

PROJECT

INTERNET OF THINGS

TITLE: TRAFFIC MANAGEMENT SYSTEM

SUBMITTED BY:

NAME: SYED DAWOOD ASIK S

REGISTER NUMBER: 911921104304

NAAN MUDHALVAN ID: **aut2291190004**

DEPARTMENT: B.E (COMPUTER SCIENCE ENGINEERING)

TRAFFIC MANAGEMENT SYSTEM

PROJECT DEFINITION: The project involves using IoT devices and data analytics to monitor traffic flow and congestion in real-time, providing commuters with access to this information through a public platform or mobile apps. The objective is to help commuters make informed decisions about their routes and alleviate traffic congestion. This project includes defining objectives, designing the IoT traffic monitoring system, developing the traffic information platform, and integrating them using IoT technology and python.

PHASE 1: PROJECT DEFINITION AND DESIGN THINKING

DESIGN THINKING:

1. PROJECT OBJECTIVES:

- **Define objectives such as real-time traffic monitoring:**
 - Estimation of traffic bulkiness performed using real time video feed.
 - Vehicle recognition in order to differentiate between emergency and non-emergency vehicles.
 - The system proposed relies on the efficient det lite model, which later provides output in order to manage light switching for each lane.
- **Congestion detection:**
 - The only way a sender can guess that congestion has happened is the need to retransmit a segment.
 - Retransmission is needed to recover a missing packet that is assumed to have been dropped by a router due to congestion.
- **Route optimization:**
 - Route optimization is the process of planning the fastest and most cost-effective way for your mobile field service workers to get from one appointment to another.

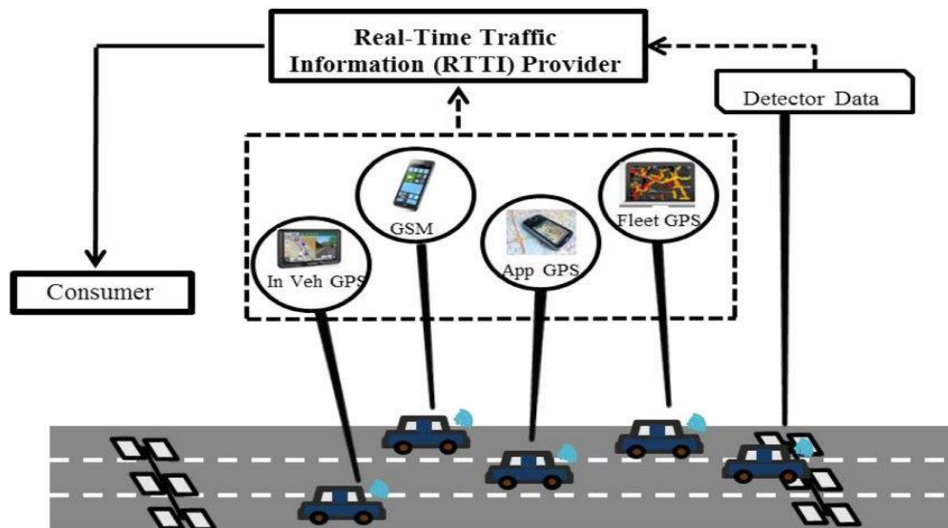
2. IoT SENSOR DESIGN:

- **Plan the deployment of IoT devices (sensors) to monitor traffic flow and congestion:**

- Sensors and cameras are deployed across cities to monitor vehicle movement and congestion levels.

3. REAL-TIME TRANSIT INFORMATION PLATFORM:

- Design a web-based platform and mobile apps to display real-time traffic information to the public:



- Real time traffic information helps you to keep a check on the traffic situation so that you can better plan your route.

4. INTEGRATION APPROACH:

- Design a web-based platform and mobile apps to display real-time traffic information to the public:
 - Data is available on the TomTom map or via openLR format for map-agnostic delivery.
 - When your website sends or receives information from another application, system, or website.

PHASE 2: INNOVATION

Consider integrating historical traffic data and machine learning algorithms to predict congestion patterns.

HISTORICAL TRAFFIC DATA:

SUMMARY:

Provides information about the historical traffic information stored in the network dataset such as the speed profile table and time slice durations.

PROPERTIES:

| PROPERTY | EXPLANATION | DATASET |
|---------------------------------|---|---------|
| Time interval | The time interval of the traffic data. | Double |
| Time interval units | The units of the time interval of the traffic data. This property returns the following keywords: <ul style="list-style-type: none">• Millisecond• Second• Minutes• Hours• Days• Weeks• Months• Years• Decades• Centuries• Unknown | String |
| First time slice field name | The field name of the first time slice of the given period in the profile table. | String |
| Last time slice field name | The field name of the first time slice of the given period in the profile table. | String |
| First slice duration in minutes | The duration of the time slice in minutes. | Integer |
| First time slice start time | The start time of valid period of day for traffic data. | String |
| Profiles table name | The name of the table containing profiles. | String |

| | | |
|--|---|--------|
| Join table name | The name of the table containing profiles. | String |
| Join table base travel time field name | The field name for base travel time in the join table. | String |
| Join table base speed field name | The field name for base speed in the join table. | String |
| Join table base travel time field name | The field name for base travel time in the join table. | String |
| Join table base travel time units | The units for the base travel time in the join table. This property Returns the following keywords: <ul style="list-style-type: none"> • Seconds • Minutes • Hours • Days | String |
| Join table profile Id field names | A python list containing field names of the join table pointing to speed profiles. | List |
| Join table base speed units | The units for the base speed in the join table. This property returns the following keywords: <ul style="list-style-type: none"> • Miles Per Hour • Kilometers Per Hours • Unknown | String |

Code sample:

Historical traffic data properties example

Display a summary of the historical traffic information for the network dataset.

```
import arcpy
import sys
{
arcpy.env.workspace = "C:/Data/SanFrancisco.gdb/Transportation"
desc = arcpy.Describe("Streets_ND") if
desc.supportsHistoricalTrafficData: traffic =
desc.historicalTrafficData else:
    print("No historical traffic information")
    sys.exit() {
print("Historical Traffic Information ----") print("Time interval: " ,
traffic.timeInterval) print("Time interval units: " ,
traffic.timeIntervalUnits) print("First time slice field name: " ,
traffic.firstTimeSliceFieldName) print("Last time slice field name: " ,
traffic.lastTimeSliceFieldName) print("First time slice start time: " ,
traffic.firstTimeSliceStartTime) print("Time slice duration in minutes:
",traffic.timeSliceDurationInMinutes) print("Profiles table name: ",
traffic.profilesTableName) print("Join table name: ",
traffic.joinTableName) print("Join table base travel time field name: ",
traffic.joinTableBaseTravelTimeFieldName) print("Join table base travel
```

```
time units: ", traffic.joinTableBaseTravelTimeUnits) print("Join table
Profile ID field names: ", traffic.joinTableProfileIDFieldNames)
}
};
```

OUTPUT:

| Vehicle_ID | Road_ID | Time_group | Start_time | End_time | Travel_time (min) | Velocity (km/min) |
|------------|---------|------------|------------|----------|----------------------|----------------------|
| 1 | 1 | 6 | 16:50 | 16:57 | 7 | 1.8725 |
| 2 | 1 | 6 | 17:20 | 17:31 | 11 | 1.1916 |
| 3 | 1 | 6 | 17:43 | 17:56 | 13 | 1.0082 |
| 4 | 1 | 6 | 16:02 | 16:11 | 9 | 1.456 |
| 5 | 1 | 6 | 16:16 | 16:32 | 16 | 0.8192 |
| 6 | 1 | 6 | 16:05 | 16:18 | 13 | 1.0082 |
| 7 | 1 | 6 | 17:03 | 17:10 | 7 | 1.8725 |
| 8 | 1 | 6 | 17:11 | 17:18 | 7 | 1.8725 |
| 9 | 1 | 6 | 17:35 | 17:46 | 11 | 1.1916 |
| 10 | 1 | 6 | 16:09 | 16:16 | 7 | 1.8725 |

MACHINE LEARNING ALGORITHMS TO PREDICT CONGESTION PATTERNS IN TRAFFIC DATA:

There are several types of machine learning algorithms that can be used for traffic prediction, including regression, time-series analysis, and artificial neural networks. Regression models use historical traffic data to predict future traffic conditions based on past trends.

SAMPLE CODE:

1. **import** numpy as np
2. **import** pandas as pd
3. **import** matplotlib.pyplot as plt
4. **import** seaborn as sns
5. **import** datetime
6. **import** tensorflow

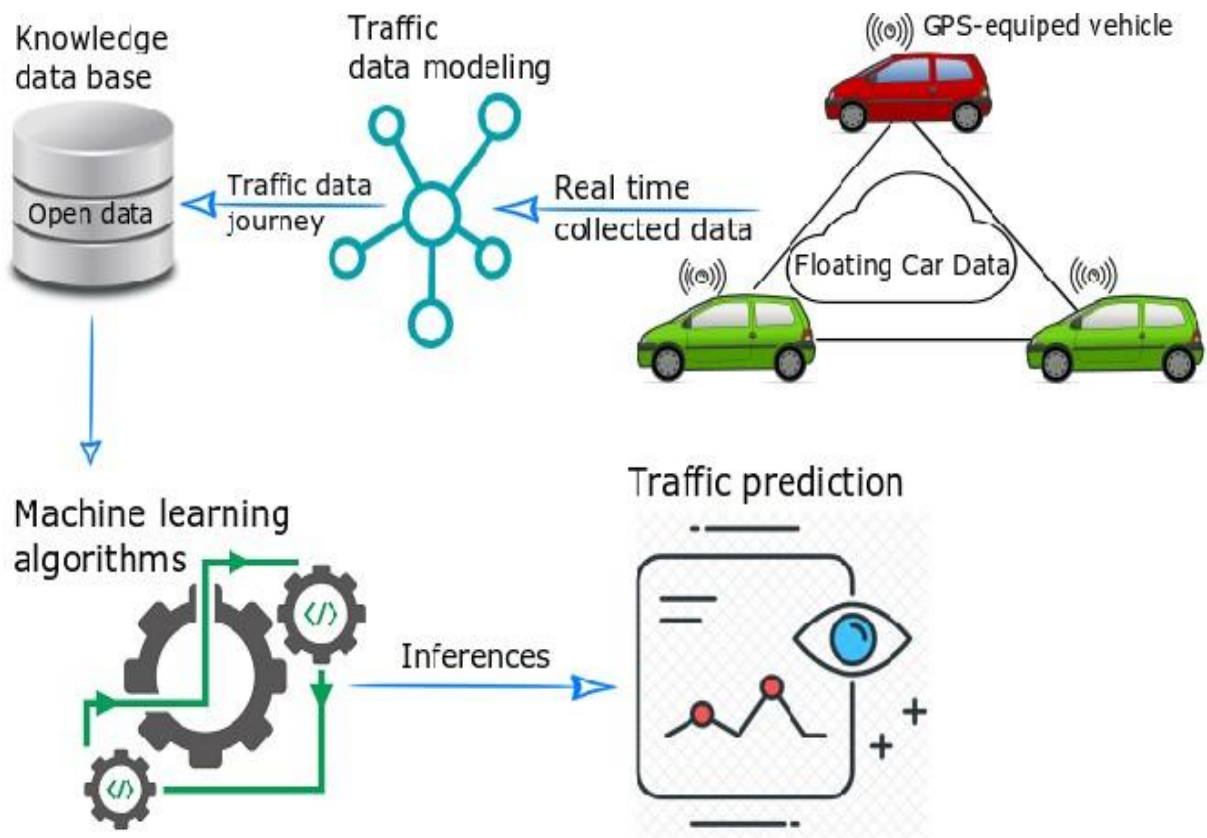
7. from statsmodels.tsa.stattools **import** adfuller
8. from sklearn.preprocessing **import** MinMaxScaler
9. from tensorflow **import** keras
10. from keras **import** callbacks
11. from tensorflow.keras **import** Sequential
12. from tensorflow.keras.layers **import** Conv2D, Flatten, Dense, LSTM, Dropout, GRU, Bidirectional
13. from tensorflow.keras.optimizers **import** SGD
14. **import** math
15. from sklearn.metrics **import** mean_squared_error
- 16.
17. **import** warnings
18. warnings.filterwarnings("ignore")

Loading the Dataset

1. dataset = pd.read_csv("traffic.csv")
2. dataset.head()

Output:

| | DateTime | Junction | Vehicles | ID |
|----------|---------------------|-----------------|-----------------|-------------|
| 0 | 2015-11-01 00:00:00 | 1 | 15 | 20151101001 |
| 1 | 2015-11-01 01:00:00 | 1 | 13 | 20151101011 |
| 2 | 2015-11-01 02:00:00 | 1 | 10 | 20151101021 |
| 3 | 2015-11-01 03:00:00 | 1 | 7 | 20151101031 |
| 4 | 2015-11-01 04:00:00 | 1 | 9 | 20151101041 |



PHASE 3: DEVELOPMENT PART 1

START BUILDING THE IOT TRAFFIC MONITORING SYSTEM:

1. Define the Objectives and Requirements:

- Determine the purpose of the traffic monitoring system (e.g., managing parking lots, optimizing traffic flow, security, etc.).
- List the specific requirements, such as the number of cameras, sensors, data storage capacity, and user interfaces.

2. Choose Hardware and Sensors:

- Select cameras, sensors, and other hardware components based on your requirements.
- Consider using video cameras, infrared sensors, ultrasonic sensors, or other appropriate technologies to monitor traffic.

3. Set Up Data Collection:

- Install and configure cameras and sensors to capture data from the parking lot.
- Ensure that the hardware is properly connected to the monitoring system.

4. Data Transmission:

- Establish a reliable data transmission mechanism to send data from cameras and sensors to the central monitoring system.
- Use wired or wireless communication depending on the infrastructure and range.

5. Data Processing:

- Develop software for real-time data processing. This can include image recognition and analysis to count vehicles, detect parking violations, or identify security issues.

6. Data Storage:

- Design a data storage solution to store historical data for analysis and reporting.
- Consider using a database system for efficient data management.

7. Real-time Monitoring and Alerts:

- Create a dashboard or user interface for real-time monitoring.
- Implement alerting mechanisms for detecting unusual traffic patterns, security breaches, or parking violations.

8. Reporting and Analytics:

- Develop reporting features that allow users to generate traffic reports, occupancy statistics, and other insights.
- Consider implementing analytics to optimize traffic flow and improve the efficiency of the parking lot.

9. User Interfaces:

- Create user-friendly interfaces for administrators, operators, and end-users.
- Ensure that the system is accessible through web or mobile applications.

10. Security:

- Implement security measures to protect data and ensure the system's integrity.
- Use encryption, access controls, and authentication mechanisms.

11. Integration:

- If required, integrate the monitoring system with other systems, such as access control systems, payment systems, or security systems.

12. Testing and Optimization:

- Thoroughly test the system to identify and resolve any issues.
- Optimize the system's performance and accuracy.

13. Scalability:

- Plan for scalability as traffic monitoring needs may change over time.

14. Documentation:

- Create comprehensive documentation for users and maintainers.

15. Maintenance and Support:

- Establish a maintenance plan to ensure the system's continued functionality.

16. Compliance:

- Ensure that the system complies with relevant regulations and privacy laws.

17. Training:

- Train operators and users on how to use the system effectively.

18. Data Privacy:

- Address data privacy concerns and ensure that data is handled in a compliant and ethical manner.

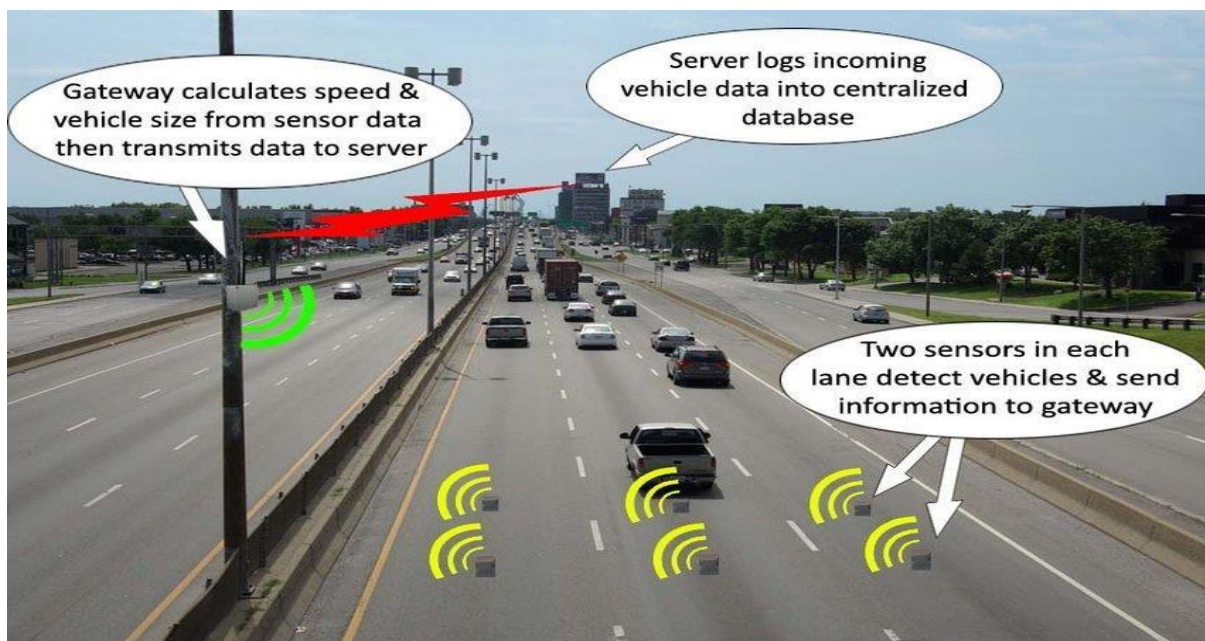
19. Deployment:

- Deploy the system in the parking lot, and monitor its performance in a real-world environment.

20. Continuous Improvement:

- Continuously gather feedback and make improvements to the system based on user and operational experiences.
- Remember that building a lot traffic monitoring system can be a complex project, and it's essential to plan, design, and implement it carefully to meet your specific needs and requirements. Depending on your resources and expertise, you may also consider partnering with experts or companies specializing in traffic monitoring

SAMPLE PICTURE:



SAMPLE PROGRAM:

```
Import necessary libraries
import datetime
import random
class ParkingLot:
    def __init__(self, capacity):
```

```

self.capacity = capacity
self.occupancy = 0
self.vehicles = []
def is_full(self):
    return self.occupancy >= self.capacity

def add_vehicle(self, vehicle_id):
    if not self.is_full():
        self.occupancy += 1
    self.vehicles.append(
        {
            'vehicle_id': vehicle_id,
            'entry_time': datetime.datetime.now()
        })
    print(f"Vehicle {vehicle_id} entered the parking lot.")

def remove_vehicle(self, vehicle_id):
    for vehicle in self.vehicles:
        if vehicle['vehicle_id'] == vehicle_id:
            self.occupancy -= 1
            entry_time = vehicle['entry_time']
            exit_time = datetime.datetime.now()
            duration = exit_time - entry_time
            print(f"Vehicle {vehicle_id} exited the parking lot. Parking duration: {duration}")
    self.vehicles.remove(vehicle)

    return

def main():
    capacity = 50 # Set the parking lot capacity
    parking_lot = ParkingLot(capacity)
    while True:
        if not parking_lot.is_full():
            vehicle_id = random.randint(1, 100)

```

```
parking_lot.add_vehicle(vehicle_id)
if parking_lot.occupancy > 0:
    vehicle_id = random.choice(parking_lot.vehicles)['vehicle_id']
parking_lot.remove_vehicle(vehicle_id)

if __name__ == "__main__":
    main()
```

OUTPUT:

Vehicle 12 entered the parking lot.

Vehicle 58 entered the parking lot.

Vehicle 94 entered the parking lot.

Vehicle 89 exited the parking lot. Parking duration: 0:00:05.171821

Vehicle 29 exited the parking lot. Parking duration: 0:00:03.435076 Vehicle 12 exited
the parking lot. Parking duration: 0:00:01.857361

Vehicle 35 entered the parking lot.

PHASE 4: CONTINUE BUILDING THE PROJECT BY DEVELOPING THE TRAFFIC INFORMATION PLATFORM AND MOBILE APPS.

DEVELOPMENT PART 2:

PROJECT PLANNING:

- Define the project scope, objectives, and requirements.
- Create a detailed project plan with milestones and timelines.
- Allocate resources and budget for development.

MARKET RESEARCH:

- Analyze the target audience and their needs.
- Study competitors and identify unique features for your platform and apps.



DESIGN:

- Develop wireframes and prototypes for the mobile apps.
- Design the user interface (UI) and user experience (UX) for both the platform and apps.

BACKEND DEVELOPMENT:

- Set up the server infrastructure for the traffic information platform.
- Build a robust backend to handle data collection, storage, and processing.



DATA PROCESSING:

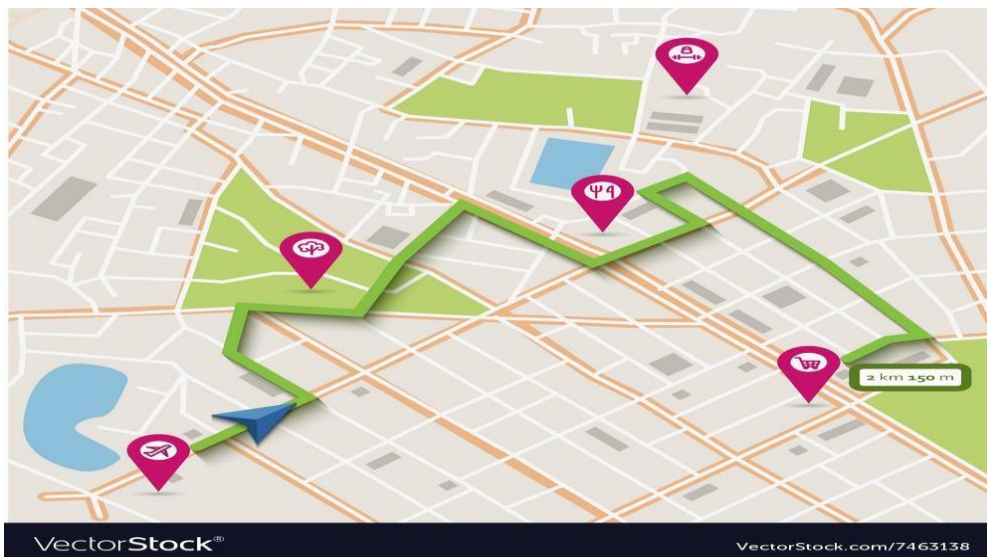
- Process and analyze the collected data to provide accurate traffic information.

USER AUTHENTICATION:

- Implement secure user registration and login mechanisms.

MAPPING AND NAVIGATION:

- Integrate mapping services and navigation features in the mobile apps.



ALERTS AND NOTIFICATIONS:

- Implement push notifications for traffic alerts and updates.



TESTING AND QUALITY ASSURANCE:

- Conduct thorough testing to identify and fix bugs.
- Ensure the platform and apps work well on different devices and screen sizes.

DEPLOYMENT:

- Deploy the platform to a web server and make the mobile apps available on app stores.

MARKETING AND PROMOTION:

- Develop a marketing strategy to promote your traffic information platform and mobile apps.

MAINTENANCE AND UPDATES:

- Provide ongoing support, maintenance, and regular updates to improve functionality and fix issues.



TRAFFIC INFO PLATFORM & APPS

PLATFORM DEVELOPMENT:

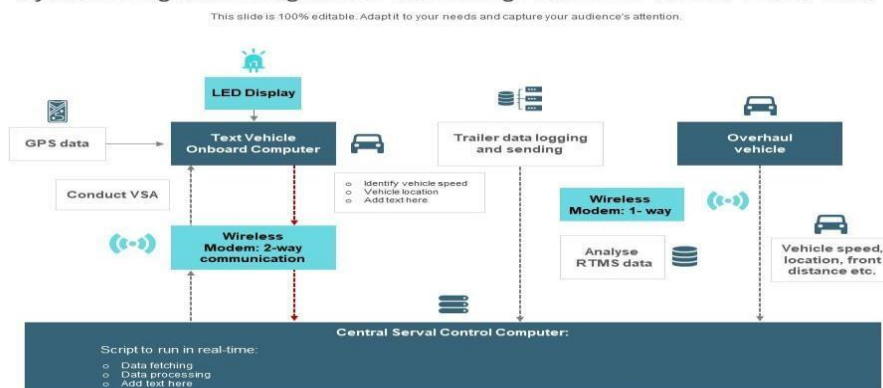
- Design the architecture for the traffic information platform.
- Develop the backend infrastructure, including servers, databases, and APIs for data collection and storage.
- Implement real-time data collection mechanisms, such as sensors, traffic cameras, and GPS systems.
- Utilize data analytics to process and analyze traffic data for insights.



DATA INTEGRATION:

- Integrate traffic data sources such as government agencies, navigation services, and user-generated content.
- Ensure data accuracy, reliability, and real-time updates.

System Integration Diagram for Monitoring Real Time Vehicle Traffic Data



MAPPING AND NAVIGATION:

- Implement mapping functionality for navigation within the mobile apps.
- Include features like turn-by-turn directions, traffic alerts, and alternative route suggestions.

USER ENGAGEMENT:

- Incorporate features to engage users, such as social sharing, user reviews, and community-driven updates.

TESTING AND QUALITY ASSURANCE:

- Conduct thorough testing of the platform and mobile apps to identify and resolve any bugs or issues.

MARKETING AND USER ADOPTION:

- Promote the mobile apps to attract users through various marketing strategies and advertising.

DEVELOPING A TRAFFIC INFORMATION PLATFORM AND MOBILE APPS INVOLVES SEVERAL KEY COMPONENTS AND CONSIDERATIONS:

DATA PROCESSING AND ANALYSIS:

- Implement algorithms to process and analyze traffic data, identifying congestion, accidents, and road closures.
- Utilize machine learning or data analytics to predict traffic patterns.

APIS FOR MOBILE APPS:

- Create APIs that mobile apps can use to fetch traffic data and interact with the platform.

MOBILE APPS:

PLATFORM SELECTION:

- Decide whether to develop separate apps for iOS and Android or opt for crossplatform development.

USER INTER FACE (UI) AND USER EXPERIENCE (UX):

- Design an intuitive and user-friendly interface.
- Consider features like real-time maps, route planning, and traffic alerts.

REAL-TIME UPDATES:

- Implement features to provide real-time traffic information.
- Notify users of accidents, road closures, and alternative routes.

CUSTOMIZATION:

- Allow users to customize their routes, preferences, and traffic alert settings.

SOCIAL COMMUNITY FEATURES:

- Include social sharing options, user reviews, and the ability for users to report incidents.

OFFLINE FUNCTIONALITY:

- Offer offline maps and route planning for users with limited internet connectivity.

SAMPLE CODE:

```
def traffic_advice(traffic_severity):  
    if  
    traffic_severity == "low":  
        return "Traffic is light. You can stay on your current route."  
  
    elif traffic_severity == "moderate":  
        return "Traffic is moderate. Consider an alternative route if available."  
  
    elif traffic_severity == "high":  
        return "Traffic is heavy. It's advisable to take an alternative route."  
  
    else:  
        return "Invalid traffic information. Please provide 'low', 'moderate', or 'high'."
```

Input the traffic severity (e.g., 'low', 'moderate', or 'high')

traffic_severity = input("Enter traffic severity: ") advice =

traffic_advice(traffic_severity) print(advice)

-THANK YOU