Assignment –1

Problem 3: Real-Time Traffic Monitoring System

1. Data Flow Diagram

Data Flow Steps:

1. User Input:The user provides input, including the starting point and destination.

2. API Request: The system sends a request to the traffic monitoring API, including the starting point, destination, and any other relevant parameters.

3. Traffic API: The API processes the request and returns real-time traffic data, including current traffic conditions, estimated travel time, and any incidents or delays.

4. Data Parsing: The system parses the received traffic data to extract relevant information.

5. Display Traffic Data: The system displays current traffic conditions, travel time, and alternative routes to the user.

6. Alternative Routes: If the current route has significant delays, the system suggests alternative routes based on traffic data.

Data Flow Diagram (DFD):

[User] ---> (Enter Starting Point & Destination) ---> [Traffic Monitoring System] ---> (Send Request) ---> [Traffic API]

[Traffic API] ---> (Return Data) ---> [Traffic Monitoring System] ---> (Parse Data) ---> [Display Traffic Conditions/Alternative Routes] ---> [User]

2. Pseudocode for Traffic Monitoring System

Here is a high-level pseudocode outline:

BEGIN

PROMPT user to input starting point and destination

SEND request to traffic API with the starting point and destination

RECEIVE response from the traffic API

PARSE the API response to extract:

- current traffic conditions

- estimated travel time

- any incidents or delays

DISPLAY the current traffic conditions, estimated travel time, and any incidents to the user

IF there are significant delays:

SUGGEST alternative routes based on traffic data

DISPLAY alternative routes with travel times

REPEAT the process to keep the traffic data updated in real-time

END

```

3. Python Implementation

Below is a Python example using the Google Maps API to fetch and display real-time traffic data. You will need to replace `'your\_google\_maps\_api\_key'` with your actual API key from Google Cloud.

Install Required Libraries:

First, make sure you have the necessary Python libraries installed:

```bash

pip install requests

```

Example Code:

```python

import requests

# Define API Key and base URL for Google Maps Traffic API

API\_KEY = 'your\_google\_maps\_api\_key'

BASE\_URL = 'https://maps.googleapis.com/maps/api/directions/json'

def get\_traffic\_data(start, destination):

# Set up parameters for API request

params = {

'origin': start,

'destination': destination,

'key': API\_KEY,

'departure\_time': 'now', # Use 'now' for real-time traffic data

'traffic\_model': 'best\_guess'

}

# Send request to Google Maps Directions API

response = requests. Get (BASE\_URL, params=params)

# Parse response JSON

data = response. Json()

# Check for successful response

if response.status\_code == 200 and 'routes' in data:

route = data['routes'][0]

legs = route['legs'][0]

duration\_in\_traffic = legs['duration\_in\_traffic']['text']

distance = legs['distance']['text']

summary = route['summary']

print (f"Route: {summary}")

print (f"Distance: {distance}")

print (f"Estimated Travel Time (with traffic): {duration\_in\_traffic}")

# Check for alternative routes if there are delays

if len(data['routes']) > 1:

print ("\nAlternative Routes:")

for i, alt\_route in enumerate(data['routes'][1:], 1):

alt\_legs = alt\_route['legs'][0]

alt\_duration = alt\_legs['duration\_in\_traffic']['text']

alt\_summary = alt\_route['summary']

print(f"{i}. Route: {alt\_summary}, Estimated Travel Time: {alt\_duration}")

else:

print ("Error: Unable to fetch traffic data. Please check the starting point and destination.")

def main ():

# Prompt user for starting point and destination

start = input ("Enter the starting point: ")

destination = input ("Enter the destination: ")

# Fetch and display traffic data

get\_traffic\_data(start, destination)

if \_\_name\_\_ == "\_\_main\_\_":

main ()

4. Documentation of API Integration

API Used: [Google Maps Directions API] (https://developers.google.com/maps/documentation/directions/start)

Methods Used:

- `requests. Get () `: Sends a GET request to the Google Maps Directions API with the specified parameters, including origin, destination, and API key.

- `response. Json () `: Parses the JSON response returned by the API into a Python dictionary.

Traffic Data Fields:

- \*\*`routes [0] ['legs'][0] ['duration\_in\_traffic'] ['text'] `\*\*: Estimated travel time considering current traffic conditions.

- \*\*`routes [0] ['legs'][0] ['distance'] ['text'] `\*\*: Distance between the origin and destination.

- \*\*`routes [0] ['summary'] `\*\*: Summary of the route.

5. Explanation of Assumptions and Potential Improvements

Assumptions:

- The user provides valid starting point and destination inputs that the Google Maps API can recognize.

- The API key is valid and has sufficient access to make requests.

Potential Improvements:

Error Handling: Improve error handling for various scenarios, such as network errors, invalid inputs, or quota limitations.

User Interface: Implement a graphical user interface (GUI) for better user experience, potentially using Tkinter or a web-based framework like Flask.

Real-Time Updates: Implement real-time updates that continuously check for changes in traffic conditions and notify the user of any significant updates.

Multi-Mode Transportation: Extend the application to support different modes of transportation, such as walking, biking, or public transit, and provide traffic data accordingly.

This approach provides a solid foundation for a real-time traffic monitoring system with potential for further enhancements based on the smart city's requirements.

## Output:

Enter the starting point: Times Square, New York

Enter the destination: Central Park, New York

Route: 7th Ave

Distance: 2.1 km

Estimated Travel Time (with traffic): 7 mins

Alternative Routes:

1. Route: Broadway, Estimated Travel Time: 8 mins

2. Route: 5th Ave, Estimated Travel Time: 9 mins