shown for invalid inputs.
4	Signal color	AQI > 200 or invalid data should set signal to red.	Pass	Junctions display red signals appropriately.
5	Vehicle Movement	Vehicles should move between safe junctions with non-red signals.	Pass	Vehicles move and reroute as expected.
6	Start/End Junction Validation	Warning should show if start and end junctions are same or empty.	Pass	Appropriate warning is displayed.
7	Safe-Route Calculation	System should calculate and display safest route avoiding red signals.	Pass	Safe path is shown and highlighted on map.
8	No-Safe-Route Available	If no path is safe, system should display a warning message.	Pass	Warning is shown: No safe route found.
9	API-Timeout Handling	Dashboard should show error.	Fail	No visual feedback; user confused during API timeout.
10	SNS Email Alert on AQI > 300	SNS should trigger email alert when AQI exceeds 300	Fail	Email SNS is not sent due to incorrect (sub).
11	S3 Data Corruption Handlig	If JSON in S3 is corrupt, system should log error and skip the record.	Fail	Lambda throws error.

Table no 7.2 Test Cases

Test Case 1:

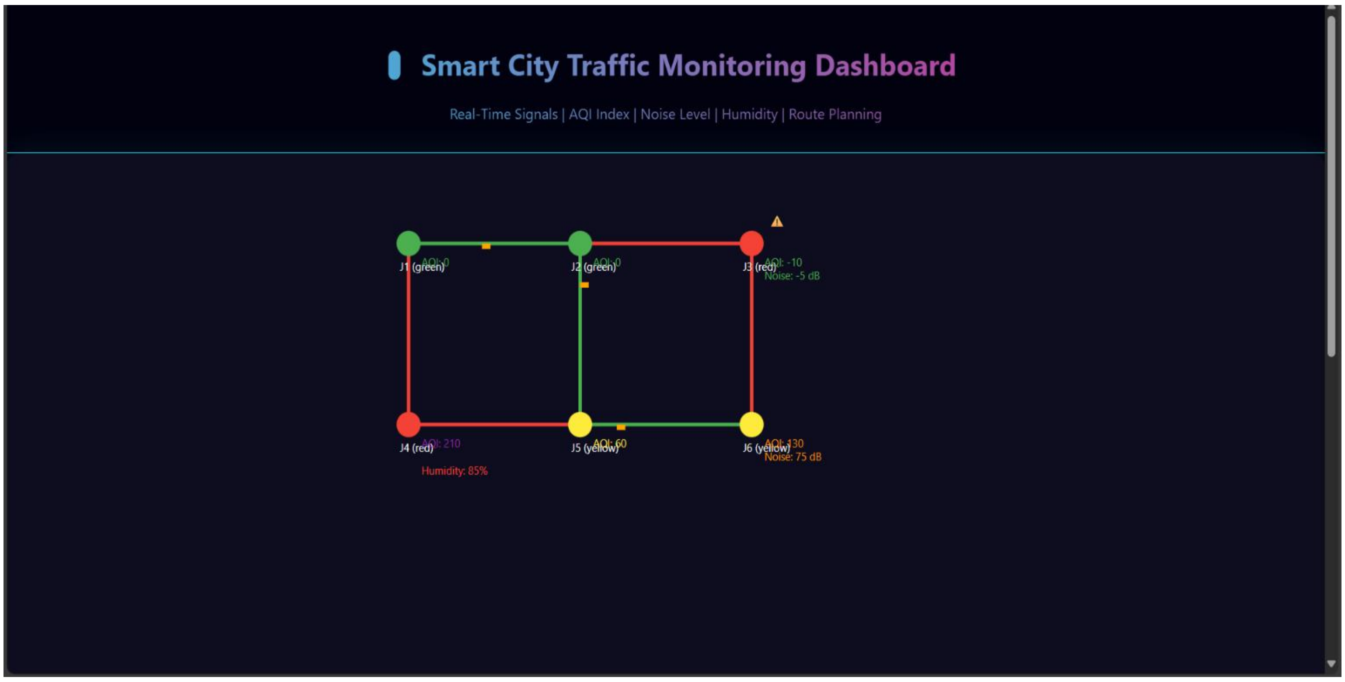
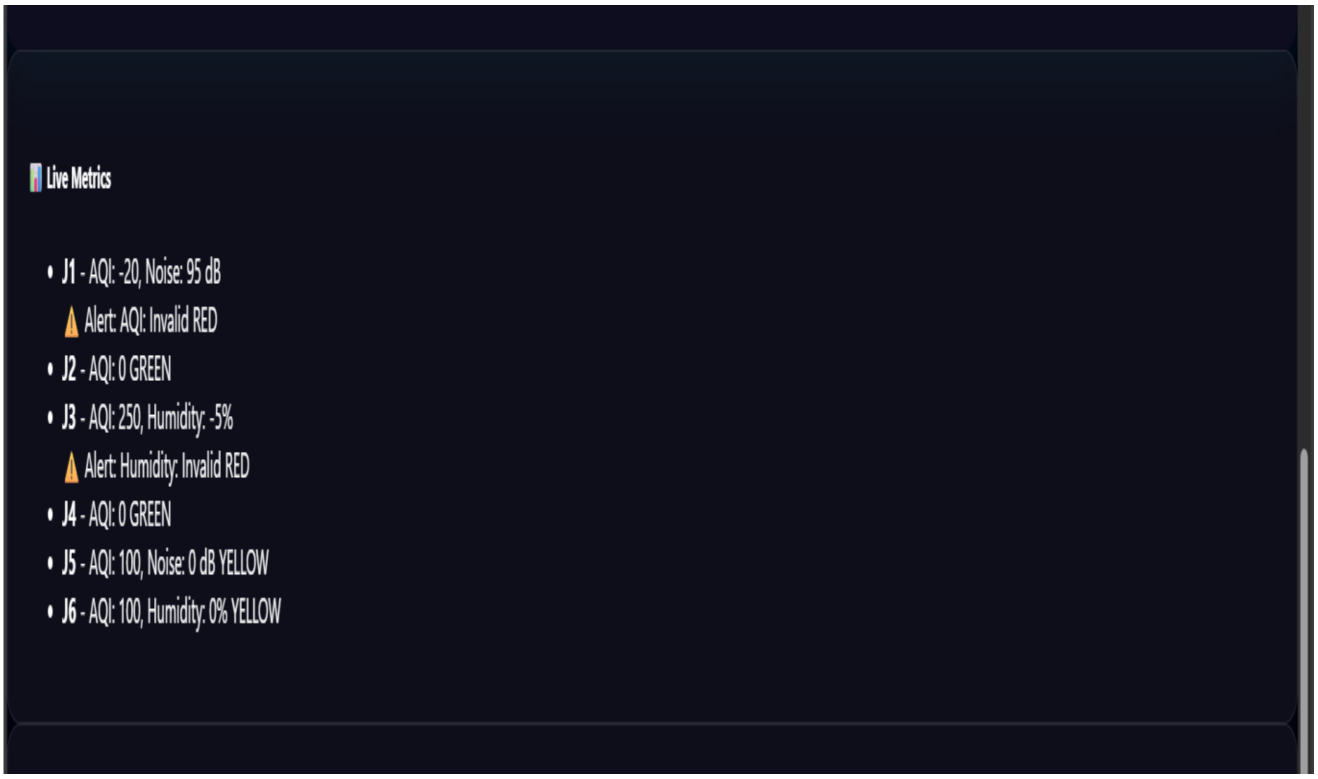


Figure 7.2.2: Test Case 1

Test Case 2:

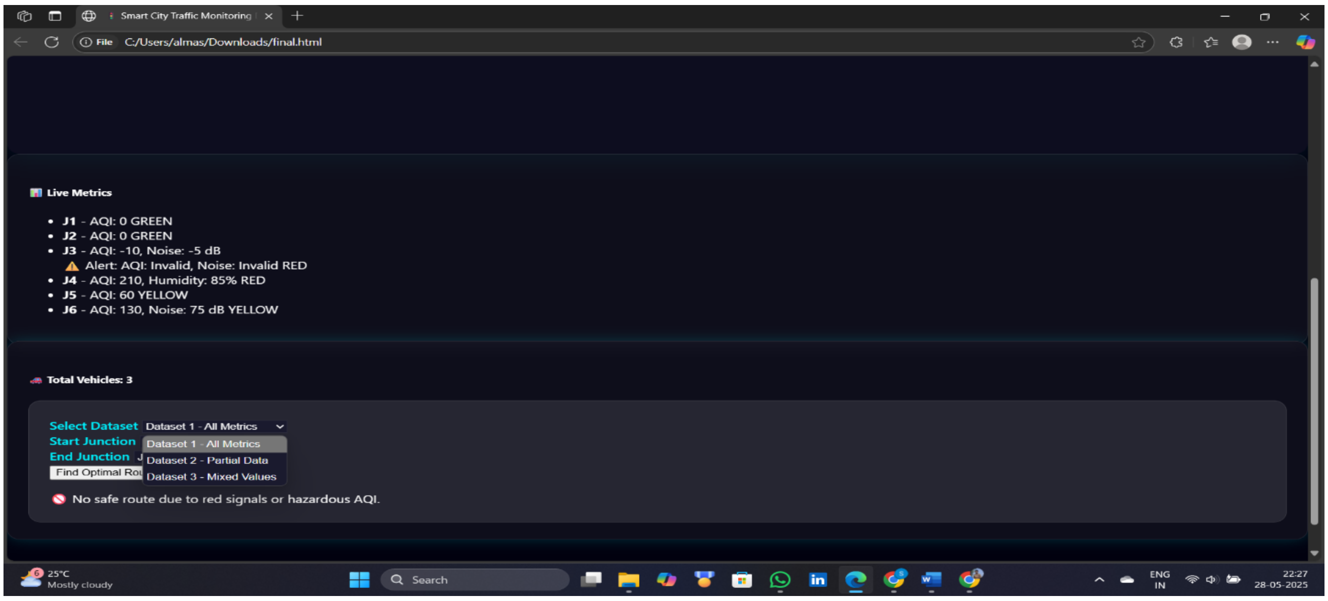


Live Metrics

- J1 - AQI: -20, Noise: 95 dB
  ⚠ Alert: AQI: Invalid RED
- J2 - AQI: 0 GREEN
- J3 - AQI: 250, Humidity: -5%
  ⚠ Alert: Humidity: Invalid RED
- J4 - AQI: 0 GREEN
- J5 - AQI: 100, Noise: 0 dB YELLOW
- J6 - AQI: 100, Humidity: 0% YELLOW

Figure 7.2.3: Test Case

Test Case 3:



File C:/Users/almas/Downloads/final.html

Live Metrics

- **J1** - AQI: 0 GREEN
- **J2** - AQI: 0 GREEN
- **J3** - AQI: -10, Noise: -5 dB
  ⚠ Alert: AQI: Invalid, Noise: Invalid RED
- **J4** - AQI: 210, Humidity: 85% RED
- **J5** - AQI: 60 YELLOW
- **J6** - AQI: 130, Noise: 75 dB YELLOW

Total Vehicles: 3

Select Dataset Dataset 1 - All Metrics
Start Junction
End Junction
Find Optimal Rou

Dataset 1 - All Metrics
Dataset 2 - Partial Data
Dataset 3 - Mixed Values

No safe route due to red signals or hazardous AQI.

Figure 7.2.4: Test Case

Test Case 4:

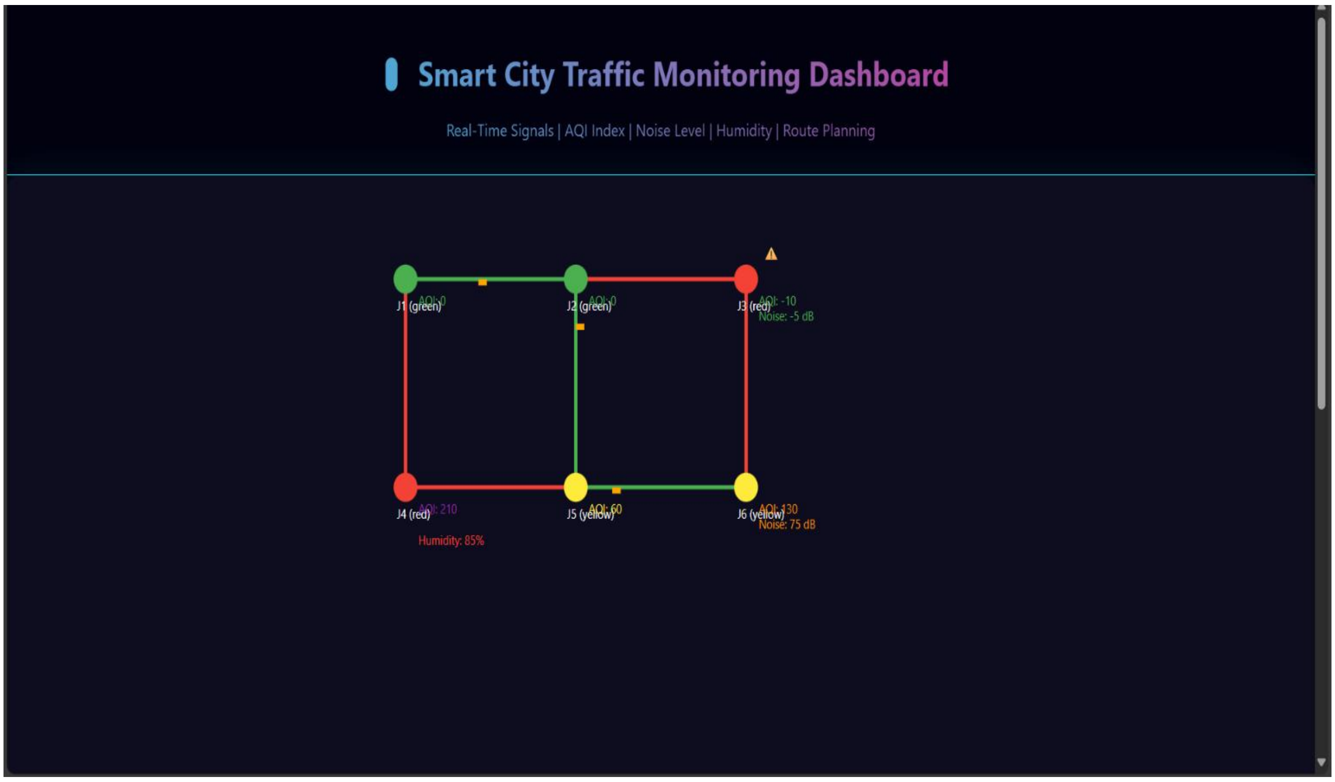


Figure 7.2.5: Test Cases

Test Case 5:

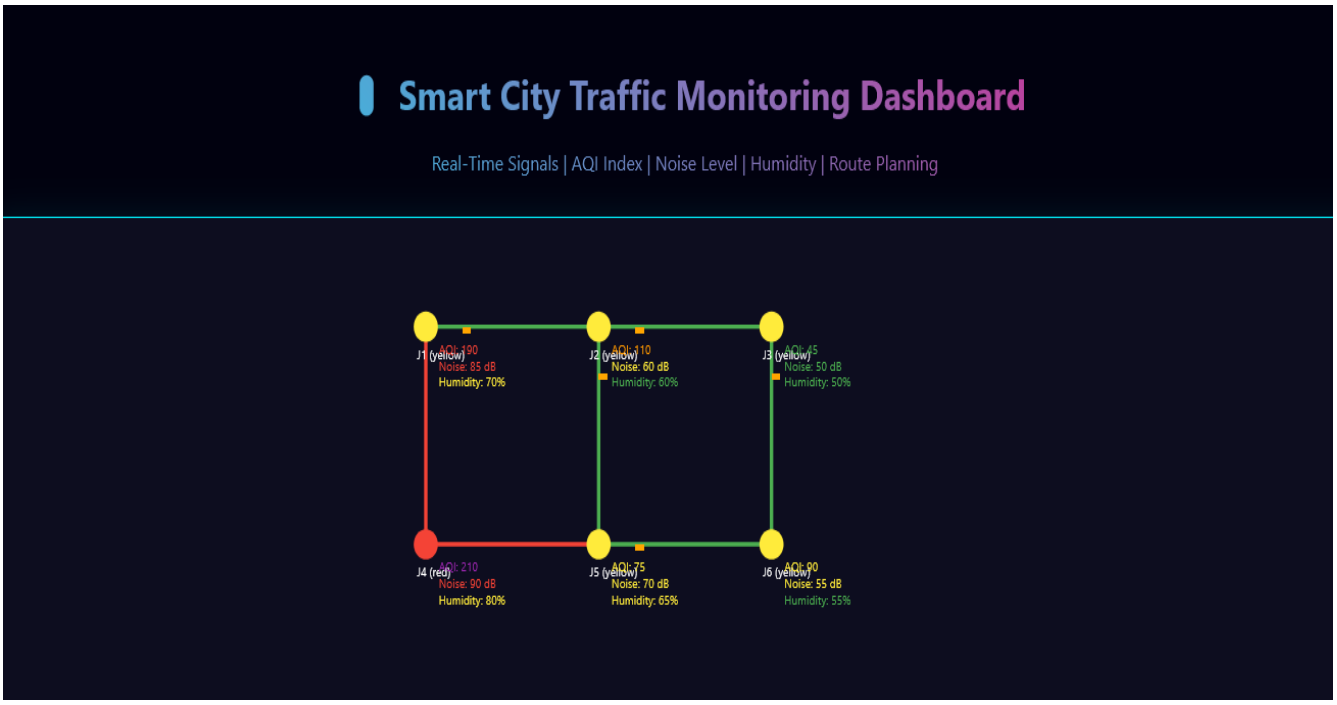
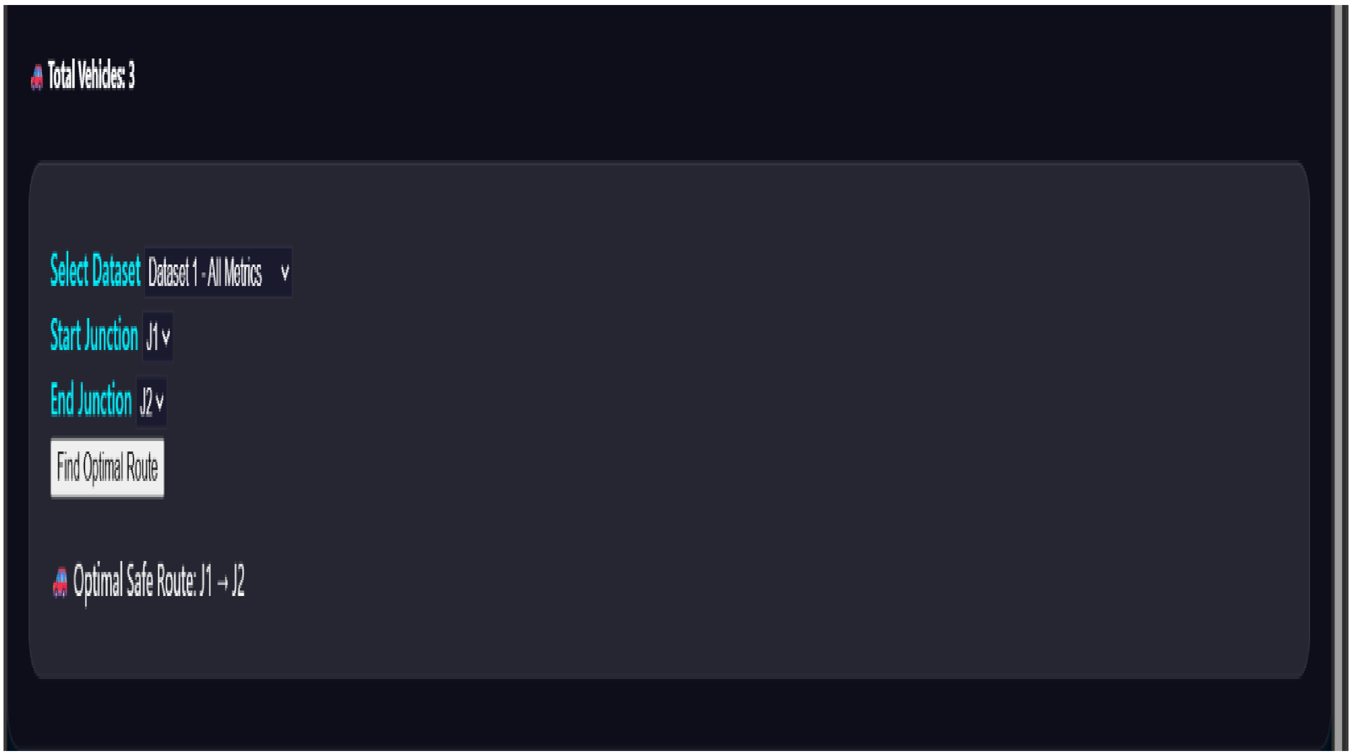


Figure 7.2.6: Test Cases

Test Case 6:



Total Vehicles: 3

Select Dataset: Dataset 1 - All Metrics

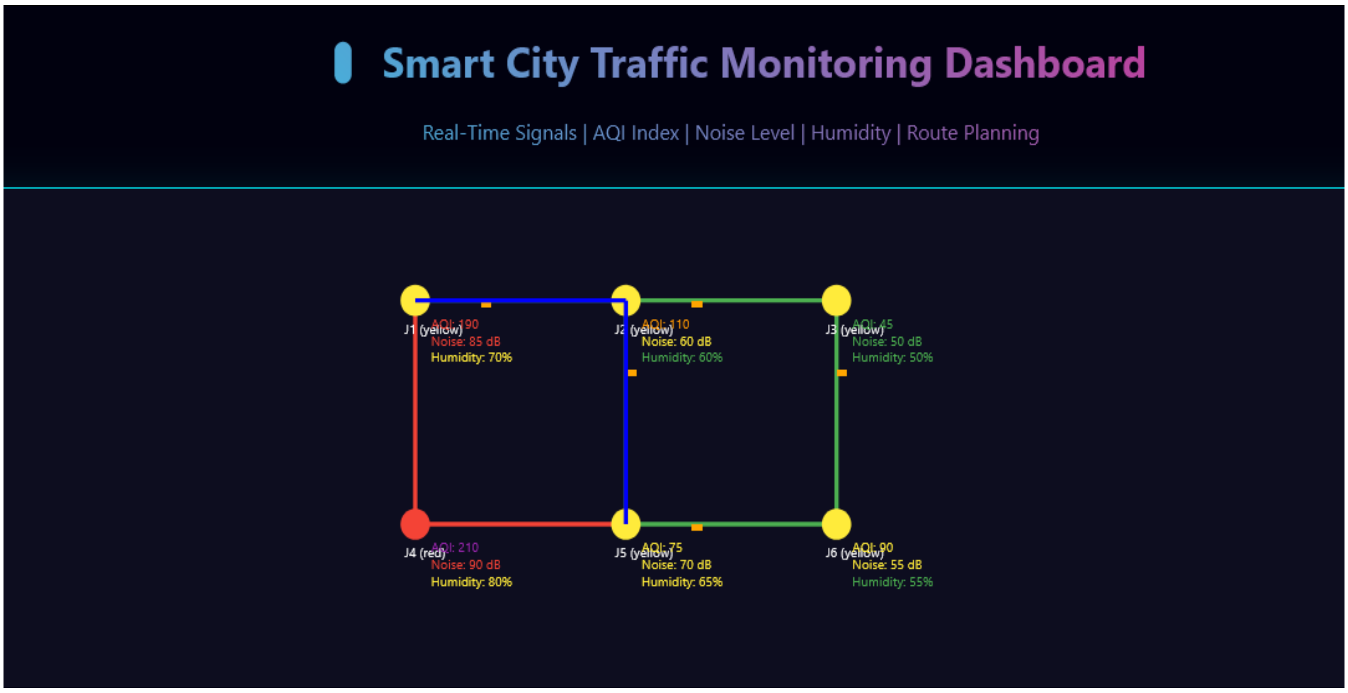
Start Junction: J1

End Junction: J2

Optimal Safe Route: J1 → J2

Figure 7.2.7: Test Cases

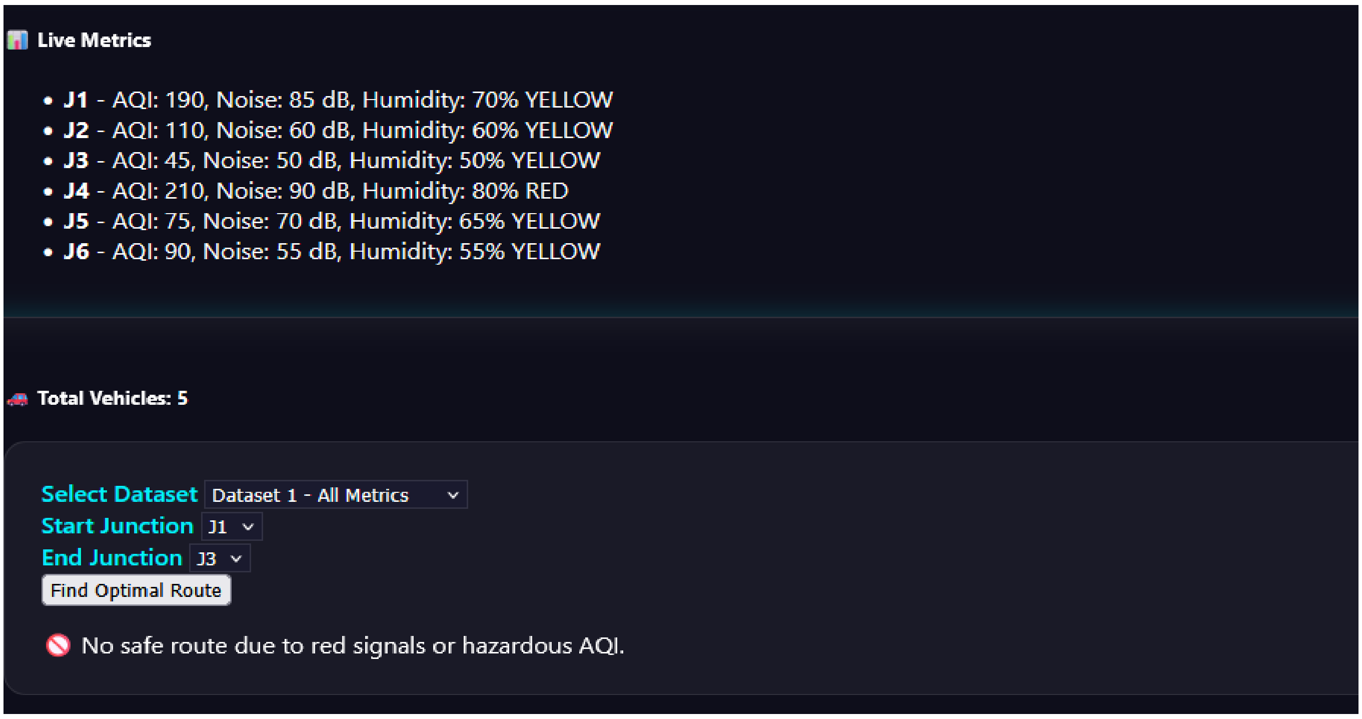
Test Case 7:


Smart City Traffic Monitoring Dashboard

Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning

Figure 7.2.8: Test Cases

Test Case 8:


Live Metrics

- **J1** - AQI: 190, Noise: 85 dB, Humidity: 70% YELLOW
- **J2** - AQI: 110, Noise: 60 dB, Humidity: 60% YELLOW
- **J3** - AQI: 45, Noise: 50 dB, Humidity: 50% YELLOW
- **J4** - AQI: 210, Noise: 90 dB, Humidity: 80% RED
- **J5** - AQI: 75, Noise: 70 dB, Humidity: 65% YELLOW
- **J6** - AQI: 90, Noise: 55 dB, Humidity: 55% YELLOW

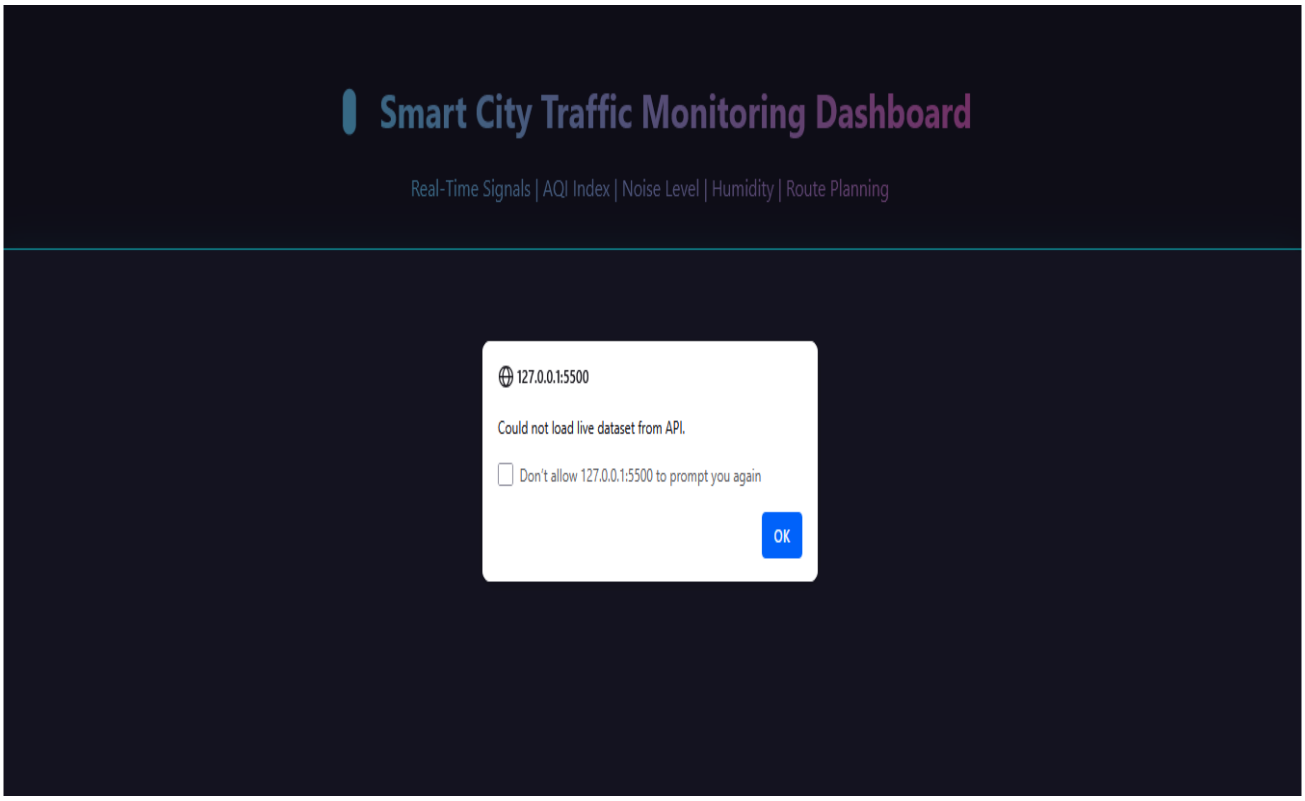
Total Vehicles: 5

Select Dataset: Dataset 1 - All Metrics
Start Junction: J1
End Junction: J3

🚫 No safe route due to red signals or hazardous AQI.

Figure 7.2.9: Test case

Test Case 9:


Smart City Traffic Monitoring Dashboard

Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning

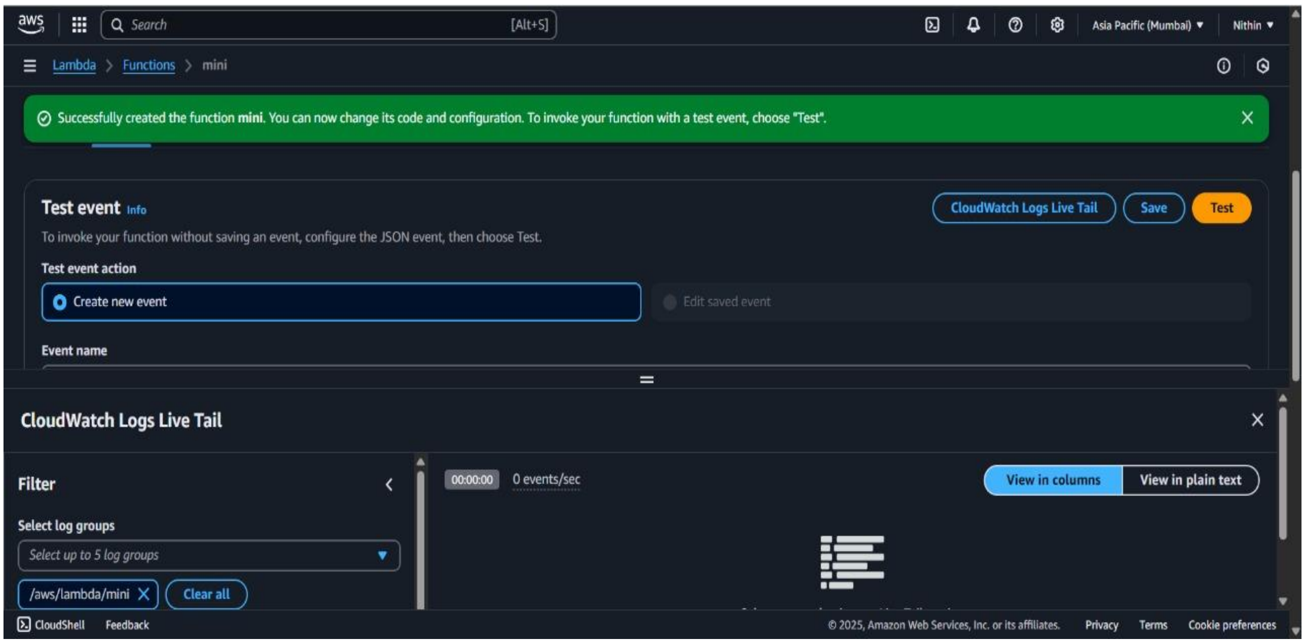
127.0.0.1:5500

Could not load live dataset from API.

Don't allow 127.0.0.1:5500 to prompt you again

Figure 7.2.10: Test case

Test Case 10:


Successfully created the function **mini**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Test event

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action: Create new event

CloudWatch Logs Live Tail

Select log groups: /aws/lambda/mini

00:00:00 0 events/sec

Figure 7.2.11: Test case

Test Case 11:

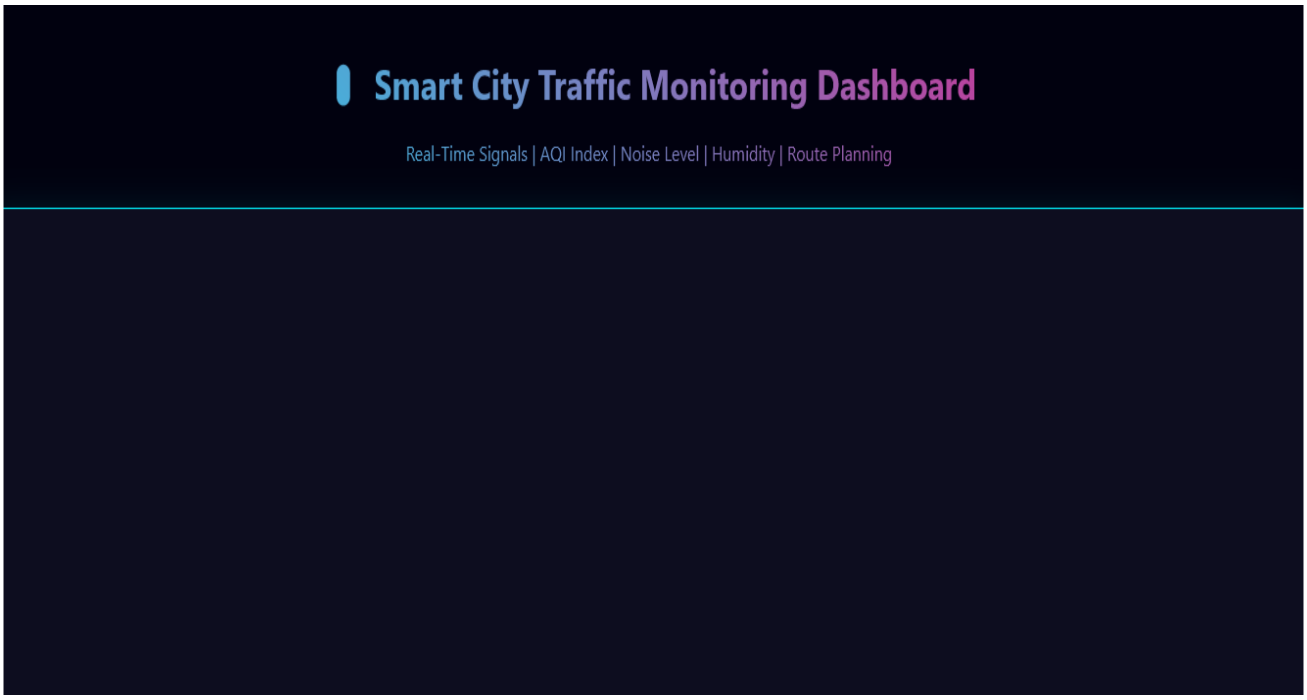
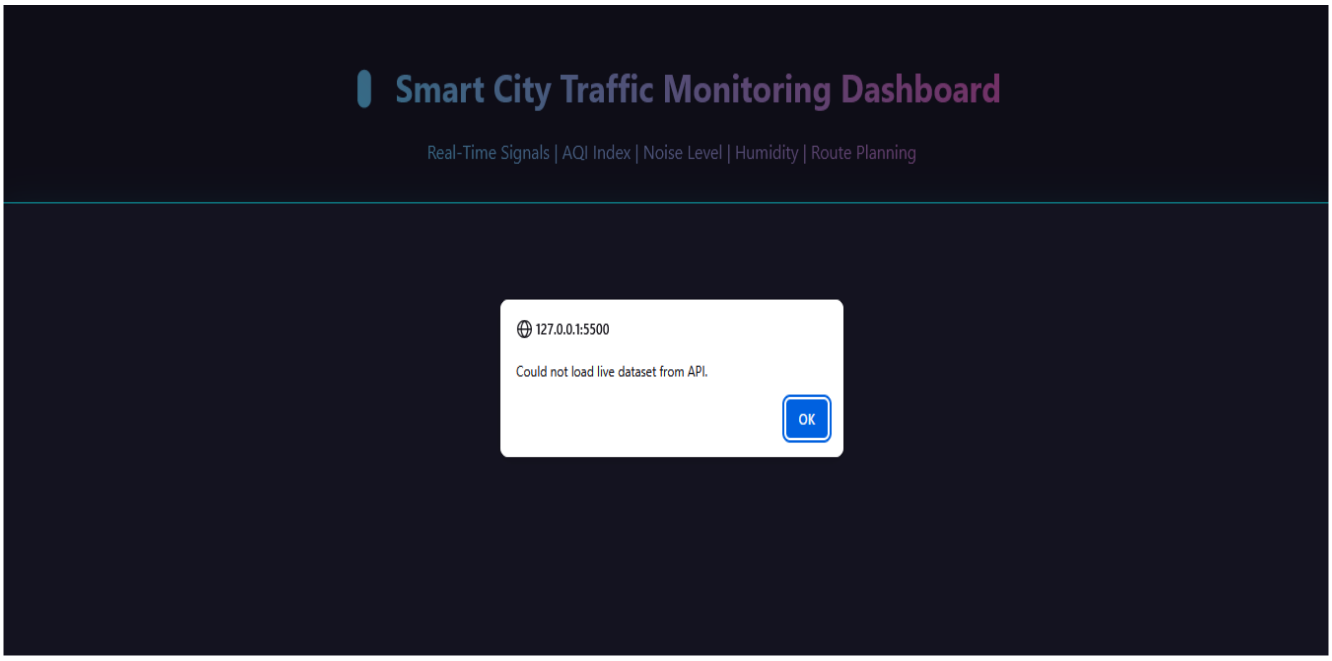

Smart City Traffic Monitoring Dashboard
Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning

Figure 7.2.12: Test case

8. OUTPUT SCREENS

Figure 8.1: Before fetching Data



Smart City Traffic Monitoring Dashboard

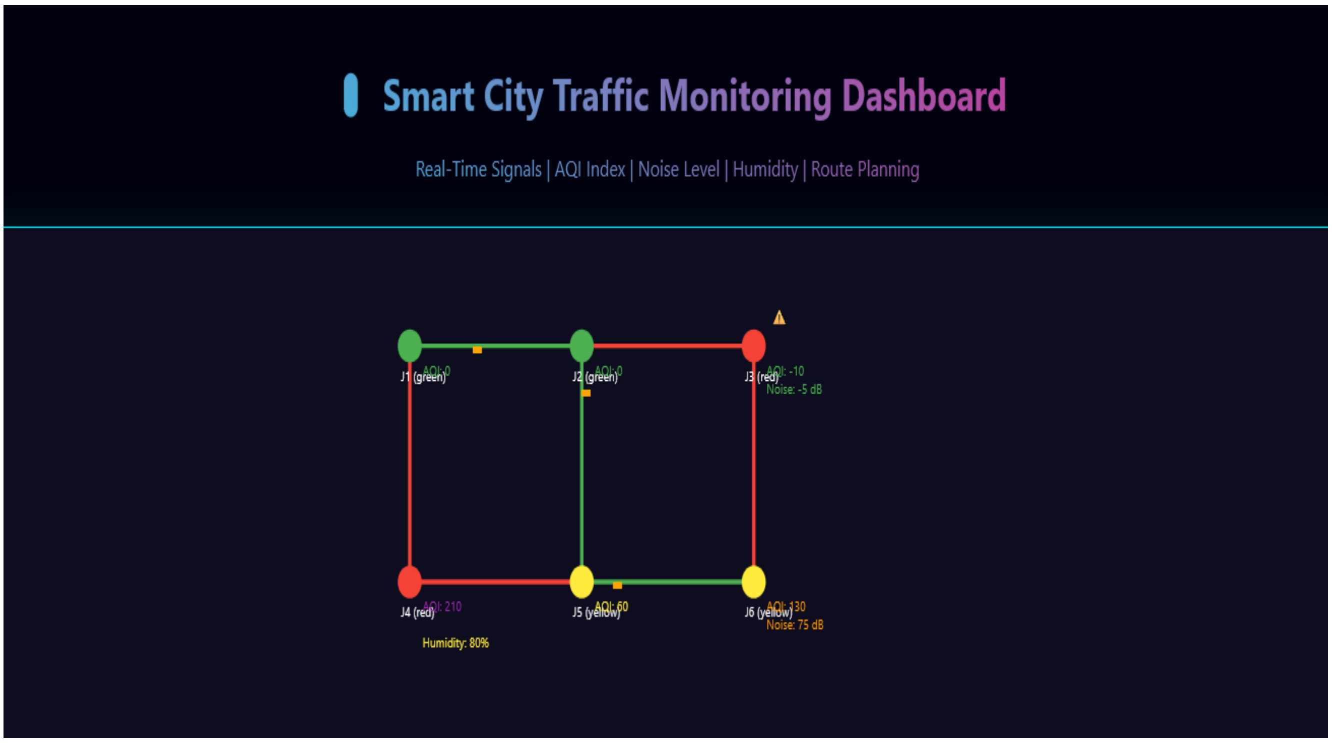
Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning

127.0.0.1:5500

Could not load live dataset from API.

OK

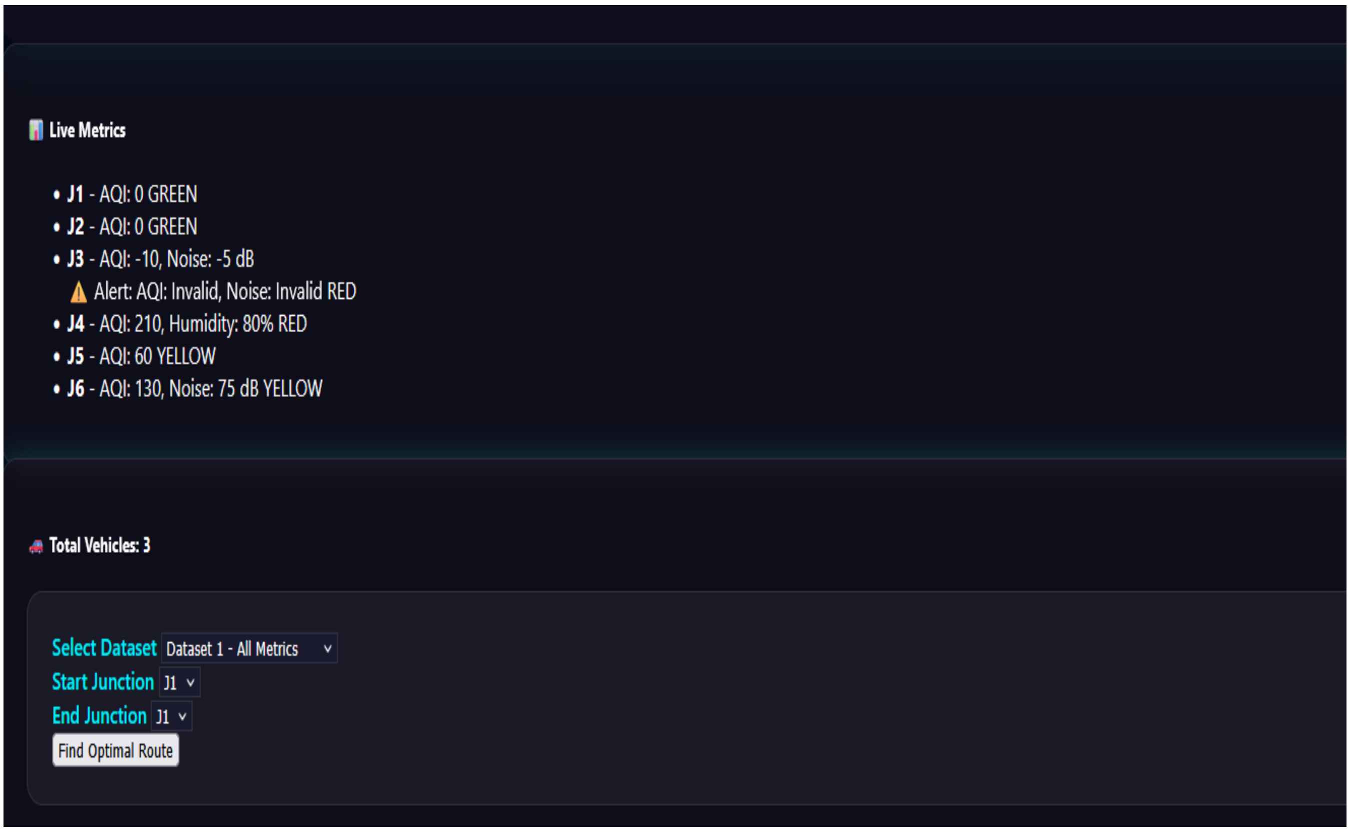
Figure 8.2: After fetching Data



Smart City Traffic Monitoring Dashboard

Real-Time Signals | AQI Index | Noise Level | Humidity | Route Planning

Figure 8.2.1: After fetching Data junction



Live Metrics

- **J1** - AQI: 0 GREEN
- **J2** - AQI: 0 GREEN
- **J3** - AQI: -10, Noise: -5 dB
  ⚠ Alert: AQI: Invalid, Noise: Invalid RED
- **J4** - AQI: 210, Humidity: 80% RED
- **J5** - AQI: 60 YELLOW
- **J6** - AQI: 130, Noise: 75 dB YELLOW

Total Vehicles: 3

Select Dataset: Dataset 1 - All Metrics
Start Junction: J1
End Junction: J1

9. CONCLUSION

The Smart City Infrastructure Monitoring and Traffic Control system successfully demonstrates how modern IoT devices and cloud computing services can be combined to create an intelligent, responsive urban management solution. By deploying an extensive network of sensors that continuously monitor critical parameters such as air quality index (AQI), noise pollution, humidity, and vehicular traffic, the project provides a real-time, data-driven overview of the city's environmental and traffic conditions. Leveraging AWS cloud services—including API Gateway for secure API management, Lambda for serverless data processing, S3 for scalable storage, DynamoDB for efficient database operations, and SNS for instant alert notifications—ensures a robust, flexible, and cost-effective backend infrastructure. The serverless architecture enables the system to scale automatically based on data input volume, reducing operational overhead and costs. The interactive frontend dashboard visualizes live sensor data with clear graphical representations of traffic flows, environmental metrics, and alerts. This empowers city administrators, traffic controllers, and citizens with actionable insights, allowing timely interventions such as traffic signal adjustments and public health advisories. The automated alert system enhances safety by promptly notifying relevant authorities and the public when sensor readings exceed critical thresholds, ensuring rapid responses to pollution spikes or traffic congestions. Furthermore, the system's modular design supports easy integration of additional sensors and future functionalities, such as AI-based traffic prediction, machine learning-driven anomaly detection, and advanced data analytics for long-term urban planning. This project exemplifies the practical application of IoT and cloud technologies to address urban challenges, improving quality of life, reducing environmental hazards, and optimizing traffic management. It sets a scalable framework for smart city initiatives, encouraging sustainable development and efficient resource utilization.

10 FUTURE ENHANCEMENTS

To further elevate the capabilities of the Smart City Infrastructure Monitoring and Traffic Control system, integrating advanced machine learning and artificial intelligence (AI) algorithms can provide predictive analytics and automated decision-making. For example, AI models can analyze historical traffic and environmental data to forecast congestion patterns, pollution spikes, or hazardous weather conditions. This predictive insight would enable proactive measures such as dynamic traffic light adjustments or preemptive public warnings, improving urban mobility and public health outcomes.

Another important enhancement is expanding the sensor network to cover additional environmental factors such as temperature fluctuations, carbon monoxide levels, and pedestrian density. Incorporating more diverse data points would offer a comprehensive view of urban conditions and help city planners make more informed decisions. Additionally, deploying edge computing devices at sensor nodes could reduce data transmission latency, allowing for faster, localized responses to critical events without always relying on cloud processing.

Security and privacy are paramount in smart city applications. Future versions of the system should incorporate robust encryption protocols, secure authentication mechanisms, and data anonymization techniques to protect sensitive citizen information and prevent unauthorized access to the infrastructure. Implementing blockchain-based audit trails could further enhance data integrity and transparency, fostering trust among users and stakeholders.

Moreover, expanding the user interface to include mobile applications and integration with popular navigation platforms would increase accessibility and convenience for citizens. Personalized notifications about traffic conditions, air quality alerts, and suggested alternative routes can empower individuals to make safer and healthier travel choices in real time. Coupled with crowd-sourced data inputs, the system could continuously improve accuracy and responsiveness.

Finally, the system can be extended to include energy management features, such as adaptive street lighting based on pedestrian presence and ambient light conditions, contributing to energy savings and sustainability goals. Integration with public transportation schedules and smart parking solutions would further enhance the overall efficiency of urban transit, reducing carbon footprints and improving the quality of urban life.

11 REFERENCES

[1] Jeremy Reed; Ali Tosun; Turgay Korkmaz; Benjamin Topolosky. “ IoT Device Classification in Edge-Cloud Models” Publisher: IEEE 20 May 2025.

[2] S Shahul Hammed; C. Preethi; K Haripriya; S. Pavalarajan; M.R. Gowtham; N. Akshaay Krithick. “Traffic and Pollution Control Using IoT-Enabled Smart Routing Algorithm”. Publisher: IEEE 25 April 2025.

[3] P. Chinnasamy; Patan Feridoz Khan; C Sasikala; Mavalluru Swathi; M Prema Kumar; Ajay Kumar. “Industrial IoT Smart Manufacturing Systems based on Cloud Computing Energy Efficiency Analysis”.Publisher: IEEE 02 April 2025.

[4] Zeinab Nazemi Absardi ; Reza Javidan.“A QoS-Aware Traffic Management Policy for IoT-enabled Smart City Applications based on Cloud Computing”. Publisher: IEEE 21 November 2025.

[5] Ahgaly Subbiah. “AI-Enhanced IoT System for Efficient Traffic Management: Leveraging Black Data in Smart Cities”. Publisher: IEEE 14 February 2025.

[6] Vijay Gurbade; Prateek Verma; Swapnil Gundewar; Vijendra Singh Bramhe “Building a Data Lake for Power BI in the Cloud: A Review on Utilizing Cloud Storage Services for Large Datasets” Publisher: IEEE 20 January 2025.

[7] Anurag Pal; Prince Jay Shankar; Ananya Das; P Yellamma. Efficient Data Storage Algorithm for IoT in Cloud-Distributed Environments. Publisher: IEEE 02 April 2025.

[8] S. Arul; P. Kavitha; S. Kamalakkannan. “Innovative Solutions for Sustainable IoT in Smart City Infrastructure”.Publisher: IEEE 24 October 2024.

[9] Kariuki Pius Muriuki; Japheth Odiwour Okello; Joan Chepkoech. “Advanced Intelligent Traffic Management System(AITMS)”. Publisher:IEEE 28 November 2024 .

[10] A.C. Santha Sheela ; B.Shamreen Ahamed; D. Poornima; Charles Sagayaraj. “Integration of Wireless Sensor Networks with IoT and Traffic Management”. IEEE21 February 2024.

[11] Santosh Kumar; Sandip Kumar Goyal. “A Comparative Study on Reputation Management in Social Cloud: Validating Our Proposed Trust Architecture with Multiple Datasets” Publisher: IEEE 19 July 2024.

[12] Toushif Hossain; Riyad Hossain; Rubaiyat Islam; Saadia Binte Alam. “Optimizing Warehouse Operations in Bangladesh: Leveraging IoT and Cloud Migration for Enhanced Security and Efficiency”. Publlisher: IEEE 08 November 2024.

[13] P. Rajakumar; N. Partheeban; Ms. Indervati; Himanshu Kumar; Pius Mlawa; Utkarsh Gautam. “Client-Level Monitoring Approach to Secure Data in a Cloud Environment”. Publisher: IEEE 26 January 2024.

[14] Lin Dewei; Jiang Meiling; Zhang Helin; Xu Yiming; Yan An. “Research on Data Operation Monitoring and Analysis System of Computer Intelligent Cloud Platform”. Publisher: IEEE 23 January 2024.

[15] Nada Masood Mirza; Adnan Ali; Nura Shifa; Mohamad Khairi Ishak; Khalid Ammar. “Predictive Modeling for Smart Traffic Systems: Harnessing IoT Data Insights”.Publisher: IEEE 8 March 2024.

[16] N Renuka; M Panneer Selvam; P Ponmurugan; S Kiruthika; N Selvam; N Kannan. “Automated Email Notifications and Security Monitoring for Cloud Storage Access using Cloud Function”. Publisher: IEEE 18 April 2024.

[17] T. Pflanzner; K. Zs. Leszko; A. Kertesz. “SUMMON: Gathering smart city data to support IoT-Fog-Cloud simulations”. Publisher: IEEE 31 May 2018.

[18] Dong Chen; Phuthipong Bovornkeeratiroj; David Irwin; Prashant Shenoy. “Private Memoirs of IoT Devices: Safeguarding User Privacy in the IoT Era”. Publisher: IEEE 23 July 2018.

[19] Ardi Imawan; Joonho Kwon. “A timeline visualization system for road traffic big data”.Publisher: IEEE 28 December 2020.

[20] Shih-Chi Lin; Po-Hsuan Wu; Hsin-Tse Lu; Wei-Jen Wang. “Dynamic throttling for IoT streaming hub services on multi-tenant cloud environment”. Publisher: IEEE 25 June 2018.