COLLEGE CODE : 5134

COLLEGE NAME : University College of Engineering Kanchipuram

DEPARTMENT : ECE

STUDENTNM-ID : 79d2a9bc3935d5ca1ff45e115faca622

ROLL NO : 513423106013

DATE : 15-5-2025

Completed the project named as

TRAFFIC FLOW OPTIMIZATION

SUBMITTED BY,

R.SHARMITHA

C.MAYAVATHI

R.DEVAKI

E.SUDDHALAKSHMI

R.DEVAKI

Phase 5: Project Demonstration & Documentation

Title: AI-Based Traffic Flow Optimization System

Abstract: The **AI-Based Traffic Flow Optimization System** is designed to improve urban traffic management using artificial intelligence, real-time data analytics, and intelligent control strategies. The system full fills four primary objectives: **AI-powered traffic prediction and optimization**, **adaptive traffic signal control**, **real-time incident detection and management**, and **emergency vehicle prioritization**. It integrates machine learning models, live traffic feeds, and sensor data to optimize vehicle movement and reduce congestion. The platform dynamically adjusts signal timing based on predicted traffic flow, detects incidents such as accidents or blockages in real-time, and ensures priority clearance for emergency vehicles. This document includes a complete overview of the project development, live demonstration, performance metrics, code structure, and testing outcomes, along with visual proofs like architecture diagrams and working screen short.

1. Project Demonstration

Overview:

The demonstration highlights the live functioning of the traffic flow optimization system using simulated or real-time data streams. It covers intelligent traffic prediction, signal adaptation, emergency prioritization, and incident alerts.

Demonstration Details:

- **System Walkthrough:** Interactive tour from data input (sensor/video feed) to real-time output (optimized signals and alerts).
- **AI Traffic Prediction:** Use of trained models (e.g., LSTM or regression) to forecast traffic density and patterns.
- **Adaptive Signal Control:** Signal lights change dynamically based on congestion levels and predictive models.
- **Incident Detection:** AI identifies anomalies such as sudden slowdowns, crashes, or roadblocks via sensors/cameras.
- **Emergency Prioritization:** The system detects approaching emergency vehicles and modifies signal paths to clear the way.
- **Performance Metrics:** Showcases reductions in average wait times, congestion levels, and improvements in emergency response time.

Outcome:

A clear demonstration of the system's real-time adaptability and effectiveness in smart traffic management and emergency handling.

2. Project Documentation

Overview:

In-depth documentation of the full system architecture, development process, and technical implementation of all four objectives.

Documentation Sections:

- **System Architecture:** Diagrams showing AI model integration, traffic light logic, emergency detection flow, and camera/sensor inputs.
- **AI Model Design:** Explanation of training data, model type (e.g., LSTM, Random Forest), features used, and accuracy metrics.

- **Code Modules:** Documentation of source code for prediction engine, signal controller, and incident detection.
- **User Guide:** Interface usage for traffic authorities, including dashboard overview and manual overrides.
- **Admin Guide:** Instructions on retraining models, updating signal logic, and integrating new sensors or intersections.
- **Testing Reports:** Simulated vs. real-time test results for accuracy, responsiveness, and error handling.

Outcome:

Complete technical and functional documentation enabling future scalability, replication, or enhancement.

3. Feedback and Final Adjustments

Overview:

Constructive feedback will be gathered and used to improve prediction reliability, user experience, and edge-case handling.

Steps:

- **Feedback Collection:** Input from faculty, domain experts, and test users during the demo phase.
- **System Refinement:** Tuning AI model parameters, improving signal response logic, and enhancing emergency detection.
- **Final Testing:** Post-refinement validation under high traffic simulations to ensure scalability and robustness.

Outcome:

A more accurate, stable, and user-friendly system ready for integration into smart city frameworks.

4. Final Project Report Submission

Overview:

Summarizes the entire development process, innovations, issues faced, and final outcomes.

Report Sections:

- **Executive Summary:** Goals, innovations, and performance highlights of the project.
- **Objective-wise Breakdown:** Detailed explanation of how each of the four key objectives was achieved.
- **Challenges & Solutions:** Roadblocks such as false detections or signal timing conflicts, and their resolutions.
- **Final Outcomes:** Tangible metrics like wait time reduction %, emergency clearance time, and prediction accuracy.

Outcome:

A comprehensive and well-structured final report showing real-world applicability of the system.

5. Project Handover and Future Works

Overview:

Provides recommendations and plans for further development, scaling, and field deployment.

Handover Details:

- **Next Steps:** Suggestions include integrating with city traffic APIs, adding mobile control for traffic police, and multi-intersection coordination.
- **Future Enhancements:** Include cloud-based monitoring, advanced video analytics, multilingual dashboards, and support for pedestrian and cyclist prioritization.

Outcome:

Project handover with clear guidelines for maintenance, future work, and potential smart city deployment.

CODE FOR TRAFFIC FLOW OPTIMIZATION:

PICTURE 1:

data_loader.py

C: > Users > pvais > Downloads > vaish > NM > data_loader.py

```python
import pandas as pd
import streamlit as st
import os
import datetime
import numpy as np

def load_traffic_data():
    """
    Load traffic data from CSV file or create sample data if file doesn't exist
    """
    try:
        # Try to load from file
        if os.path.exists("sample_data/traffic_data.csv"):
            data = pd.read_csv("sample_data/traffic_data.csv")
            return data
        else:
            # Create sample data for demonstration
            return create_sample_traffic_data()
    except Exception as e:
        st.error(f"Error loading traffic data: {str(e)}")
        return pd.DataFrame()

def load_incidents_data():
    """
    Load incidents data from CSV file or create sample data if file doesn't exist
    """
    try:
        # Try to load from file
        if os.path.exists("sample_data/incidents.csv"):
            data = pd.read_csv("sample_data/incidents.csv")
            return data
        else:
            # Create sample data for demonstration
            return create_sample_incidents_data()
    except Exception as e:
        st.error(f"Error loading incidents data: {str(e)}")
        return pd.DataFrame()
```

PICTURE 2:

traffic-optimizer.py

C:\Users\pvais\Downloads\vaish\NM\traffic-optimizer.py

```python
import pandas as pd
import numpy as np
import datetime

def optimize_signal_timing(traffic_df, current_cycle_length):
    """
    Calculate optimized signal timing based on traffic patterns

    Args:
        traffic_df: Traffic data dataframe for a specific location
        current_cycle_length: Current signal cycle length in seconds

    Returns:
        Optimized cycle length in seconds
    """
    if traffic_df.empty:
        return current_cycle_length

    # Calculate average volume and congestion
    avg_volume = traffic_df['volume'].mean()
    avg_congestion = traffic_df['congestion'].mean()

    # Simple optimization logic based on volume and congestion
    # In a real system, this would use more sophisticated algorithms

    # Base cycle length adjustment
    if avg_volume < 200:
        # Low volume, shorter cycles
        base_cycle = 60
    elif avg_volume < 400:
        # Medium volume
        base_cycle = 90
    else:
        # High volume, longer cycles
        base_cycle = 120

```

PICTURE 3:

replit.py

C: > Users > pvais > Downloads > vaish > NM > replit.py

```python
import streamlit as st
import pandas as pd
import numpy as np
import datetime

# Import custom modules
import data_loader as dl
import data_processor as dp
import map_visualizer as mv
import ml_models as ml
import time_series as ts
import traffic_optimizer as to

# Set page configuration
st.set_page_config(
    page_title="Traffic Analysis Platform",
    page_icon="🚦",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Application title and description
st.title("Traffic Analysis & Optimization Platform")
st.markdown("""
This platform provides tools for analyzing traffic patterns,
predicting future traffic conditions, monitoring traffic signals,
and managing traffic incidents.
""")

# Sidebar for navigation
st.sidebar.title("Navigation")
page = st.sidebar.radio(
    "Select a Page",
    ["Dashboard", "Traffic Prediction", "Signal Monitoring",
     "Incident Management", "Historical Analysis", "Optimization"]
)
```

PICTURE 4:

replit.py

C: > Users > pvais > Downloads > vaish > NM > replit.py

```python
else:
    elif page == "Traffic Prediction":
        st.header("Traffic Prediction")

        # Location selection for prediction
        locations = traffic_data['location'].unique()
        selected_location = st.selectbox("Select Location for Prediction", locations)

        # Time horizon for prediction
        prediction_days = st.slider("Prediction Horizon (days)", 1, 30, 7)

        # Run prediction when button is clicked
        if st.button("Run Prediction"):
            with st.spinner("Running prediction model..."):
                # Filter data for selected location
                location_data = traffic_data[traffic_data['location'] == selected_location]

                if location_data.empty:
                    st.warning(f"No data available for the selected location: {selected_location}")
                else:
                    # Run prediction using Prophet
                    forecast_df, forecast_plot = ts.predict_traffic(location_data, prediction_days)

                    # Display prediction results
                    st.subheader(f"Traffic Prediction for {selected_location}")
                    st.plotly_chart(forecast_plot, use_container_width=True)

                    # Show forecast data table
                    st.subheader("Forecast Data")
                    st.dataframe(forecast_df[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].rename(
                        columns={'ds': 'Date', 'yhat': 'Prediction', 'yhat_lower': 'Lower Bound', 'yhat_upper': 'Upper Bound'}
                    ))

                    # Peak traffic times
                    peak_times = ts.identify_peak_times(forecast_df)
                    st.subheader("Predicted Peak Traffic Times")
                    st.write(peak_times)
```

PICTURE 5:

map_visualizer.py

C: > Users > pvais > Downloads > vaish > NM > map_visualizer.py

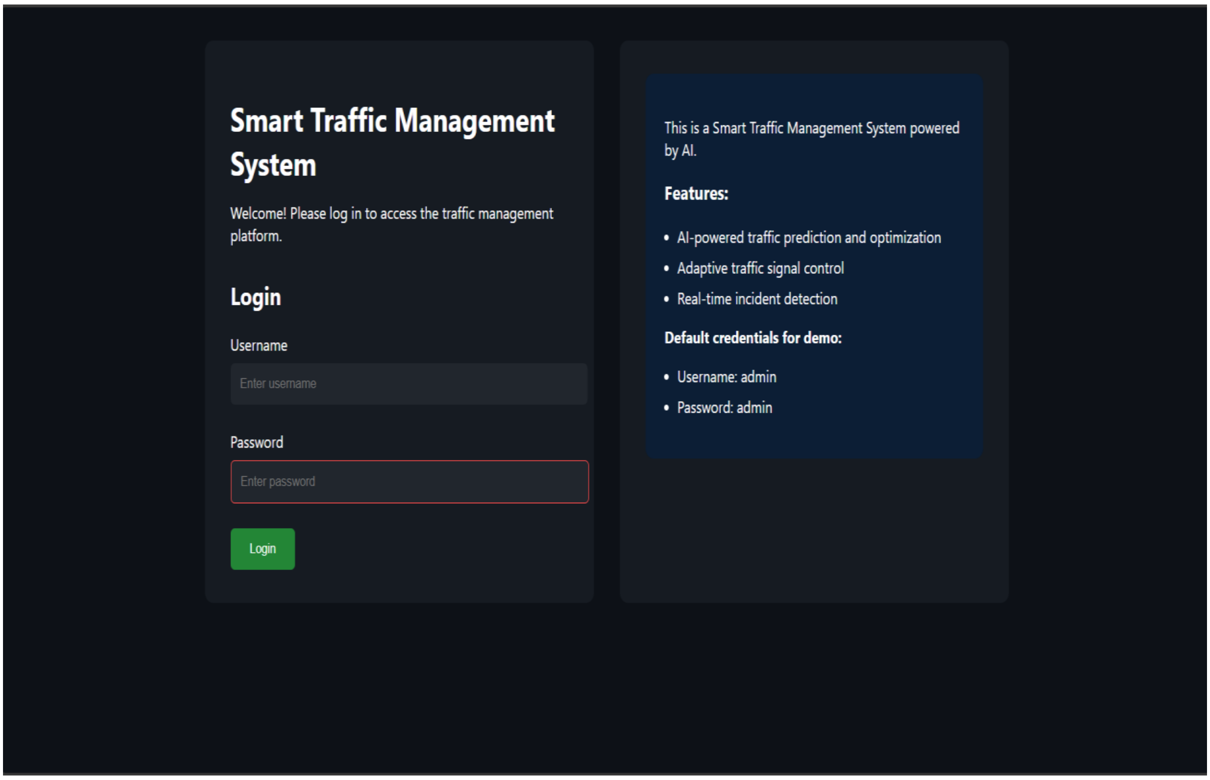
```python
import folium
from folium.plugins import HeatMap, MarkerCluster
import pandas as pd
import numpy as np

def create_traffic_map(traffic_df, incidents_df=None):
    """
    Create an interactive map visualizing traffic volume and incidents

    Args:
        traffic_df: Traffic data dataframe
        incidents_df: Incidents data dataframe (optional)

    Returns:
        Folium map object
    """
    # Check if dataframe is empty
    if traffic_df.empty:
        # Create a default map
        default_center = [40.7128, -74.0060]  # NYC coordinates as default
        traffic_map = folium.Map(location=default_center, zoom_start=12)
        folium.Marker(
            location=default_center,
            popup="No traffic data available",
            icon=folium.Icon(icon="info-sign", color="red")
        ).add_to(traffic_map)
        return traffic_map

    # Calculate map center based on data
    center_lat = traffic_df['latitude'].mean()
    center_lon = traffic_df['longitude'].mean()

    # Create base map
    traffic_map = folium.Map(location=[center_lat, center_lon], zoom_start=12)
```

PICTURE 6:

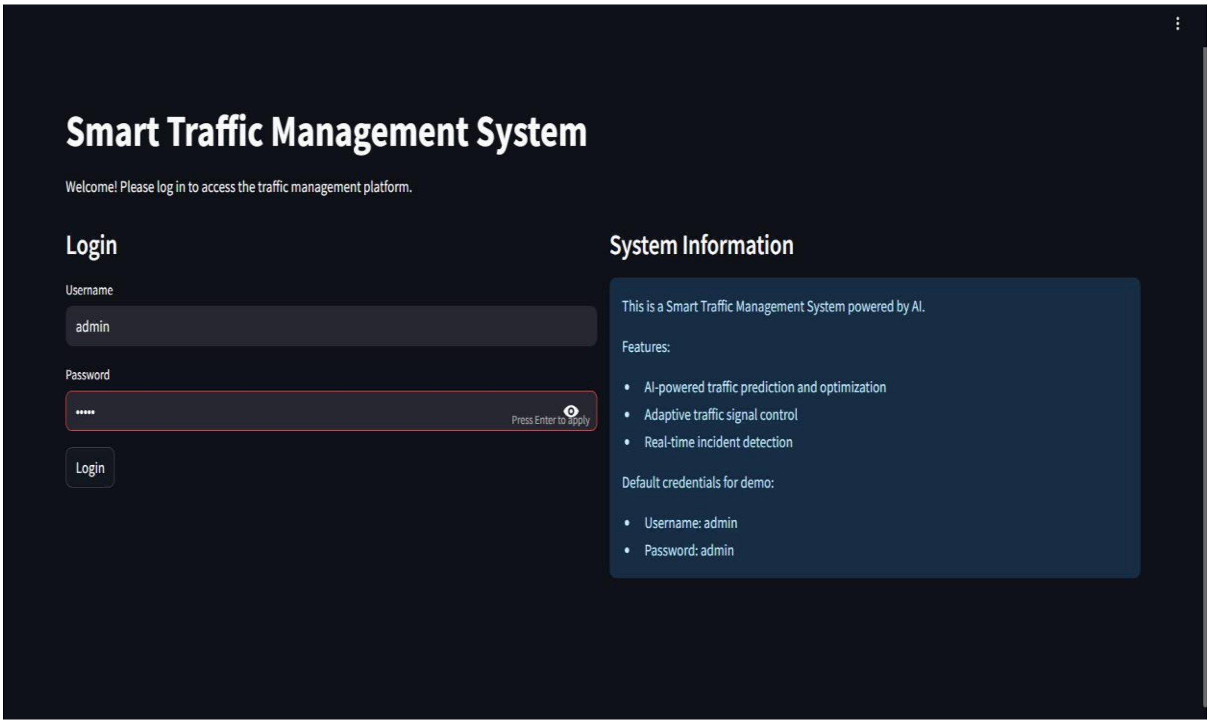
```python
def create_incident_map(incidents_df):
    for _, incident in incidents_df.iterrows():
        icon_map = {
            "Event": "calendar",
            "Hazard": "exclamation-triangle",
            "Other": "info-circle"
        }

        color_map = {
            "Low": "green",
            "Medium": "orange",
            "High": "red",
            "Critical": "darkred"
        }

        icon_type = icon_map.get(incident['type'], "info-circle")
        color = color_map.get(incident['severity'], "blue")

        # Create popup content
        popup_content = f"""
        <b>{incident['type']} ({incident['severity']})</b><br>
        Location: {incident['location']}<br>
        Description: {incident['description']}<br>
        Time: {incident['timestamp']}<br>
        Duration: {incident['duration']} hours<br>
        Impact: {incident['impact']:.2f}<br>
        Status: {incident['status']}
        """

        # Add marker
        folium.Marker(
            location=[incident['latitude'], incident['longitude']],
            popup=folium.Popup(popup_content, max_width=200),
            icon=folium.Icon(icon=icon_type, prefix="fa", color=color)
        ).add_to(marker_cluster)

    return incident_map
```

OUTPUT:

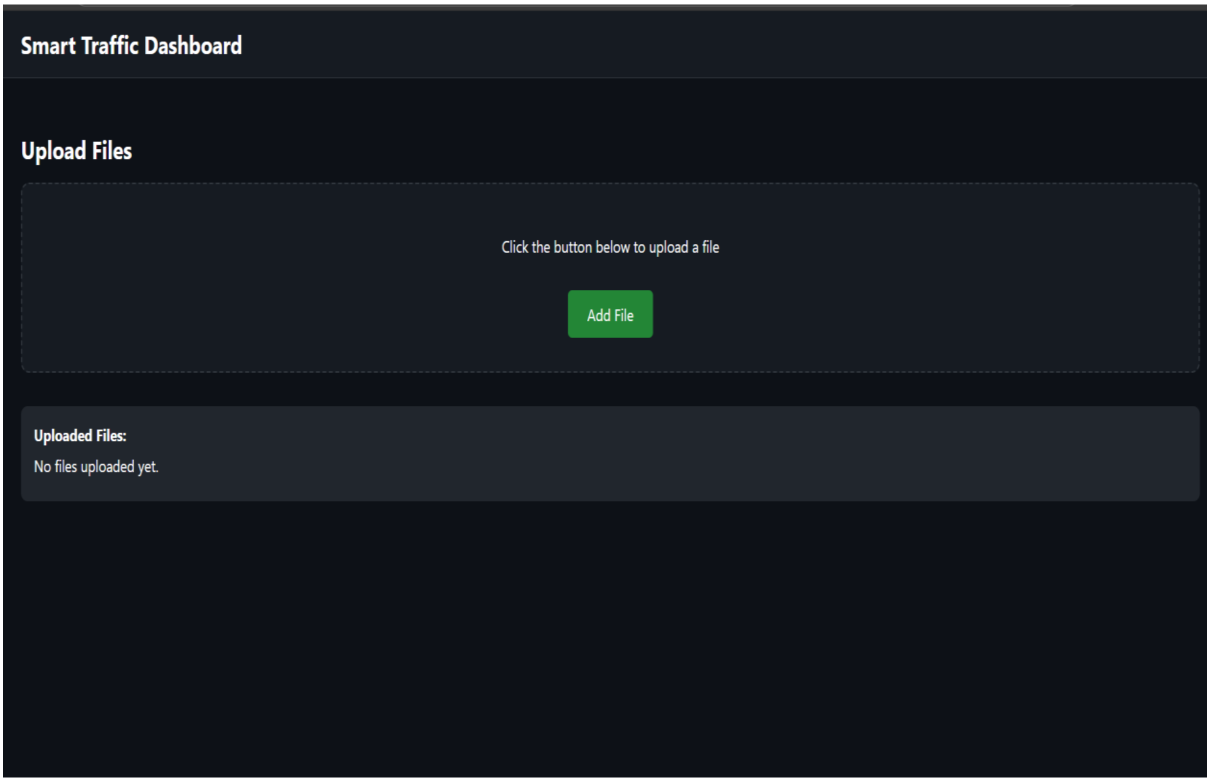
PICTURE 1:



PICTURE 2:



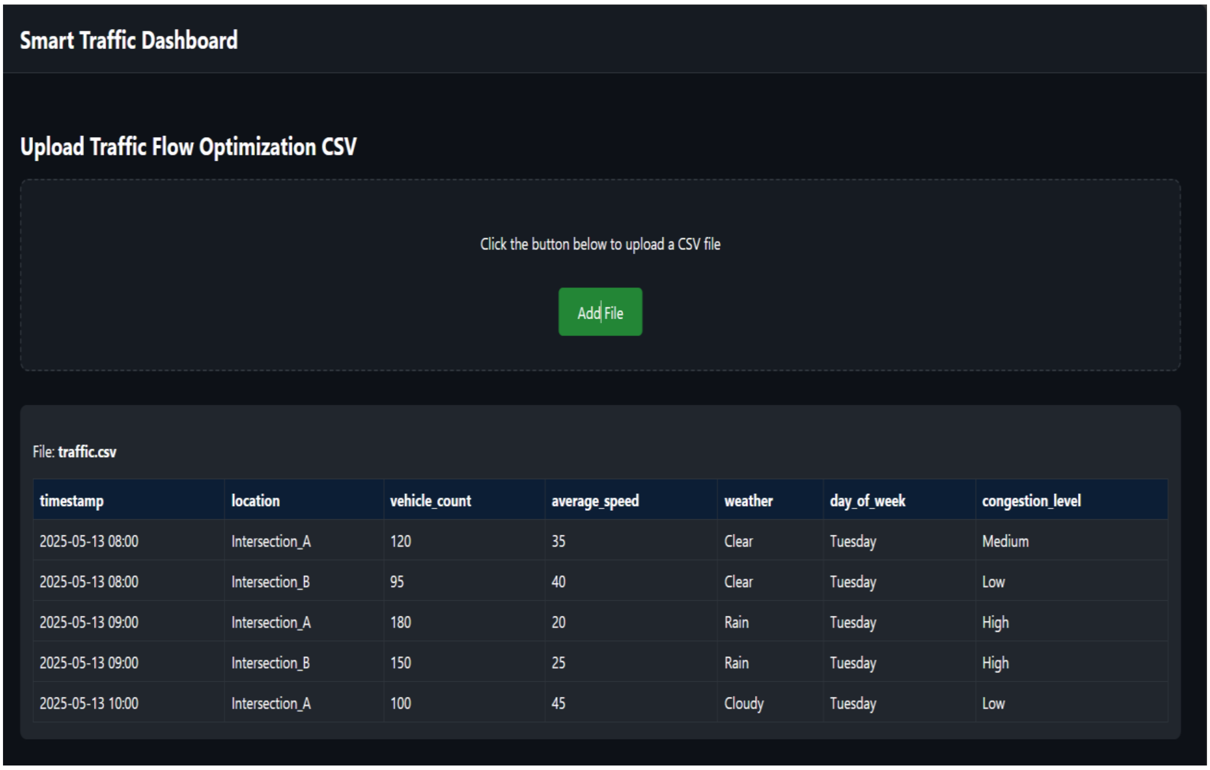
PICTURE 3:



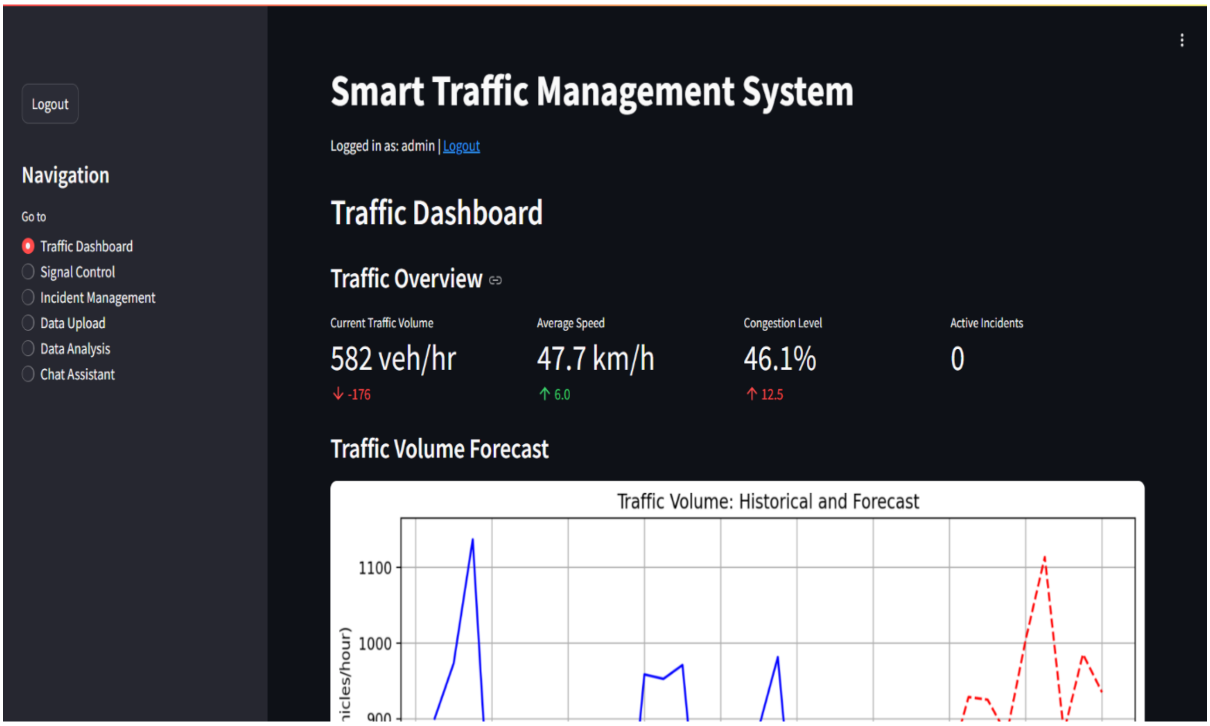
PICTURE 4:

Schema	Name	Type	Owner	Size	Description
public	incidents	table	neondb_owner	16 kB	
public	incidents_id_seq	sequence	neondb_owner	8192 bytes	
public	locations	table	neondb_owner	32 kB	
public	locations_id_seq	sequence	neondb_owner	8192 bytes	
public	traffic_data	table	neondb_owner	16 kB	
public	traffic_data_id_seq	sequence	neondb_owner	8192 bytes	
public	traffic_signals	table	neondb_owner	16 kB	
public	traffic_signals_id_seq	sequence	neondb_owner	8192 bytes	
public	users	table	neondb_owner	56 kB	
public	users_id_seq	sequence	neondb_owner	8192 bytes	

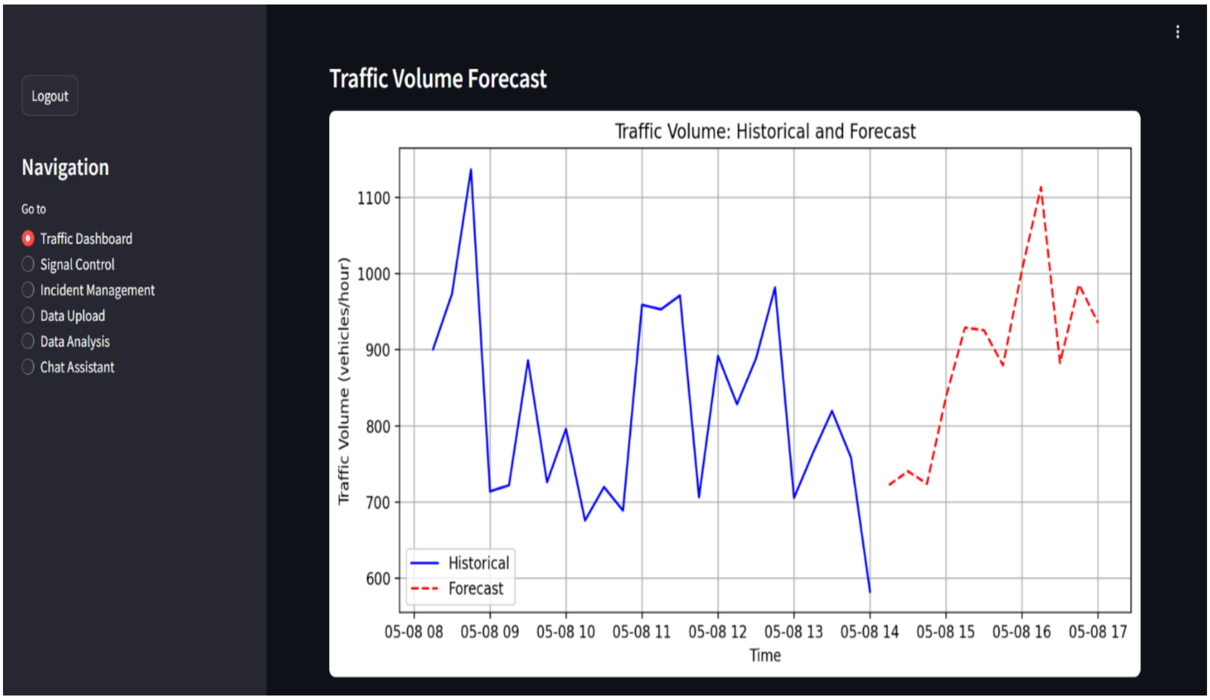
PICTURE 5:



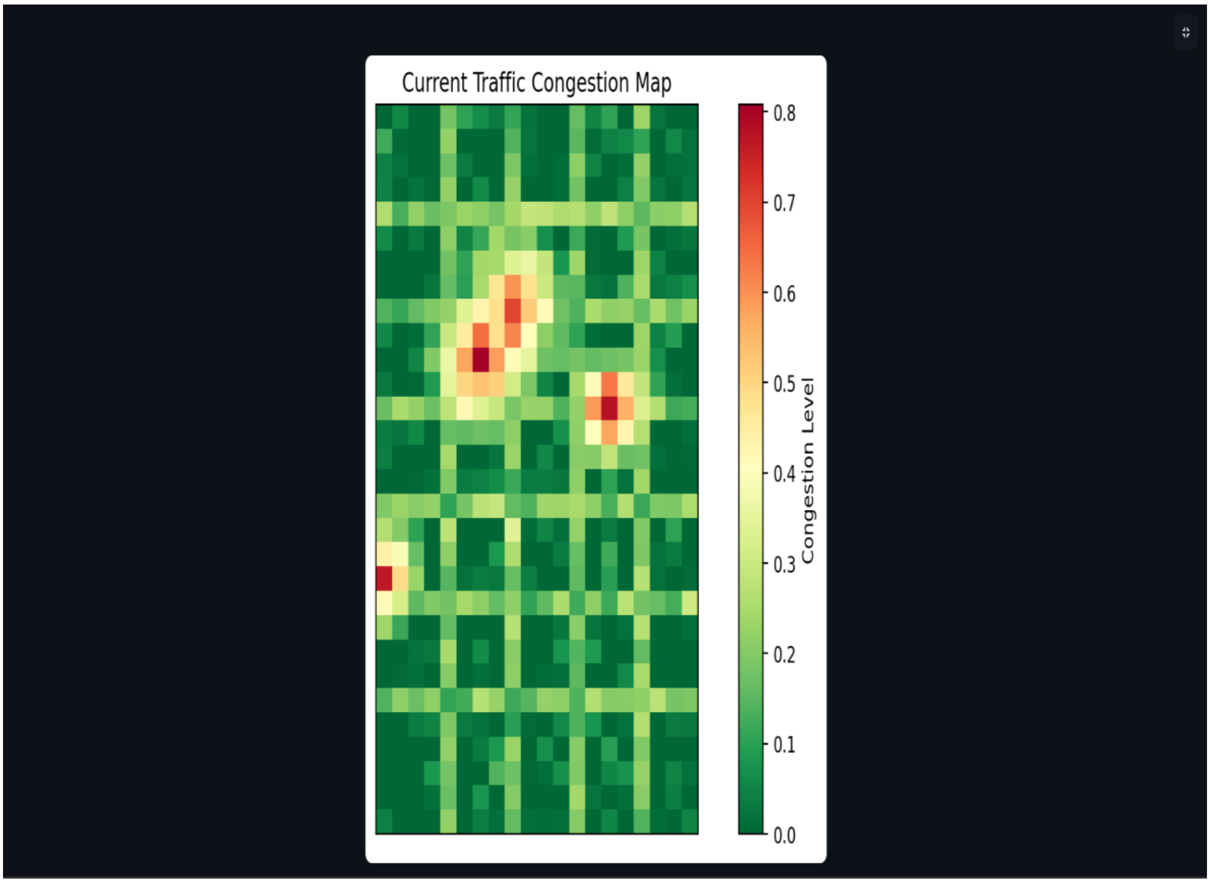
PICTURE 6:



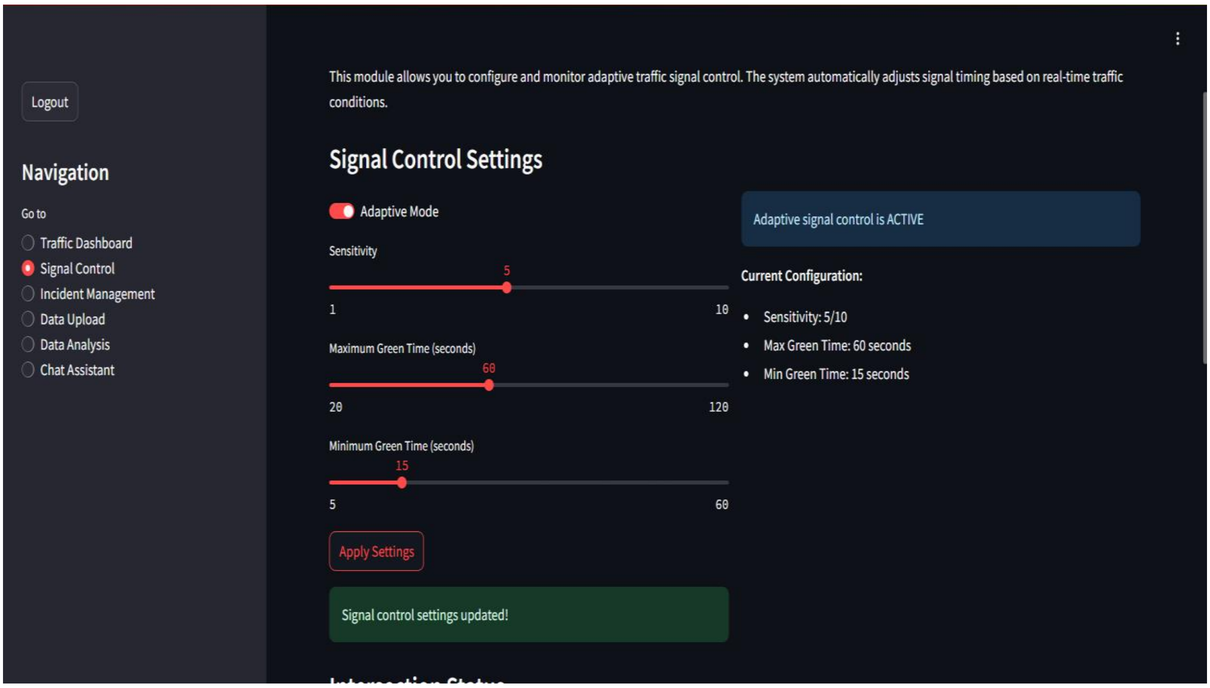
PICTURE 7:



PICTURE 8:

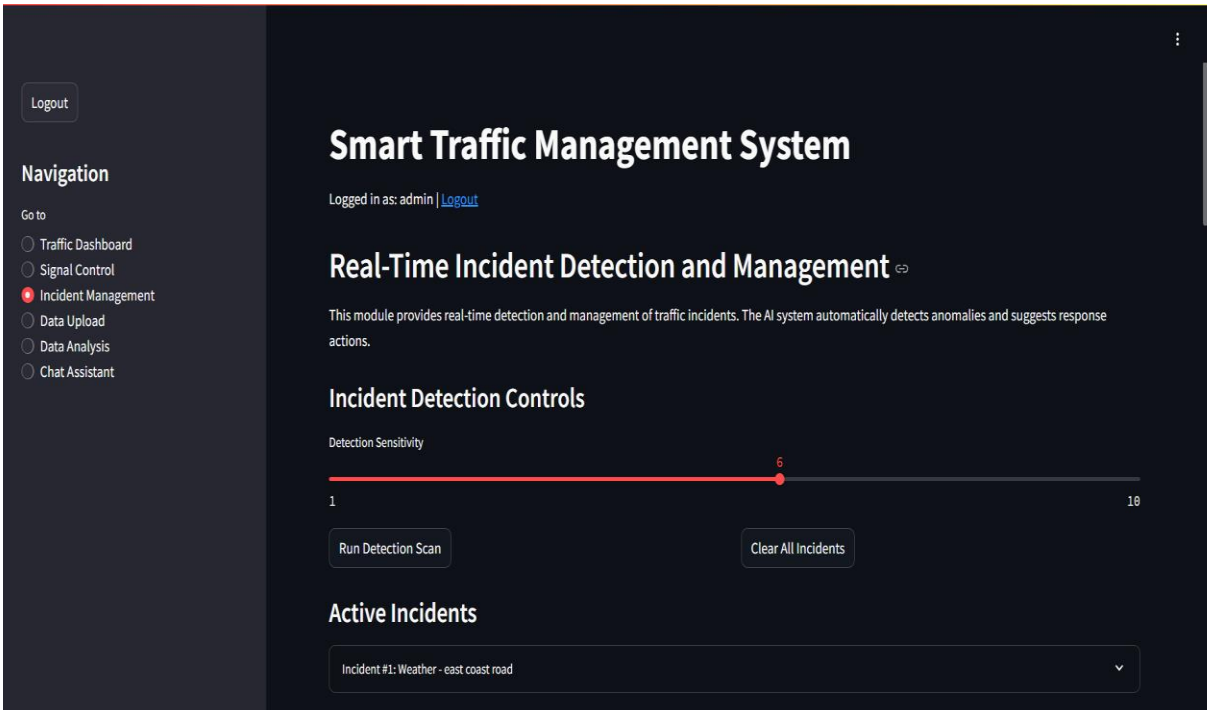


PICTURE 9:

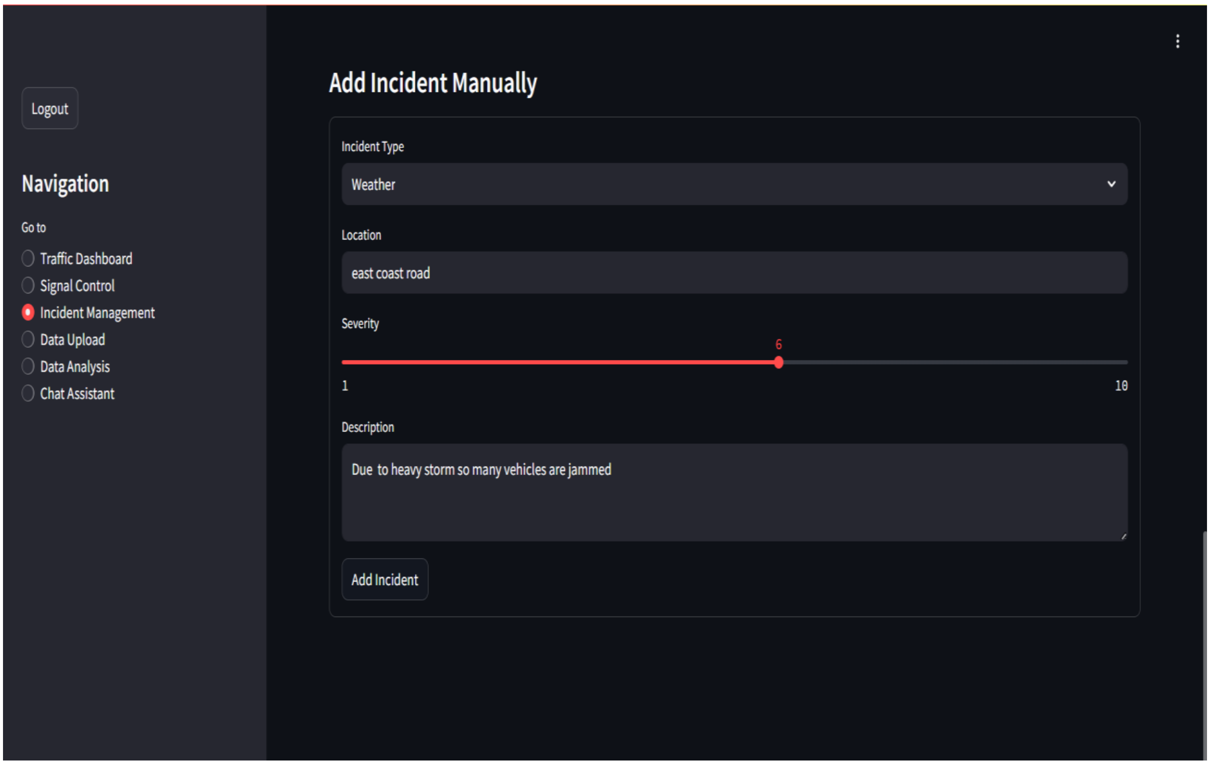


PICTURE 10:

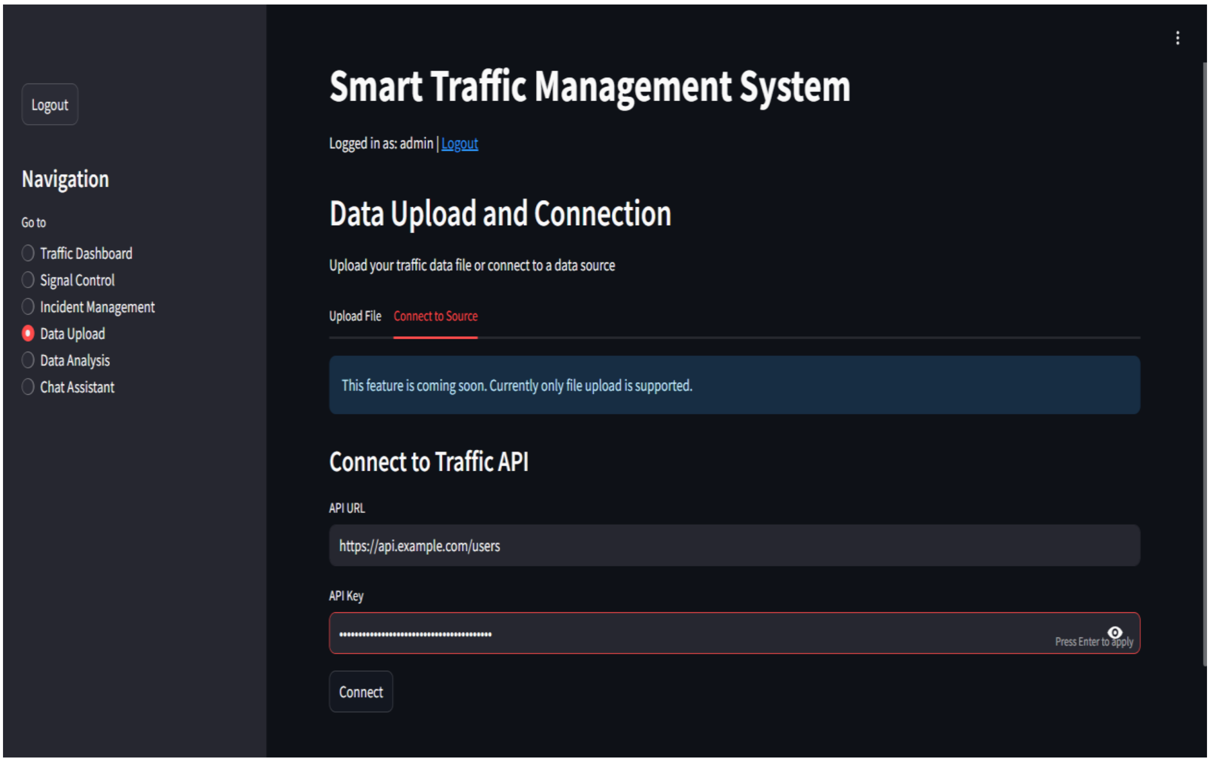
PICTURE 11:



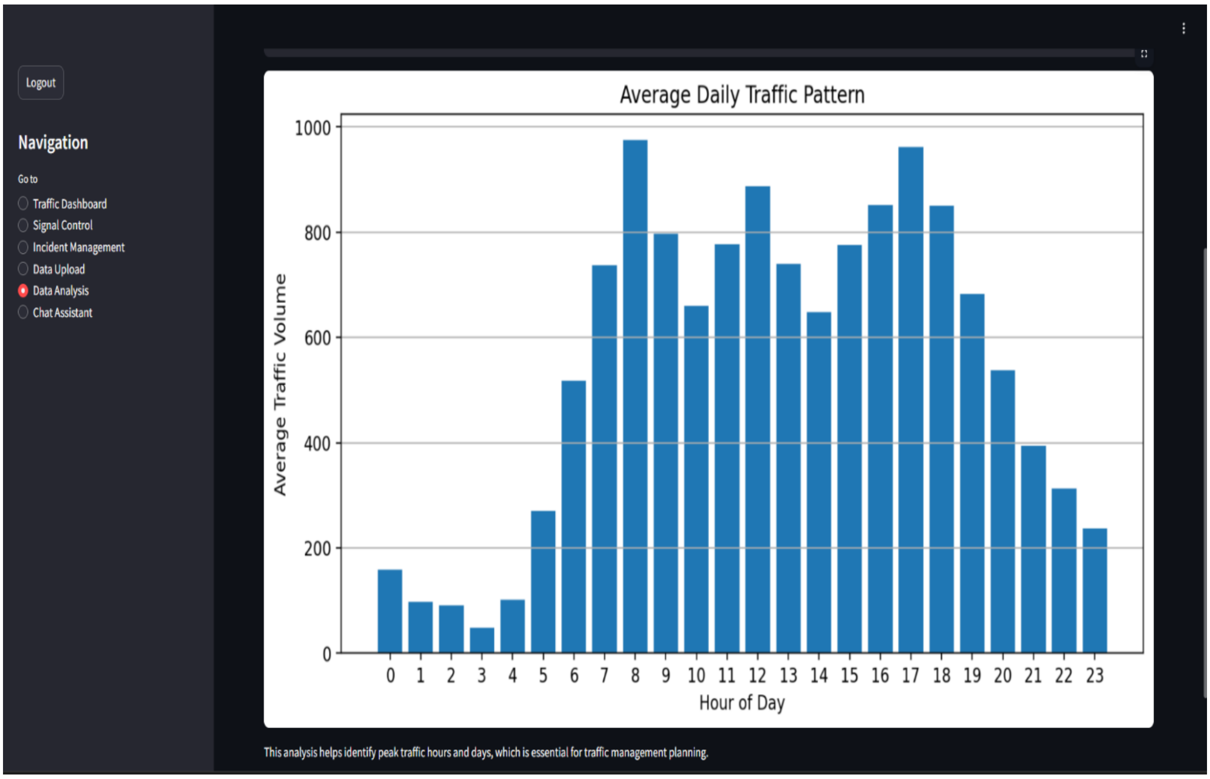
PICTURE 12:



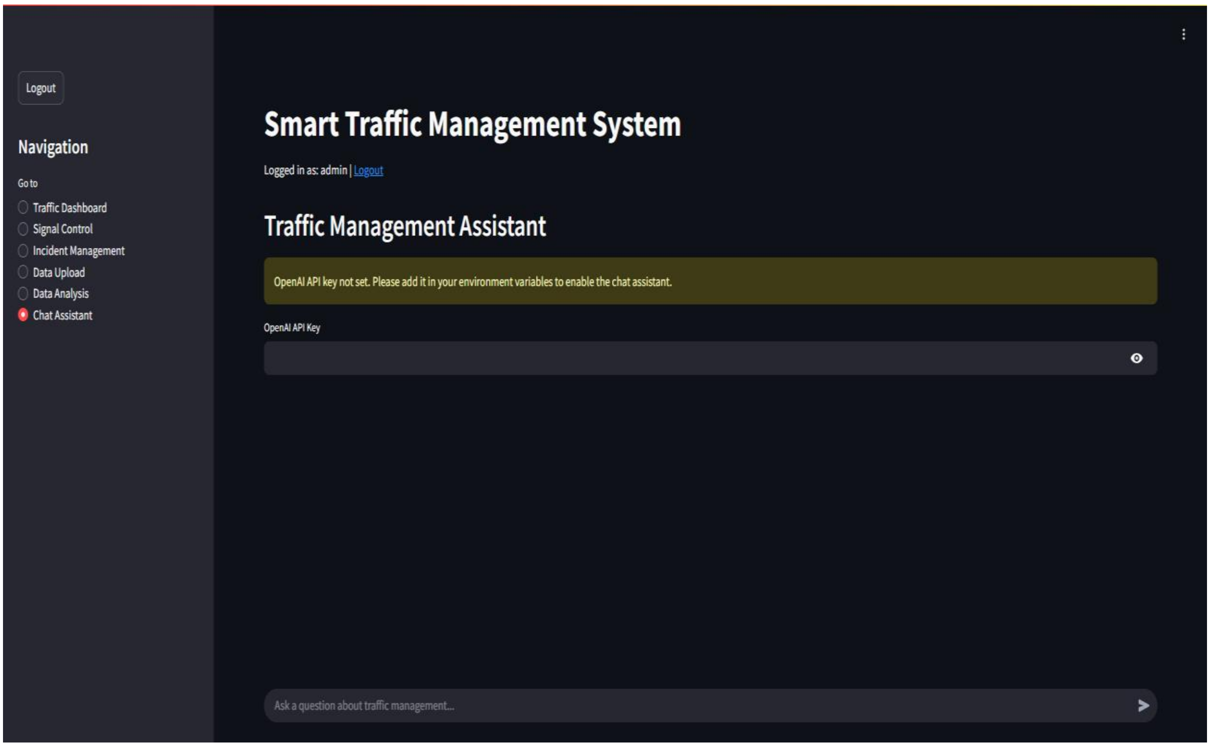
PICTURE 13:



PICTURE 14:



PICTURE 15:


Logout
Navigation
Go to
Traffic Dashboard
Signal Control
Incident Management
Data Upload
Data Analysis
Chat Assistant
Smart Traffic Management System
Logged in as: admin | Logout
Traffic Management Assistant
OpenAI API key not set. Please add it in your environment variables to enable the chat assistant.
OpenAI API Key
Ask a question about traffic management...