Real-time Traffic Analytics Using Stream Analytics

for

DEPI Graduation Project

Prepared by Abd El-Hakim Abdelaziz, Ahmed Mostafa, Sarah Mohamed, Shahd Mohamed ,and Wessal Osama

Table of Contents

1. Project Planning & Management

1.1 Project Proposal

Project Title: Real-Time Traffic Analytics Using Azure Stream Analytics **Overview:**
This project aims to design and implement a real-time traffic analytics system capable of monitoring live traffic data, detecting congestion, and providing actionable insights through dashboards. The system leverages Microsoft Azure's cloud ecosystem—particularly Event Hubs, Stream Analytics, SQL Database, and Power BI—to ensure scalability, reliability, and real-time performance.

Objectives:

- Develop an automated data pipeline for continuous traffic data ingestion and processing.
- Enable real-time detection of traffic congestion and anomalies.
- Provide dynamic visualization and alerting capabilities for end-users.
- Demonstrate practical application of cloud-based streaming analytics and IoT concepts.

Scope:

The system covers the full data lifecycle—from data collection and streaming to real-time analytics and visualization—integrating multiple Azure services for seamless cloud operation.

1.2 Project Plan

The project follows a structured timeline divided into five main phases over approximately 12 weeks. Each phase includes deliverables and checkpoints for evaluation.

Phase	Duration	Milestones	Key Deliverables
Phase 1: Project Initiation	Week 1–2	Define requirements, system architecture	Project proposal, architecture draft
Phase 2: Data Ingestion Setup	Week 3–4	API integration with Azure Event Hubs	Working ingestion script, event streaming
Phase 3: Real-Time Processing	Week 5–6	Stream Analytics job configuration	Queries for congestion detection
Phase 4: Data Storage & Visualization	Week 7–9	Connect Stream Analytics to Azure SQL/Power BI	Live dashboard with metrics and alerts
Phase 5: Testing & Finalization	Week 10–12	End-to-end testing, documentation, presentation	Final report, Power BI dashboard

Resource Allocation:

The team utilizes shared Azure resources (Event Hubs, Stream Analytics jobs, SQL DB) and development tools (Power BI, Python environment, VS Code) under the DEPI student subscription.

1.3 Task Assignment & Roles

Team Member	Role	Responsibilities
Abdelhakim Abdelaziz	Data Engineer / Team Lead	Project coordination, Azure Stream Analytics setup, documentation management

Team Member	Role	Responsibilities
Ahmed Mostafa	Backend Developer	API integration, data ingestion scripts, Event Hub configuration
Sarah Mohamed	Data Analyst	Dashboard creation, Power BI visualization, metrics definition
Shahd Mohamed	Data Scientist	Designing Stream Analytics queries, alert logic, anomaly detection
Wessal Osama	QA & Testing Engineer	System validation, performance testing, preparing testing documentation

---

1.4 Risk Assessment & Mitigation Plan

Risk	Description	Likelihood	Mitigation Strategy
API Downtime or Rate Limits	External traffic API may become unavailable or restricted.	Medium	Implement data caching and retry logic; use backup data source.
Stream Analytics Job Failure	Query errors or job interruptions during real-time processing.	Medium	Enable job auto-restart and configure logging in Azure Monitor.
Storage Overload or Cost Overrun	Excessive data volume increases Azure costs.	Low	Apply data retention policies; store summarized data.
Team Coordination Challenges	Miscommunication or delay in deliverables.	Low	Weekly progress meetings; task tracking using shared Kanban board.
Dashboard Performance Issues	Slow rendering with large datasets.	Medium	Optimize queries and enable incremental data loading in Power BI.

1.5 Key Performance Indicators (KPIs)

To evaluate the effectiveness and performance of the project, the following KPIs are defined:

KPI	Target Metric	Measurement Method
System Uptime	≥ 99%	Azure Monitor uptime reports
Data Latency	< 10 seconds delay from ingestion to visualization	Stream Analytics job metrics
Alert Accuracy	≥ 95% of alerts correctly triggered	Manual validation during testing
Dashboard Refresh Rate	Real-time (auto-refresh every 5 seconds)	Power BI settings and performance logs
User Interaction	100+ successful dashboard sessions	Power BI usage metrics

---

Summary:
This plan ensures structured project execution, balanced workload distribution, proactive risk mitigation, and measurable success indicators. It forms the foundation for delivering a functional and scalable real-time traffic analytics system.

2. Lecturer Review

This section provides an academic assessment of the project **"Real-Time Traffic Analytics Using Azure Stream Analytics"**, focusing on how effectively it meets its objectives, applies cloud-based data engineering concepts, and demonstrates end-to- end implementation of a real-time analytics solution.

Feedback & Evaluation
The project presents a well-structured and technically sound approach to real-time data processing using Microsoft Azure services. It successfully simulates live traffic data, processes it through Azure Stream Analytics, and visualizes meaningful insights through Power BI or Streamlit dashboards. Overall, the system reflects a strong grasp of streaming analytics, IoT concepts, and cloud integration.

Key Strengths:

- **Comprehensive Pipeline Design:**

The project covers the full data lifecycle — from data generation and ingestion to realtime analysis and visualization. This holistic approach demonstrates a solid understanding of modern data engineering workflows.

- **Effective Use of Azure Services:**

Seamless integration of Azure Event Hubs, Stream Analytics, and visualization tools highlights proficiency in leveraging the Azure ecosystem for scalable and efficient solutions.

- **Real-World Relevance:**

The chosen use case — monitoring live traffic to detect congestion or anomalies — is highly practical and aligns well with real-world IoT and smart city applications.

- **Structured Milestones:**

The division of the project into clear phases (simulation, processing, visualization) helped maintain focus and ensured consistent progress toward the final objective.

Areas for Improvement:

- **Data Simulation Depth:**

The simulation could be expanded to include richer contextual data, such as vehicle types, lane occupancy, or weather conditions, making the dataset more realistic and valuable for analysis.

- **Alerting Logic:**

The current alerting system relies primarily on static thresholds. Integrating adaptive or intelligent anomaly detection methods (e.g., ML-based detection) would make the solution more robust and scalable.

- **Scalability and Optimization:**

Future iterations should include a discussion on scaling the architecture, optimizing performance, and managing costs within Azure — essential considerations for production-level deployments.

Suggested Improvements

To strengthen the system's capabilities and enhance its professional depth, the following improvements are recommended:

1. **Implement Intelligent Anomaly Detection:**

Replace fixed thresholds with dynamic models using Azure Anomaly Detector or a custom machine learning model integrated into the Stream Analytics pipeline.

2. **Enhance Data Context:**

Integrate external data sources such as weather APIs or road condition data to create a more complete picture of traffic behavior and environmental influence.

3. **Add Historical and Predictive Insights:**

Extend the dashboard with a historical analytics tab that tracks long-term trends and uses simple forecasting models to predict traffic congestion patterns.

4. **Improve Monitoring and Logging:**

Set up Azure Monitor or Application Insights to track pipeline health, latency, and error rates — ensuring the system remains reliable and maintainable over time.

5. **Include Security Best Practices:**

Highlight the use of secure authentication methods (e.g., SAS tokens, managed identities) to reinforce good cloud security awareness and compliance practices

Final Grading Criteria

Evaluation Area	Description	Weight (%)
Documentation Quality	Clarity, completeness, and professionalism of written documentation	25%
Implementation Functionality	Technical accuracy, working solution, and pipeline performance	35%
Testing & Validation	Evidence of thorough testing, validation, and result accuracy	20%
Presentation & Reporting	Clarity, confidence, and effectiveness of project presentation	10%
Innovation & Improvement	Creativity in design, use of advanced features, and problem-solving approach	10%

3. Requirements Gathering

3.1 Stakeholder Analysis

The success of the *Real-Time Traffic Analytics Using Azure Stream Analytics* project depends on effectively addressing the needs of all involved stakeholders.

Stakeholder	**Role / Interest**	**Needs / Expectations**
Traffic Management Authorities	Primary users and decision-makers	Real-time insights into traffic congestion, alerts for critical incidents, and historical reporting for planning.
General Public / Commuters	End users accessing traffic dashboards	Clear, up-to-date visualization of traffic conditions and congestion alerts for better route planning.
Project Development Team	System designers, data engineers, and analysts	Seamless data pipeline, efficient stream processing, and system scalability.
Supervisors / Lecturers	Academic evaluators	Well-documented project implementation, demonstration of data engineering and analytics concepts.
Azure Cloud Platform	Service provider	Reliable infrastructure for ingestion, processing, and visualization of real-time data streams.

3.2 User Stories & Use Cases

To capture system functionality from the end-user's perspective, the following user stories and use cases are defined. **User Stories:**

1. **As a commuter**, I want to view live traffic data so that I can avoid congested routes.
2. **As a traffic authority officer**, I want to receive alerts when congestion exceeds normal thresholds, so I can take corrective action.
3. **As a data engineer**, I want to ingest live data from external APIs into Azure Event Hubs to ensure continuous streaming.
4. **As a system admin**, I want to monitor pipeline health to ensure the analytics system runs without interruption.
5. **As a project evaluator**, I want to visualize results and verify that real-time analytics are functioning as expected.

Use Case Summary:

Use Case	Actor	Description
View Live Dashboard	Public User	Displays real-time traffic conditions, congestion levels, and alerts via Power BI dashboard.
Receive Traffic Alerts	Public User / Authority	Generates automatic alerts when congestion thresholds are exceeded.
Ingest Live API Data	Data Engineer	Fetches live traffic data from API and streams it into Azure Event Hub.
Configure Stream Analytics	Data Engineer	Sets up Stream Analytics queries for real-time aggregation and pattern detection.
Monitor System Performance	System Admin	Tracks Stream Analytics job health, latency, and uptime through Azure Monitor.

3.3 Functional Requirements

The system must provide the following core functionalities:

Functional ID	Requirement	Description
FR-1	**Data Ingestion from API**	Retrieve live traffic data periodically from an external REST API.
FR-2	**Stream Processing**	Process incoming data using Azure Stream Analytics for real-time aggregation and filtering.
FR-3	**Data Storage**	Store processed data in Azure SQL Database or Data Lake for analysis and visualization.
FR-4	**Alert Generation**	Trigger alerts when congestion levels exceed predefined or dynamic thresholds.
FR-5	**Dashboard Visualization**	Present real-time and historical data insights using Power BI.
FR-6	**System Monitoring**	Continuously monitor data flow, query performance, and system uptime.
FR-7	**User Interaction**	Allow users to select locations and filter traffic data dynamically.

3.4 Non-Functional Requirements

These requirements ensure system performance, reliability, and usability under real-world conditions.

Category	Requirement	Description
Performance	Real-time data updates	The dashboard should reflect new traffic data within 5–10 seconds.
Scalability	Horizontal scaling	The pipeline should handle increasing data volume without performance loss.
Reliability	High availability	Stream Analytics and Event Hub jobs should maintain ≥99% uptime.

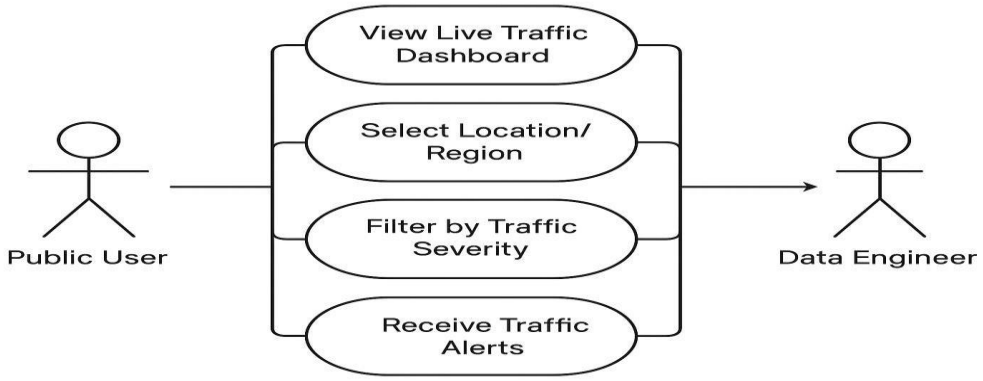
Security	Secure API access	Use encrypted API keys, SAS tokens, and Azure Managed Identities for authentication.
Usability	Intuitive dashboard design	Interface must be simple, visually clear, and accessible across devices.
Maintainability	Modular architecture	Each component (API ingestion, analytics, visualization) should be independently updatable.
Data Quality	Validity and completeness	Implement validation checks for missing or incorrect traffic data.
Cost Efficiency	Optimized resource usage	Use pay-as-you-go Azure tiers and stop idle services to minimize cost.

Summary:
This requirements specification defines what the system should achieve and how it should perform. It ensures that all stakeholders' needs are met while maintaining technical feasibility, performance, and scalability

4. System Analysis & Design

4.1 Problem Statement & Objectives – Define the problem being solved and project goals. ✞

Use Case Diagram & Descriptions



1. View Live Traffic Dashboard

Description:
Public users can view real-time traffic data visualized on a dashboard, including congestion levels, average speeds, and alerts.

Precondition:
System is connected to the traffic API and has received live data.

Postcondition:
User sees updated traffic conditions in their selected area.

Actor:

Public User (Primary Actor) **Main**
Success Scenario:

2. User opens the dashboard.
3. The system fetches and displays live traffic data.
4. User sees visualizations like maps, congestion levels, and speed indicators.

Alternate Scenario:
User's internet connection is lost or data refresh is delayed.
Exception:
API or Stream Analytics service fails to provide live data.

2. Select Location or Region

Description:
Public users can choose a city or region (e.g., Cairo, Giza) to view localized traffic information.
Precondition:
Dashboard is running; geocoding API is accessible.
Postcondition:
The dashboard displays traffic data for the chosen location.
Actor:
Public User (Primary Actor) **Main**
Success Scenario:

1. User selects a location from the dashboard.
2. System retrieves latitude/longitude using the Geocoding API.
3. System calls the Traffic API and refreshes the dashboard.

Alternate Scenario:
User enters an invalid or unknown location. **Exception:**
Geocoding API fails to resolve coordinates.

3. Receive Traffic Alerts

Description:
Public users receive notifications or visual alerts when congestion or incidents exceed a set threshold.
Precondition:
Thresholds for congestion are predefined.
Postcondition:
User is notified via the dashboard or email alert.
Actor:
Public User (Primary Actor) **Main**
Success Scenario:

1. System continuously monitors traffic stream data.
2. Congestion or incident threshold is exceeded.
3. Alert appears on the dashboard (or via email).

Alternate Scenario:
User disables or ignores alerts.
Exception:
Alert delivery fails due to configuration or API error.

4. **Ingest Live API Data**

Description:
System retrieves and stores traffic data from external APIs for real-time processing.
Precondition:
API key and endpoints are valid.
Postcondition:
Traffic data is successfully pushed into cloud storage or Event Hub.
Actor:
Data Engineer (Primary Actor) **Main**
Success Scenario:

1. Python script runs to fetch traffic data.
2. Data is streamed to Azure Blob Storage or Event Hub.
3. Logs confirm successful ingestion.

Alternate Scenario:
Data is retrieved but not streamed (offline logging instead).
Exception:
API rate limit exceeded or network timeout.

5. Configure Stream Analytics Queries

Description:
Data engineers create and manage real-time queries that detect congestion and calculate averages.
Precondition:
Event Hub or Blob contains ingested traffic data.
Postcondition:
Processed data is stored in Azure SQL or Data Lake.
Actor:
Data Engineer (Primary Actor) **Main**
Success Scenario:

1. Engineer sets up Stream Analytics job.

2. Defines SQL-like query for speed averages, incidents, etc.
3. System processes incoming data in real time.

Alternate Scenario:
Query syntax needs adjustments after testing.
Exception: Stream Analytics job fails or storage is unavailable.

✞ Functional & Non-Functional Requirements – Clearly state system capabilities and constraints.

Traffic congestion and inconsistent traffic flow continue to be significant challenges in modern urban environments. Existing systems often lack effective real-time analytics and automated alert mechanisms, resulting in delays, inefficiencies, and poor traffic management decisions.

This project aims to develop a **real-time traffic analytics solution** using **Azure Stream Analytics**, designed to process, analyze, and visualize live traffic data for improved monitoring and decision-making.

Objectives

- Collect and process live traffic data from external APIs.
- Apply real-time analytics to identify congestion and traffic flow patterns.
- Store processed insights in a cloud database for accessibility and long-term analysis.
- Develop a live **Power BI** dashboard for visualization and alerting.
- Demonstrate a scalable and efficient data engineering architecture on **Microsoft Azure**.

✞ Software Architecture – High-level design outlining system components, interactions, and architecture style (e.g., MVC, Microservices). The system is designed using a **cloud-based, event-driven architecture** leveraging various Azure services to ensure real-time processing, scalability, and fault tolerance.

Architecture Description

1. **Data Source:** Real-time traffic API providing JSON data on vehicle movement, speed, and congestion levels.

2. **Ingestion Layer: Azure Event Hub** receives continuous streams of traffic data.
3. **Processing Layer: Azure Stream Analytics** performs real-time aggregation and congestion detection.
4. **Storage Layer: Azure SQL Database** stores processed data for historical analysis and integration.
5. **Visualization Layer: Power BI** connects to the SQL database to display live dashboards and generate alerts.
   This architecture provides **low latency**, **high scalability**, and **reliable fault tolerance**, enabling near-instantaneous insights from incoming data streams.

4.2 Database Design & Data Modeling

Entity-Relationship Model (ERD)

The database is optimized for efficient storage and querying of continuous traffic streams. The primary entities are:

- **TrafficRecord:** Stores both raw and processed data, including timestamp, location, speed, and congestion level.
- **Alert:** Contains system-generated alerts with severity, timestamp, and alert message.
- **Region:** Holds static geographic information for each monitored area.

Relationships

- Each **Region** can have multiple **TrafficRecords**.
- Each **TrafficRecord** can trigger one or more **Alerts**.

Logical Schema

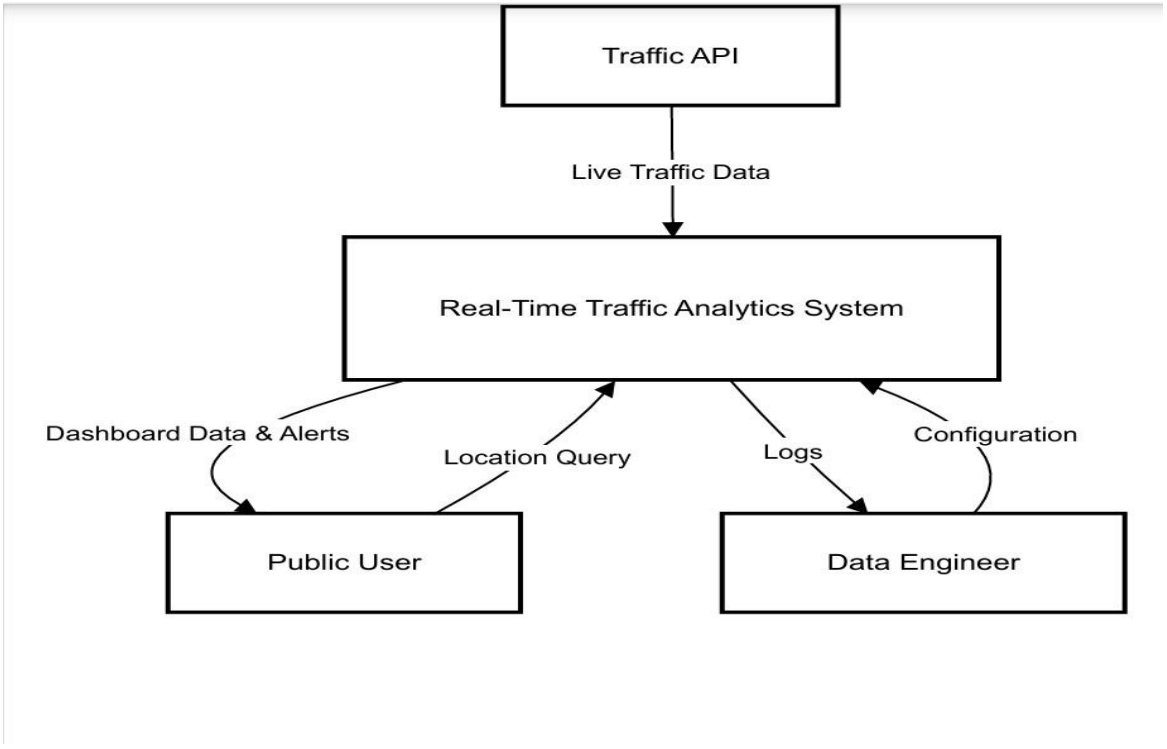
Table	Attributes	Primary Key	Foreign Key
Region	region_id, region_name, latitude, longitude	region_id	–
TrafficRecord	record_id, region_id, speed, congestion_level, timestamp	record_id	region_id
Alert	alert_id, record_id, alert_type, severity, timestamp	alert_id	record_id

Table	Attributes	Primary Key	Foreign Key
Region	region_id, region_name, latitude, longitude	region_id	–
TrafficRecord	record_id, region_id, speed, congestion_level, timestamp	record_id	region_id
Alert	alert_id, record_id, alert_type, severity, timestamp	alert_id	record_id

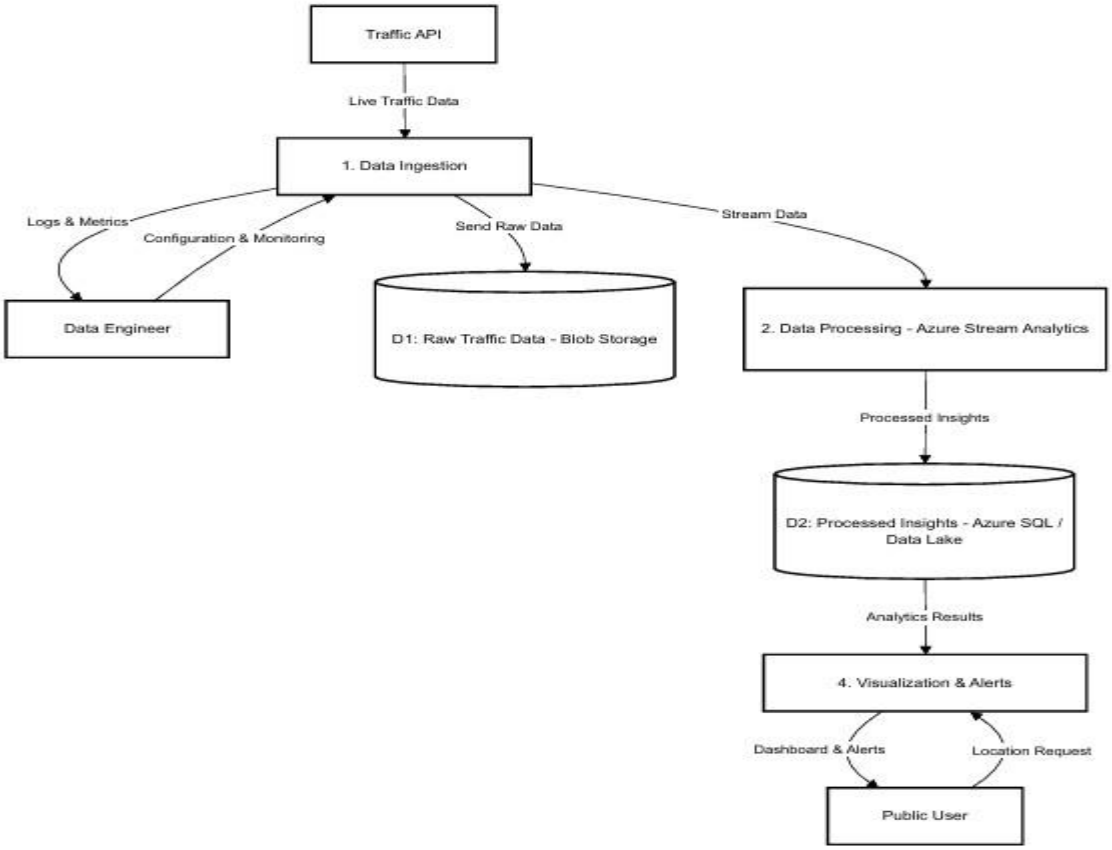
4.3 Data Flow & System Behavior

✞ DFD (Data Flow Diagram) ○

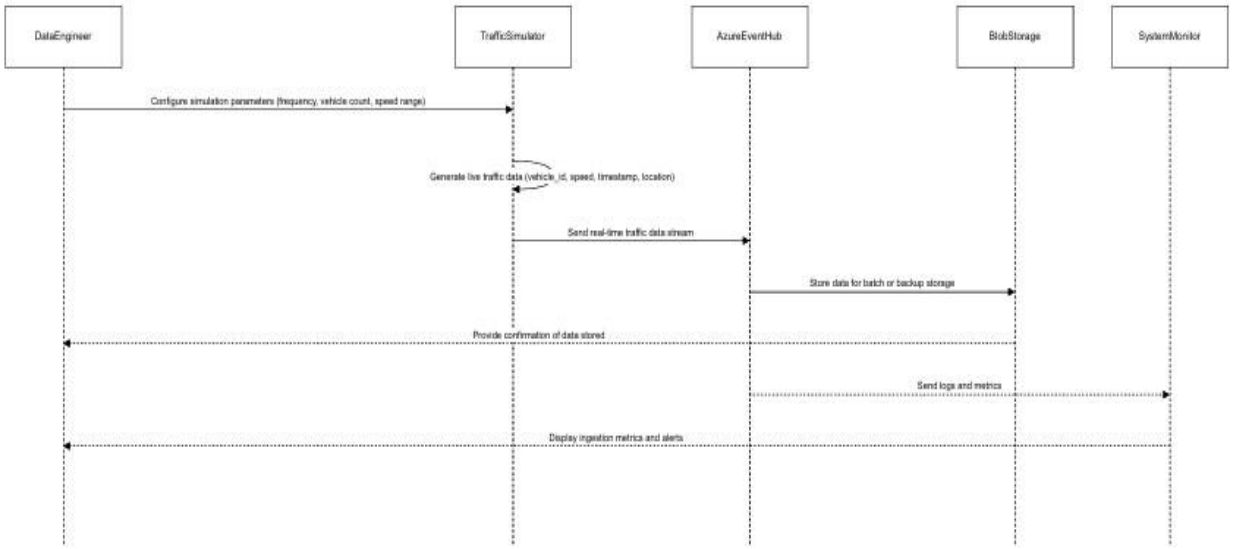
Level 0



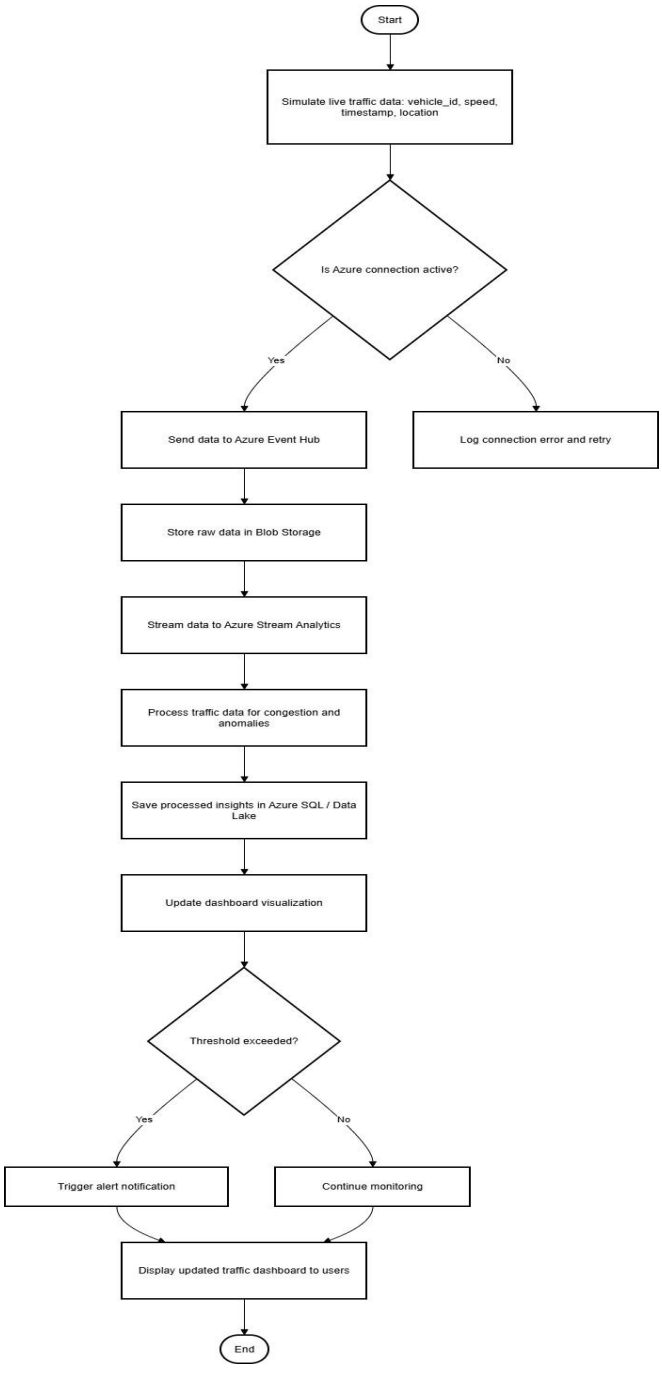
○ Level 1


Traffic API
Live Traffic Data
1. Data Ingestion
Logs & Metrics
Configuration & Monitoring
Send Raw Data
Stream Data
Data Engineer
D1: Raw Traffic Data - Blob Storage
2. Data Processing - Azure Stream Analytics
Processed Insights
D2: Processed Insights - Azure SQL / Data Lake
Analytics Results
4. Visualization & Alerts
Dashboard & Alerts
Location Request
Public User

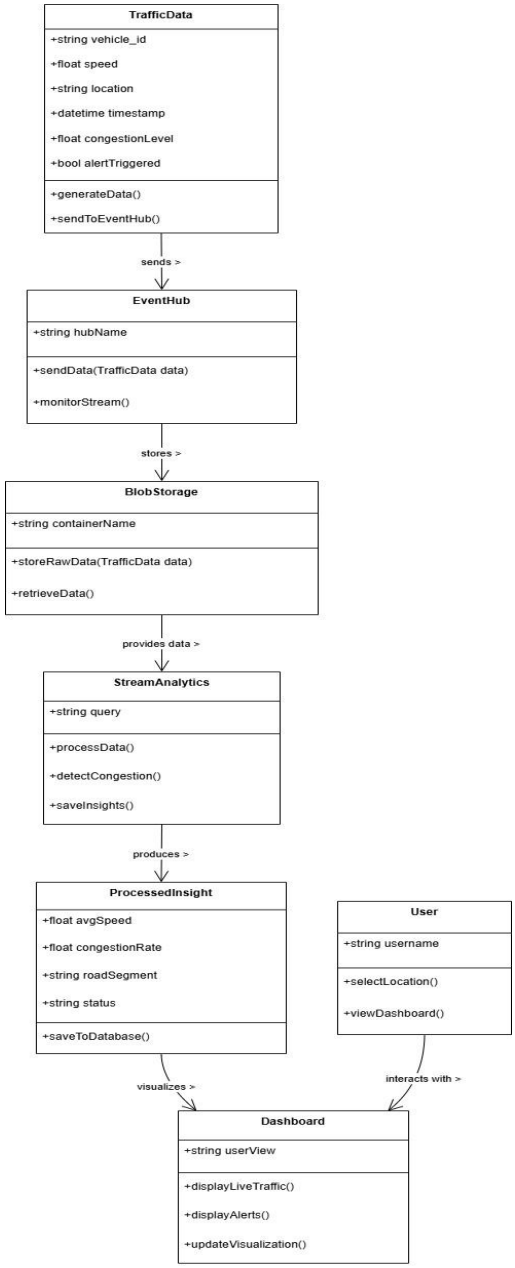
- Sequence Diagram



✞ Activity Diagram



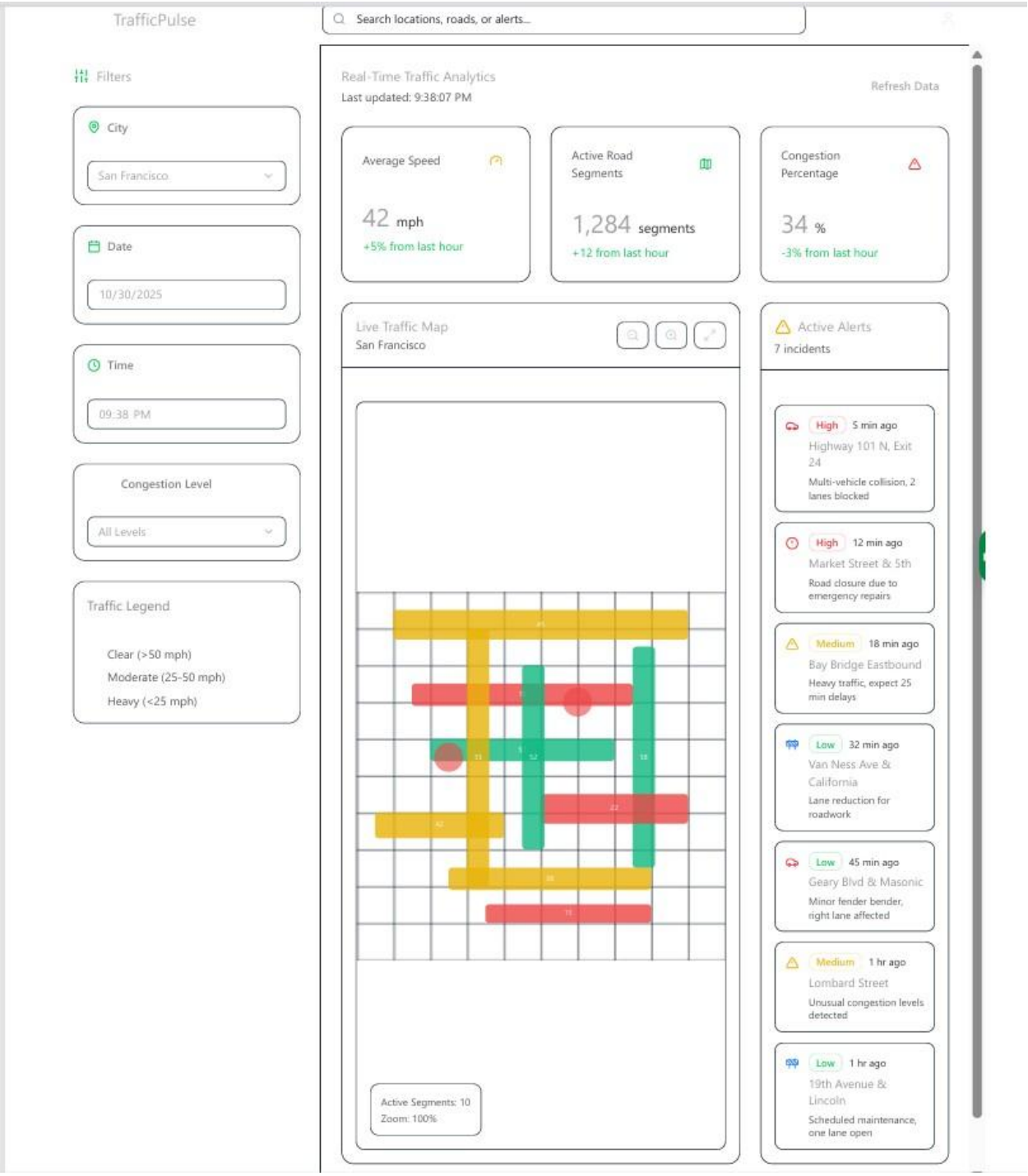
- Class Diagram



4.4 UI/UX Design & Prototyping

- Wireframes & Mockups

UI/UX Guidelines



1. Design Principles

- **Clarity & Simplicity:**
  Keep the interface intuitive; users should quickly understand traffic status at a glance. Avoid clutter.

- **Consistency:**

Uniform styling across navigation, buttons, cards, and maps. Consistent icons, padding, and font sizes help users scan the dashboard efficiently.

- **Visual Hierarchy:**
  Highlight the most important information (e.g., congestion alerts, map) using size, color, and position.
- **Feedback & Responsiveness:**
  Buttons, filters, and map interactions should give immediate visual feedback (hover, click states).
- **Real-Time Awareness:**
  Use subtle animations or color updates to reflect live traffic changes without overwhelming the user.

2. **Color Scheme**

- **Dark Theme Base:**
  - Background: #121212 (charcoal/black) ○ Panels & Cards: #1E1E1E (slightly lighter dark gray)
- **Highlights:**
  - Traffic Clear: Green (#4CAF50) ○ Moderate Congestion: Yellow (#FFEB3B)
  - Heavy Congestion: Red (#F44336)
- **UI Accents:**
  - Buttons, icons, and hover states: Neon green or light gray (#B0BEC5) ○ Alerts table: subtle borders or separators in gray (#424242)

3. **Typography**

- **Primary Font:** Roboto or Inter (sans-serif for readability)
- **Font Weights & Sizes:**
  - Headings: Bold, 18–24px
  - Summary Cards: Medium, 16px ○ Body / Table Text: Regular, 14px
- **Contrast:**

Ensure text has sufficient contrast against dark backgrounds (WCAG AA standard: ratio ≥ 4.5:1).

4. **Layout & Spacing**

- **Navigation:** Sticky top nav with padding 16–24px
- **Sidebars:** 250–300px width, enough padding for filters
- **Map Area:** Full remaining space; ensure responsive resizing
- **Cards & Panels:** Consistent padding (16px), subtle shadows for separation

5. **Accessibility Considerations**

- **Color Blindness:**
  Do not rely on color alone; add icons or patterns to indicate congestion levels.
- **Keyboard Navigation:**
  Users should navigate through filters, buttons, and map controls using Tab/Enter keys.
- **Screen Readers:**
  Provide aria-labels for buttons, alerts, and map elements.
- **Contrast & Font Size:**
  Maintain readability for all text; allow dynamic resizing where possible.

6. **Interactivity & Microinteractions**

- **Hover Effects:** Highlight buttons, map areas, and table rows on hover.
- **Refresh Button:** Animated spinning icon on click to indicate data reload.
- **Alert Notifications:** Subtle slide-in/out animations for new incidents.

7. **Footer**

- Keep minimal: “Data powered by TomTom API” in smaller, muted gray (#B0BEC5).

4.5 System Deployment & Integration

Technology Stack

- **Backend:** Python (for API integration and data ingestion)
- **Processing:** Azure Stream Analytics
- **Database:** Azure SQL Database

- **Visualization:** Microsoft Power BI
- **Cloud Infrastructure:** Azure Event Hubs, Data Lake, and Azure Monitor

Deployment Overview

Traffic data is fetched through a Python-based API client and pushed to **Azure Event Hubs**. Stream Analytics processes this data stream in real time, saving the results in an **Azure SQL Database**. Power BI connects to this database to provide live dashboards for traffic monitoring.

All components are deployed within a **single Azure subscription**, ensuring smooth integration, unified management, and secure data flow.

Component Integration

API Client → Event Hub → Stream Analytics → SQL Database → Power BI Dashboard → End User

4.6 Additional Deliverables

To ensure system completeness and usability, the following deliverables accompany the main implementation:

1. Documentation

- **System Design Document:** Includes architecture diagrams, workflows, and component details.
- **API Reference Manual:** Provides endpoint specifications, authentication details, and data formats.
- **User Guide:** Describes how to access and interact with the Power BI dashboard.

2. Testing and Validation

- **Unit Testing:** Conducted on Python scripts for data ingestion and transformation.
- **Integration Testing:** Ensures smooth data transfer between Event Hubs, Stream Analytics, and SQL Database.
- **Performance Testing:** Measures system throughput and latency under simulated traffic loads.

3. Monitoring and Logging

- Integration with **Azure Monitor** and **Application Insights** for system health tracking.
- Automated alerts for performance issues or data anomalies.

4. Security and Compliance

- Implementation of **Role-Based Access Control (RBAC)** for all users.
- Data encryption (in transit and at rest) with **Azure-managed keys**.
- Adherence to **GDPR** and cloud data privacy standards.

5. Future Enhancements

- Predictive traffic congestion modeling with **Azure Machine Learning**.
- Development of a **mobile dashboard** for remote monitoring.
- Integration of **IoT sensor data** for more granular traffic insights.

4.6 Additional Deliverables (if applicable)

API Documentation

Overview

The project integrates multiple external APIs to collect live traffic and weather data in real time. The APIs provide geolocation, traffic flow, incident reports, and weather conditions. This data is ingested into Azure Event Hubs for further stream processing and visualization.

APIs Used

API	Provider	Purpose
TomTom Geocoding API	TomTom	Converts city names into latitude and longitude coordinates.
TomTom Traffic Flow API	TomTom	Retrieves current traffic conditions, including speed, travel time, and congestion level.
TomTom Traffic Incident API	TomTom	Fetches real-time traffic incidents such as accidents or road closures.
OpenWeather API	OpenWeatherMap	Provides live weather details including temperature, humidity, and wind speed.

---

1. Geocoding API Endpoint:

https://api.tomtom.com/search/2/geocode/{city}.json

Method: GET **Parameters:**

Name Type Required Description city string
Yes Name of the city to geocode key string
Yes TomTom API key **Response Example:**

```
{
 "results": [
   {
    "position": {"lat": 30.0444, "lon": 31.2357}
   }
 ]
}
```

Usage in Code:
Retrieves latitude and longitude for the selected city.

---

2. Traffic Flow API Endpoint:
https://api.tomtom.com/traffic/services/4/flowSegmentData/relative0/10/json
Method: GET **Parameters:**
Name Type Required Description
point string Yes Comma-separated latitude and longitude
unit string Optional Measurement unit (e.g., KMPH) key
string Yes TomTom API key **Response Example:**

```
{
 "flowSegmentData": {
   "currentSpeed": 42,
   "freeFlowSpeed": 60,
   "currentTravelTime": 220,
   "freeFlowTravelTime": 180,
   "confidence": 0.98,
   "roadClosure": false
 }
}
```

---

3. Traffic Incident API Endpoint:
https://api.tomtom.com/traffic/services/5/incidentDetails
Method: GET **Parameters:**
Name Type Required Description
bbox string Yes Bounding box coordinates around the city key string
Yes TomTom API key
language string Optional Response language
timeValidityFilter string Optional Filters active incidents only (e.g., "present") **Response Example:**

```
{
```

```
 "incidents": [
  {
   "id": "abc123",
   "severity": "major",
   "description": "Accident on main road"
  }
 ]
}
```


---

4. Weather API Endpoint:
https://api.openweathermap.org/data/2.5/weather
Method: GET **Parameters:**
Name Type Required Description
lat float Yes Latitude lon
float Yes Longitude
units string Optional Unit of measurement (metric/imperial)
appid string Yes OpenWeather API key **Response**
Example:


```
{
 "weather": [{"description": "clear sky"}],
 "main": {"temp": 28, "feels_like": 30, "humidity": 40},
 "wind": {"speed": 2.5}
}
```


---

5. Combined Functionality
The main function fetch_weather_traffic(city) performs the following:

1. Uses **TomTom Geocoding API** to get city coordinates.
2. Calls **Traffic Flow** and **Incident APIs** to retrieve live traffic data.
3. Fetches **weather details** via OpenWeather API.
4. Merges all information into a unified **Pandas DataFrame** for further processing and streaming.

Example Output:
location temperature_c current_speed_kmph humidity_pct num_traffic_incidents
Cairo 28 42 40 3

---

6. Error Handling

- Each API call uses requests.get() with r.raise_for_status() to handle HTTP errors.
- In case of rate limit or timeout, the system logs the failure and retries the request.

- Data validation ensures that incomplete or null responses are handled gracefully before ingestion.

---

7. Integration Notes

- The API data is ingested into **Azure Event Hubs** for real-time stream processing.
- The system runs on a scheduled basis (e.g., every 5 minutes).
- Processed insights are visualized in **Power BI** and used for alert generation in **Azure Stream Analytics**.

Summary:
The API integration module forms the foundation of the real-time traffic analytics system by fusing live traffic and weather data into a unified, structured dataset ready for streaming, analysis, and visualization.
The system utilizes the **HTTP GET** method to retrieve live traffic data from a third-party API endpoint. Each incoming record includes key parameters such as **vehicle speed**, **congestion level**, and **geographical coordinates**, which are then transmitted to the ingestion layer for processing.

Testing & Validation

Testing involves multiple stages to ensure system reliability and performance:

- **Data Accuracy Validation:** Verifies the correctness and consistency of traffic data retrieved from the API.
- **Stream Analytics Performance Monitoring:** Evaluates query efficiency and throughput during high data volumes.
- **End-to-End Dashboard Verification:** Confirms that processed data is accurately reflected in the live Power BI dashboard.

Deployment Strategy

All Azure components are deployed within a **single resource group** for simplified management and monitoring. Both **Azure Stream Analytics** and **Azure SQL Database** are configured with **autoscaling capabilities** to efficiently handle sudden data surges and maintain optimal performance.