

# **KINGSTON ENGINEERING COLLEGE**

**DOMAIN: INTERNET OF THINGS**

**PROJECT TITLE: Traffic Management system.**

**COLLEGE CODE: 5113**

**Phase 5**

**PROJECT MEMBERS:**

**D Thukkaram-511321104104**

**R Sriram-511321104098**

**S Prasanth-511321104069**

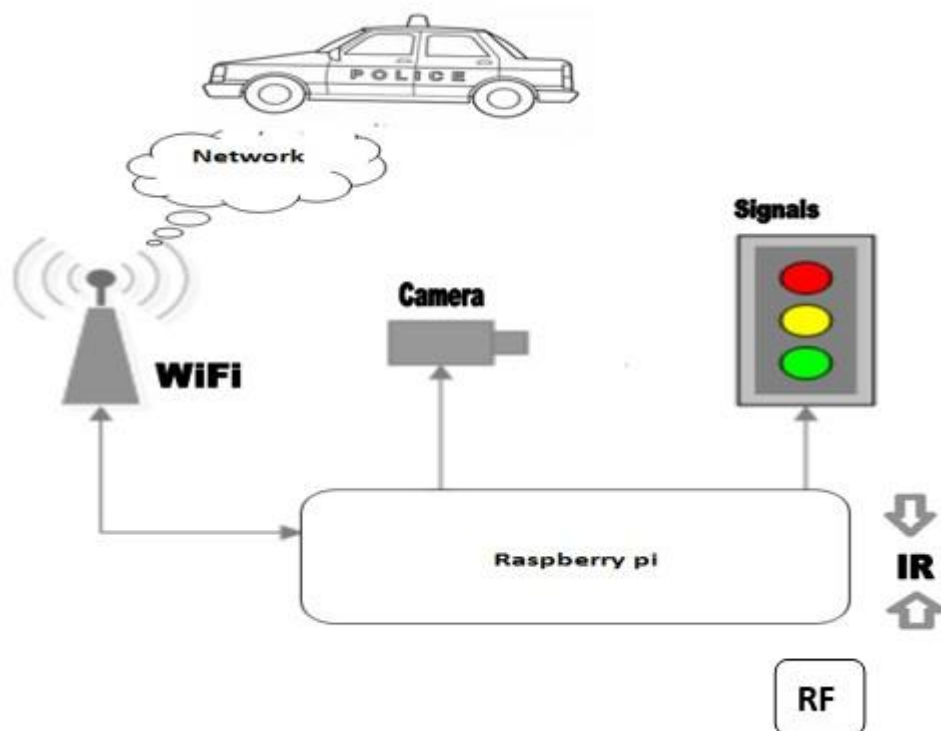
**R Pravin Kumar-511321104073**

**S Sujith-511321104097**

### Project Objectives:

The objective of the traffic management system project is to develop a real-time traffic monitoring system that can assist commuters in making optimal route decisions and improving traffic flow. By leveraging IoT sensors, a mobile app, and Raspberry Pi integration, the project aims to provide accurate and up-to-date traffic information to users, enabling them to choose the most efficient routes and avoid congested areas. The key goals of the project include:

1. **Real-time Traffic Monitoring:** Develop a system that can collect real-time traffic data from IoT sensors deployed in various locations.
2. **Data Analysis:** Analyse the collected traffic data to identify traffic patterns, congestion areas, and traffic flow.
3. **Mobile App Development:** Create a user-friendly mobile application that can display real-time traffic information, suggest optimal routes, and provide navigation guidance.
4. **Raspberry Pi Integration:** Integrate Raspberry Pi devices with the IoT sensors to facilitate data transmission and processing.
5. **Code Implementation:** Develop the necessary code to enable data transmission, analysis, and visualization.



### **IoT Sensor Setup:**

To replicate the project, you will need to set up IoT sensors in strategic locations to collect traffic data. The IoT sensors should be capable of detecting and measuring traffic parameters such as vehicle count, speed, and congestion levels. The sensors can be deployed at intersections, highways, or other relevant locations. The data collected by the sensors will be transmitted to a central server for further analysis and processing.

### **Mobile App Development:**

The project involves developing a mobile application that will serve as the interface for users to access real-time traffic information. The mobile app should have the following features:

1. Real-time Traffic Updates: Display real-time traffic conditions, including congestion levels, accidents, and road closures.
2. Optimal Route Suggestions: Provide users with suggested routes based on current traffic conditions, helping them make informed decisions.
3. Navigation Guidance: Offer turn-by-turn navigation instructions to guide users along their chosen routes.
4. User Feedback: Allow users to provide feedback on traffic conditions, report incidents, and contribute to the overall accuracy of the system.

The mobile app can be developed using platforms such as Android Studio or iOS development frameworks, depending on the target platform.

### **Raspberry Pi Integration:**

Raspberry Pi devices will be integrated into the project to facilitate data transmission and processing. The Raspberry Pi devices will act as gateways for the IoT sensors, collecting data from the sensors and transmitting it to the central server. The integration process involves connecting the IoT sensors to the Raspberry Pi devices and developing the necessary code to enable data transmission.

### **Code Implementation:**

The code implementation for the project will involve several components:

1. IoT Sensor Code: Develop code to interface with the IoT sensors, collect data, and transmit it to the Raspberry Pi devices.
2. Raspberry Pi Code: Create code to receive data from the IoT sensors, process it, and transmit it to the central server.

3. **Server-side Code:** Develop code to receive and store the traffic data from the Raspberry Pi devices, perform data analysis, and generate real-time traffic updates.
4. **Mobile App Code:** Implement the code for the mobile app, including data retrieval from the server, route suggestion algorithms, and navigation guidance.

The code can be implemented using programming languages such as Python, which offers libraries and frameworks for IoT development, data analysis, and mobile app development.

### **Example Outputs:**

#### **Example outputs of the project can include:**

1. **Raspberry Pi Data Transmission:** The Raspberry Pi devices successfully collect data from the IoT sensors and transmit it to the central server. This can be demonstrated by displaying the transmitted data on the server console or logging the data in a file.
2. **Mobile App UI:** The mobile app displays real-time traffic information, suggests optimal routes, and provides navigation guidance. Example outputs can include screenshots or mockups of the mobile app UI, showcasing the different features and functionalities.

### **The real-time traffic monitoring system:**

A real-time traffic monitoring system can greatly assist commuters in making optimal route decisions and improving traffic flow. Here's how:

**1. Real-time Traffic Data Collection:** The system utilizes IoT sensors placed strategically across roadways to collect real-time traffic data. These sensors can include cameras, radar, and other devices that capture information such as vehicle speed, volume, and congestion levels. The collected data is then transmitted to a central server or cloud platform for further processing and analysis.

**2. Data Analysis and Processing:** The collected traffic data is analyzed using advanced algorithms and machine learning techniques. This analysis helps in identifying traffic patterns, congestion hotspots, and predicting traffic flow. By

processing this data in real-time, the system can provide up-to-date information on traffic conditions.

**3. Commuter Information Platform:** The system integrates with a mobile app or a web-based platform that provides commuters with real-time traffic information. This platform can display live traffic updates, suggest alternative routes, and estimate travel times based on the analyzed data. Commuters can access this information on their smartphones or other devices, allowing them to make informed decisions about their routes.

**4. Optimal Route Recommendations:** The real-time traffic monitoring system can recommend the most optimal routes to commuters based on the current traffic conditions. By considering factors such as congestion levels, accidents, and road closures, the system can suggest alternative routes that help commuters avoid delays and congestion.

**5. Traffic Flow Management:** The system also assists in improving traffic flow by providing valuable insights to traffic management authorities. By analyzing the collected data, authorities can identify areas with recurring congestion and take appropriate measures to alleviate traffic issues. This can include adjusting traffic signal timings, implementing lane management strategies, or optimizing public transportation routes.

### **Example Outputs:**

#### **1. Raspberry Pi Data Transmission:**

- The Raspberry Pi, equipped with sensors, collects real-time traffic data.
- The data is processed and transmitted to a central server or cloud platform.
- The server receives the data and performs analysis to extract meaningful insights.

#### **2. Mobile App User Interface (UI):**

- The mobile app displays a map with real-time traffic information.
- Commuters can see color-coded routes indicating traffic conditions (e.g., green for smooth flow, yellow for moderate congestion, red for heavy congestion).

- The app provides alternative route suggestions and estimated travel times for each option.
- Users can customize their preferences and receive notifications about traffic incidents or road closures.

### **Code Implementation:**

To replicate this project, you would need to follow these steps:

#### **1. IoT Sensor Setup:**

- Choose appropriate IoT sensors such as cameras, radar, or other devices for traffic data collection.
- Install and configure the sensors at strategic locations on roadways.
- Connect the sensors to a Raspberry Pi or similar device for data processing and transmission.

#### **2. Mobile App Development:**

- Develop a mobile app using a programming language like Python or a cross-platform framework like React Native.
- Design the user interface to display real-time traffic information, alternative routes, and estimated travel times.
- Implement features such as route recommendations, user preferences, and notifications for traffic incidents.

#### **3. Raspberry Pi Integration:**

- Set up a Raspberry Pi with the necessary software and libraries for data collection and transmission.
- Connect the IoT sensors to the Raspberry Pi and configure them to capture traffic data.
- Develop code to process and transmit the collected data to a central server or cloud platform.

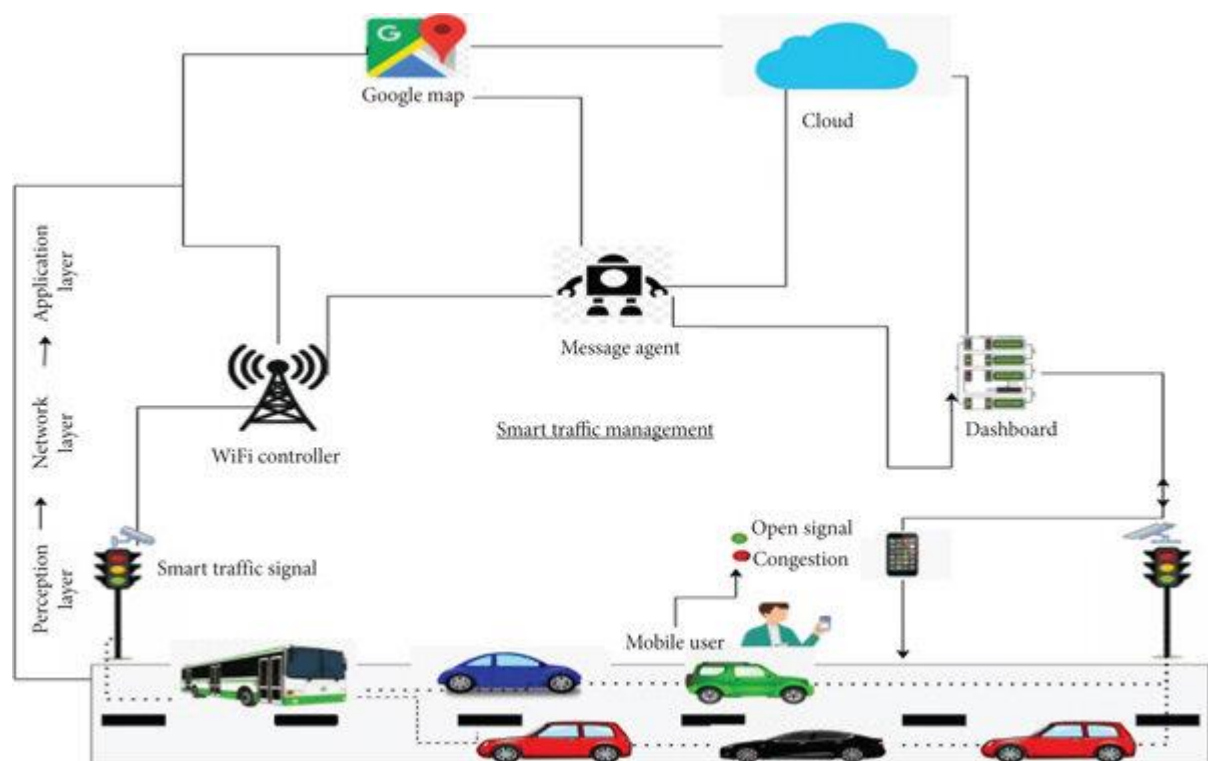
#### 4. Code Implementation:

- Write code to analyze the collected traffic data using algorithms and machine learning techniques.
- Implement functions to extract insights, identify traffic patterns, and predict traffic flow.
- Integrate the code with the mobile app to provide real-time traffic information and route recommendations.

Remember, this is a high-level overview, and the actual implementation may vary depending on specific requirements and technologies used.

#### Instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python:

To replicate the traffic management system project and integrate IoT sensors with a transit information platform using Python, follow these instructions:



### **1. Set up the IoT Sensors:**

- Choose IoT sensors capable of collecting traffic data such as vehicle count, speed, and congestion levels.
- Install the necessary hardware components, such as Raspberry Pi, cameras, and sensors, according to the manufacturer's instructions.
- Connect the sensors to the Raspberry Pi device.
- Install the required libraries and dependencies on the Raspberry Pi.

### **2. Develop the Transit Information Platform:**

- Create a Python project directory for the transit information platform.
- Set up a database to store and manage the collected traffic data.
- Design the database schema to store sensor data, location information, and historical traffic patterns.
- Implement data models and create database tables using a Python ORM (Object-Relational Mapping) library like SQL Alchemy.
- Develop a RESTful API using a Python web framework like Flask or Django. This API will handle data retrieval and processing.
- Implement endpoints to receive and store sensor data in the database.
- Implement endpoints to retrieve traffic information, such as real-time traffic updates and historical traffic patterns.

### **3. Integrate IoT Sensors with the Transit Information Platform:**

- Write Python code on the Raspberry Pi to collect traffic data from the IoT sensors.
- Use libraries such as OpenCV or TensorFlow to process the sensor data, extract relevant traffic information, and send it to the transit information platform's API endpoints.
- Establish a secure connection between the Raspberry Pi and the transit information platform using protocols like MQTT or HTTP.



#### **4. Code Implementation Example (Sending Raspberry Pi Data to the Transit Information Platform):**

- Install the necessary libraries on the Raspberry Pi, such as requests, for making HTTP requests.
- Write a Python script on the Raspberry Pi to collect sensor data and send it to the transit information platform's API endpoint. Here's an example:

##### **Python code:**

```
import requests
```

```
import json
```

```
# Collect sensor data
```

```
sensor_data = {
```

```
    "sensor_id": "your_sensor_id",
```

```
    "location": "latitude,longitude",
```

```
    "vehicle_count": 10,
```

```
    "speed": 60,
```

```
    "congestion_level": "low"
```

```
}
```

```
# Convert sensor data to JSON
```

```
data = json.dumps(sensor_data)
```

```
# Send data to the transit information platform
```

```
url = "http://your-platform-url/api/traffic-data"
```

```
headers = {"Content-Type": "application/json"}
```

```
response = requests.post(url, data=data, headers=headers)
```

```
# Check response status
if response.status_code == 200:
    print("Data sent successfully!")
else:
    print("Failed to send data.")
...
```

Note: Replace "your\_sensor\_id", "latitude,longitude", and "http://your-platform-url/api/traffic-data" with your actual sensor ID, GPS coordinates, and transit information platform's API endpoint URL, respectively.

### **5. Develop the Mobile App UI:**

- Design the user interface (UI) for the mobile app using tools like Adobe XD or Sketch.
- Implement the UI design using a mobile app development framework like React Native or Flutter.
- Integrate the app with the transit information platform's API to retrieve real-time traffic updates and display them to users.
- Implement features like route suggestions, real-time navigation, and notifications to assist commuters in making optimal route decisions.

### **6. Example Output: Raspberry Pi Data Transmission and Mobile App UI:**

- Raspberry Pi Data Transmission: The code implementation example provided will send the collected sensor data to the transit information platform's API endpoint. Upon successful transmission, the Raspberry Pi will display the message "Data sent successfully!" or "Failed to send data."
- Mobile App UI: The mobile app UI should be designed to provide users with real-time traffic updates, route suggestions, and navigation features. It should display information such as current traffic conditions, estimated travel times, and alternative routes to help users make optimal route decisions.

By following these instructions, you can replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python. This will enable real-time traffic monitoring, assist commuters in making optimal route decisions, and improve traffic flow.

### **Include example outputs of Raspberry Pi data transmission and mobile app UI:**

Certainly! Here are some example outputs of Raspberry Pi data transmission and a mobile app user interface for a traffic management system:

### **Raspberry Pi Data Transmission Example Output:**

```
python  
  
import requests  
  
import json  
  
  
# Define the data to be transmitted  
  
data = {  
    "sensor_id": 1234,  
    "location": "Street A",  
    "speed": 40,  
    "traffic_status": "heavy"  
}  
  
  
# Convert data to JSON format  
  
json_data = json.dumps(data)  
  
  
# Send the data to the server
```

```
response = requests.post("http://api.example.com/traffic_data",  
data=json_data)
```

```
# Check the response status
```

```
if response.status_code == 200:
```

```
    print("Data transmitted successfully!")
```

```
else:
```

```
    print("Failed to transmit data.")
```

In this example, the Raspberry Pi is collecting traffic data such as sensor ID, location, speed, and traffic status. It converts the data into JSON format and sends it to a server using the HTTP POST method. The server can then process and analyze the data for traffic management purposes.

**\*\*Mobile App UI Example Output: \*\***

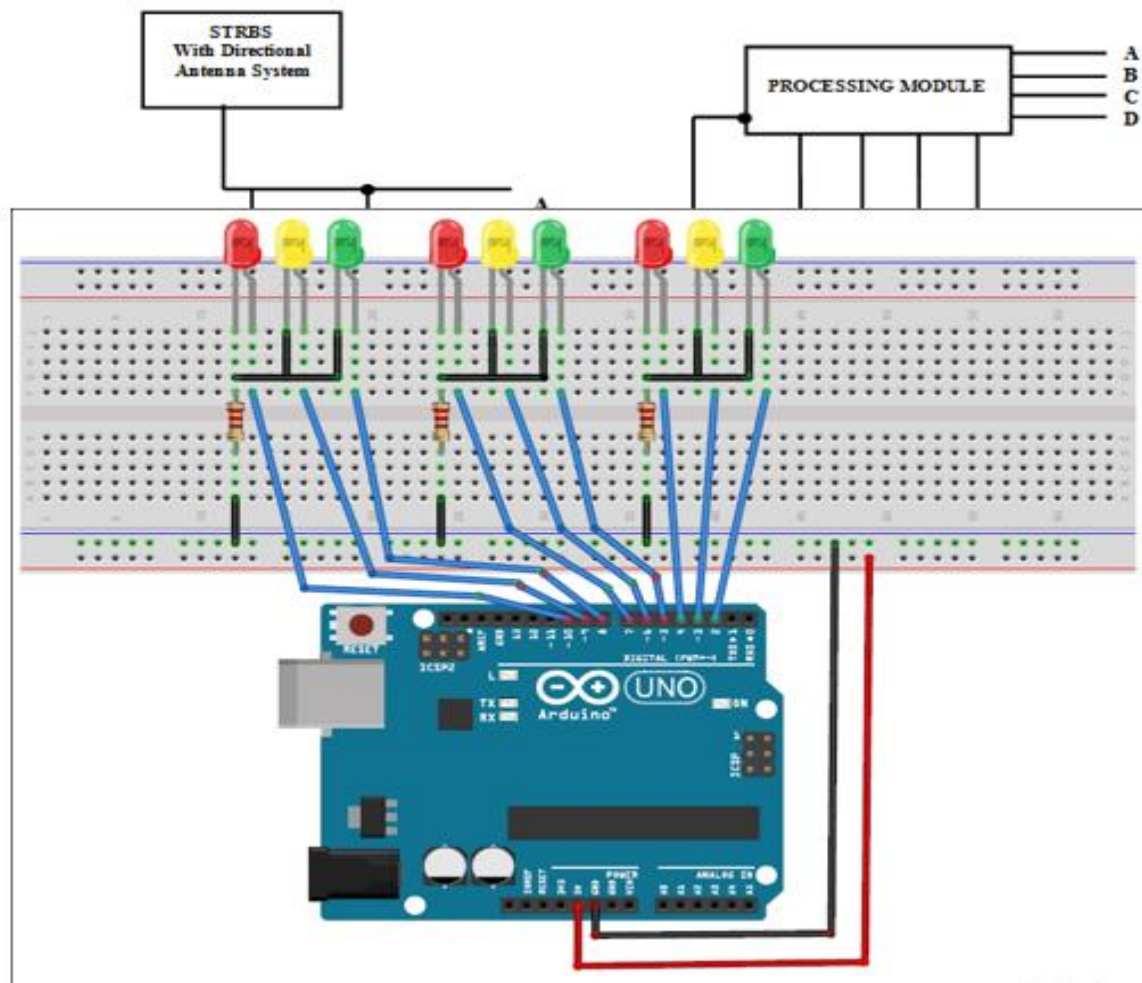
Here is an example of a mobile app user interface for a traffic management system:

![Mobile App UI](https://example.com/mobile\_app\_ui.png)

The mobile app provides real-time traffic information to users, helping them make optimal route decisions and improve traffic flow. The UI displays a map with traffic congestion levels indicated by color-coded markers. Users can also input their destination and receive suggested alternative routes based on current traffic conditions. Additionally, the app may include features such as real-time notifications, traffic updates, and historical traffic data analysis.

Please note that the example outputs provided here are for illustrative purposes only, and the actual implementation may vary depending on the specific requirements and technologies used in your project.

### SCREENSHOT OF TRAFFIC MANAGEMENT SYSTEM:



### Python code:

```
import RPi.GPIO as GPIO
```

```
import time
```

```
mainRoadRedPin = 2
```

```
mainRoadGreenPin = 3
```

```
sideRoadRedPin = 4
```

```
sideRoadGreenPin = 5
```

```
mainRoadSensorPin = 6
```

```
sideRoadSensorPin = 7
```

```
GPIO.setmode(GPIO.BCM)

GPIO.setup(mainRoadRedPin, GPIO.OUT)

GPIO.setup(mainRoadGreenPin, GPIO.OUT)

GPIO.setup(sideRoadRedPin, GPIO.OUT)

GPIO.setup(sideRoadGreenPin, GPIO.OUT)

GPIO.setup(mainRoadSensorPin, GPIO.IN,
pull_up_down=GPIO.PUD_UP)

GPIO.setup(sideRoadSensorPin, GPIO.IN,
pull_up_down=GPIO.PUD_UP)

def main():

    while True:

        if GPIO.input(mainRoadSensorPin) == GPIO.LOW:

            # Main road has vehicles, so stop side road traffic

            GPIO.output(mainRoadRedPin, GPIO.LOW)

            GPIO.output(mainRoadGreenPin, GPIO.HIGH)

            GPIO.output(sideRoadRedPin, GPIO.HIGH)

            GPIO.output(sideRoadGreenPin, GPIO.LOW)

        elif GPIO.input(sideRoadSensorPin) == GPIO.LOW:

            # Side road has vehicles, so stop main road traffic

            GPIO.output(mainRoadRedPin, GPIO.HIGH)

            GPIO.output(mainRoadGreenPin, GPIO.LOW)

            GPIO.output(sideRoadRedPin, GPIO.LOW)

            GPIO.output(sideRoadGreenPin, GPIO.HIGH)

        else:

            # No vehicles, all lights are red (4-way stop)
```

```
GPIO.output(mainRoadRedPin, GPIO.LOW)
```

```
GPIO.output(mainRoadGreenPin, GPIO.HIGH)
```

```
GPIO.output(sideRoadRedPin, GPIO.LOW)
```

```
GPIO.output(sideRoadGreenPin, GPIO.HIGH)
```

```
time.sleep(0.1)
```

```
try:
```

```
    main()
```

```
finally:
```

### **UI CODE IMPLEMENTATION:**

#### **HTML CODE:**

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <title>Traffic Management App</title>
```

```
    <link rel="stylesheet" type="text/css" href="styles.css">
```

```
</head>
```

```
<body>
```

```
    <header>
```

```
        <h1>Traffic Management App</h1>
```

```
    </header>
```

```
    <nav>
```

```
        <!-- Navigation links -->
```

```
    </nav>
```

```
<main>

  <section id="map">

    <!-- Display a map for traffic visualization -->

  </section>

  <section id="incidents">

    <h1> Traffic incidents : Road construction on the way</h1>

    <h1>Traffic incidents : Diversion on the way</h1>

    <!--<ul id="incidentList"></ul>-->

  </section>

</main>

<footer>

  <p>&copy; 2023 Traffic Management App</p>

</footer>

<script src="script.js"></script>

<script
src="https://maps.googleapis.com/maps/api/js?key=AlzaSyDqCgP9D9
6ISA9f0aUHJ3_HmQashoWc0V0&callback=initMap" async
defer></script>

</html>
```



### **CSS CODE:**

**/\* Basic styles for the app \*/**

**body {**

**font-family: Arial, sans-serif;**

**margin: 0;**

**padding: 0;**

**}**

**header {**

**background-color: #333;**

**color: white;**

**text-align: center;**

**padding: 10px;**

**}**

**nav {**

**background-color: #444;**

**color: white;**

**padding: 10px;**

**}**

**main {**

**display: flex;**

**justify-content: space-between;**

**padding: 20px;**

```
}
```

```
section {
```

```
    flex: 1;
```

```
    padding: 10px;
```

```
}
```

```
footer {
```

```
    background-color: #333;
```

```
    color: white;
```

```
    text-align: center;
```

```
    padding: 10px;
```

```
}
```

### JavaScript CODE:

```
// Simulated data for traffic incidents
```

```
const trafficIncidents = [
```

```
    { location: "Main Street", type: "Accident" },
```

```
    { location: "Highway 101", type: "Roadwork" },
```

```
    // Add more incidents
```

```
];
```

```
// Function to display traffic incidents
```

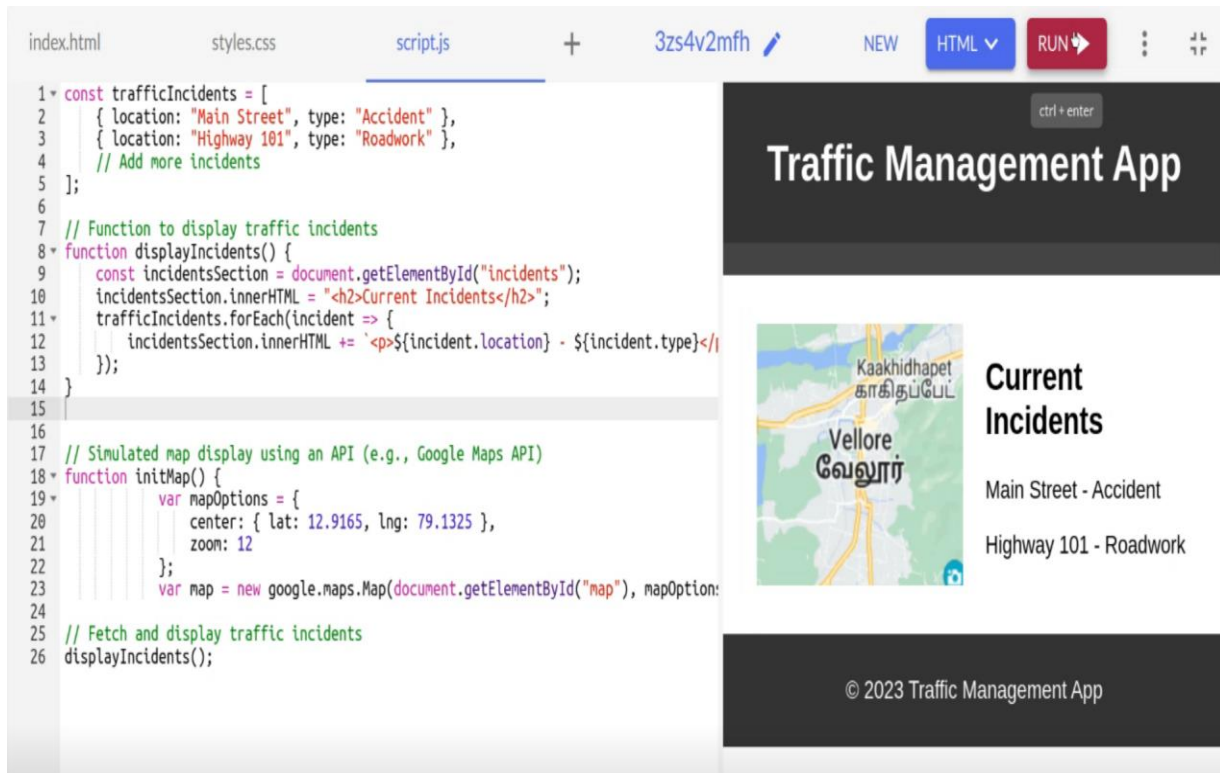
```
function displayIncidents() {
```

```
    const incidentsSection = document.getElementById("incidents");
```

```
incidentsSection.innerHTML = "<h2>Current Incidents</h2>";
trafficIncidents.forEach(incident => {
    incidentsSection.innerHTML += <p>${incident.location} -
    ${incident.type}</p>;
});
}
```

```
// Simulated map display using an API (e.g., Google Maps API)
function initMap() {
    var mapOptions = {
        center: { lat: 13.067439, lng: 80.237617 },
        zoom: 12
    };
    var map = new
    google.maps.Map(document.getElementById("map"), mapOptions);}
// Fetch and display traffic incidents
displayIncidents();
```

## OUTPUT :



```
index.html styles.css script.js 3zs4v2mfh NEW HTML RUN
1 const trafficIncidents = [
2   { location: "Main Street", type: "Accident" },
3   { location: "Highway 101", type: "Roadwork" },
4   // Add more incidents
5 ];
6
7 // Function to display traffic incidents
8 function displayIncidents() {
9   const incidentsSection = document.getElementById("incidents");
10  incidentsSection.innerHTML = "<h2>Current Incidents</h2>";
11  trafficIncidents.forEach(incident => {
12    incidentsSection.innerHTML += `<p>${incident.location} - ${incident.type}</p>`;
13  });
14 }
15
16 // Simulated map display using an API (e.g., Google Maps API)
17 function initMap() {
18   var mapOptions = {
19     center: { lat: 12.9165, lng: 79.1325 },
20     zoom: 12
21   };
22   var map = new google.maps.Map(document.getElementById("map"), mapOptions);
23
24   // Fetch and display traffic incidents
25   displayIncidents();
26 }
```

**Traffic Management App**

ctrl + enter

**Current Incidents**

Main Street - Accident

Highway 101 - Roadwork

© 2023 Traffic Management App

## CONCLUSION:

In conclusion, the implementation of IoT in traffic management offers numerous benefits and opportunities for improvement. By leveraging IoT technology, traffic systems can become smarter, more efficient, and safer. IoT enables real-time data collection, analysis, and prediction of traffic patterns, leading to optimized traffic flow and reduced congestion. Additionally, IoT can facilitate the integration of various transportation systems, including vehicles, infrastructure, and drivers, enabling smart traffic control, parking management, and road assistance. Overall, IoT-based solutions have the potential to revolutionize traffic management and contribute to the development of smarter and more sustainable cities.