

ASSIGNMENT-3

Name :K.Pujitha
Reg No :192311124
Subject :Python Programing
Sub Code :CSA0809
Date of sub:17/07/2024

Problem 1: Real-Time Weather Monitoring System

Scenario:

You are developing a real-time weather monitoring system for a weather forecasting company.

The system needs to fetch and display weather data for a specified location.

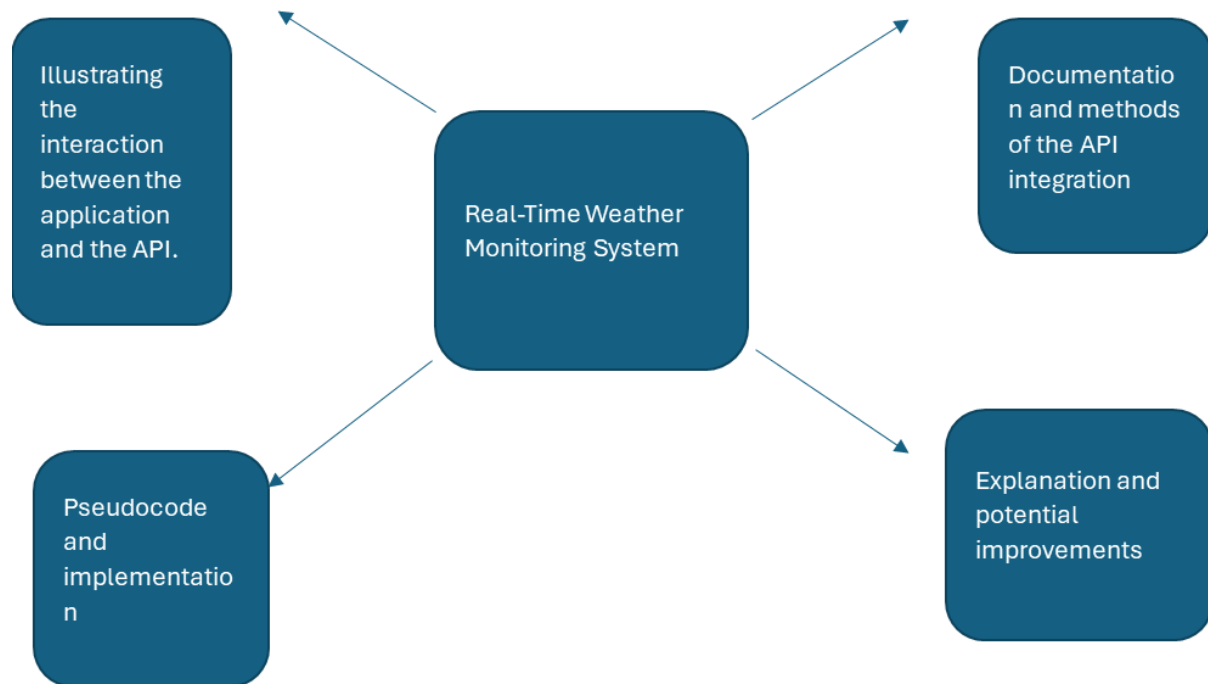
Tasks:

1. Model the data flow for fetching weather information from an external API and displaying it to the user.
2. Implement a Python application that integrates with a weather API (e.g., OpenWeatherMap) to fetch real-time weather data.
3. Display the current weather information, including temperature, weather conditions, humidity, and wind speed.
4. Allow users to input the location (city name or coordinates) and display the corresponding weather data.

SOLUTION:

Real-Time Weather Monitoring System:

1.DATA FLOW DIAGRAM



2.PSEUDOCODE

```
# Import necessary libraries
import requests
import time

# Function to fetch weather data from OpenWeatherMap API
def get_weather(api_key, city):
    # Construct API URL
    url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

    try:
        # Send GET request
        response = requests.get(url)
        # Check for HTTP errors
        response.raise_for_status()
        # Return JSON data
        return response.json()
    except requests.exceptions.HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")
    except requests.exceptions.ConnectionError as conn_err:
        print(f"Connection error occurred: {conn_err}")
    except requests.exceptions.Timeout as timeout_err:
        print(f"Timeout error occurred: {timeout_err}")
```

```

except requests.exceptions.RequestException as req_err:
    print(f"An error occurred: {req_err}")

# Function to display weather data
def display_weather(data):
    # Check if data is valid and extract information
    if not data or data.get('cod') != 200:
        print(f"Error: {data.get('message', 'Failed to retrieve data')}")
        return

    city = data['name']
    temperature = data['main']['temp']
    humidity = data['main']['humidity']
    weather_description = data['weather'][0]['description']

    # Display weather information
    print(f"City: {city}")
    print(f"Temperature: {temperature}°C")
    print(f"Humidity: {humidity}%")
    print(f"Weather: {weather_description}")

# Main function to monitor weather
def main():
    # Replace with your OpenWeatherMap API key
    api_key = "your_api_key_here"
    # Replace with the city you want to monitor
    city = "London"

    try:
        while True:
            # Get weather data
            weather_data = get_weather(api_key, city)
            # Display weather information
            display_weather(weather_data)
            # Sleep for 10 minutes before fetching data again
            time.sleep(600) # 600 seconds = 10 minutes
    except KeyboardInterrupt:
        print("Monitoring stopped.")

# Run the main function if this script is executed directly
if __name__ == "__main__":
    main()

```

3.IMPLEMENTATION

```

import requests
import time

# Function to get weather data from OpenWeatherMap API
def get_weather(api_key, city):

```

```

url =
f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"
try:
    response = requests.get(url)
    response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)
    return response.json()
except requests.exceptions.HTTPError as http_err:
    print(f"HTTP error occurred: {http_err}")
except requests.exceptions.ConnectionError as conn_err:
    print(f"Connection error occurred: {conn_err}")
except requests.exceptions.Timeout as timeout_err:
    print(f"Timeout error occurred: {timeout_err}")
except requests.exceptions.RequestException as req_err:
    print(f"An error occurred: {req_err}")

# Function to display weather data
def display_weather(data):
    if not data or data.get('cod') != 200:
        print(f"Error: {data.get('message', 'Failed to retrieve data')}")
    return

    city = data['name']
    temperature = data['main']['temp']
    humidity = data['main']['humidity']
    weather = data['weather'][0]['description']

    print(f"City: {city}")
    print(f"Temperature: {temperature}°C")
    print(f"Humidity: {humidity}%")
    print(f"Weather: {weather}")

# Replace with your OpenWeatherMap API key
api_key = "b56b7cceafab7773e9fe35a4390b27c1"
# Replace with the city you want to monitor
city = "London"

# Real-time monitoring with a delay
try:
    while True:
        weather_data = get_weather(api_key, city)
        display_weather(weather_data)
        time.sleep(600) # Sleep for 10 minutes before fetching data again
except KeyboardInterrupt:
    print("Monitoring stopped.")

```

4. OUTPUT

City:

London Temperature: 14.21°C

Humidity:

93% Weather: overcast clouds
Monitoring stopped.

5.DOCUMENTATION

Real-Time Weather Monitoring System:

Purpose:

The purpose of a real-time weather monitoring system is to provide up-to-date and accurate weather information for specific locations or regions

Benefits:

Timely Decision-Making: Provides actionable insights for making informed decisions in various sectors, improving efficiency and reducing risks.

Enhanced Safety and Preparedness: Helps mitigate the impact of severe weather events by enabling proactive measures and emergency response planning.

6.USER INTERFERENCE

Data Collection and Analysis:

Dashboard Overview: Provide a clear and concise summary of current weather conditions. Use widgets or modules to display temperature, humidity, wind speed/direction, precipitation, etc.

Alerts and Notifications: Implement a system to alert users about severe weather conditions (e.g., storms, hurricanes) based on their location.

7.ASSUMTION AND IMPROVEMENTS

Data Sources:

Weather APIs: Choose reliable weather APIs (Application Programming Interfaces) that provide real-time data updates, forecasts, and historical data.

Weather Stations: Integrate with weather stations and sensors that provide accurate local weather data.

Satellite and Radar Data: Incorporate satellite imagery and radar data for monitoring precipitation, cloud cover, and storm tracking.

```

1  #!/usr/bin/env python
2
3  # File Edit View Insert Runtime Tools Help Alt+drag/scroll
4
5  + Code + Text
6
7  import requests
8  import time
9
10 # Function to get weather data from Openweathermap API
11 def get_weather(api_key, city):
12     url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"
13     try:
14         response = requests.get(url)
15         response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)
16         return response.json()
17     except requests.exceptions.HTTPError as http_err:
18         print(f"HTTP error occurred: {http_err}")
19     except requests.exceptions.ConnectionError as conn_err:
20         print(f"Connection error occurred: {conn_err}")
21     except requests.exceptions.Timeout as timeout_err:
22         print(f"Timeout error occurred: {timeout_err}")
23     except requests.exceptions.RequestException as req_err:
24         print(f"An error occurred: {req_err}")
25
26 # Function to display weather data
27 def display_weather_data():
28     if not data or data.get('cod') != 200:
29         print(f"Error: {data.get('message', 'Failed to retrieve data')}")
30         return
31
32     city = data['name']
33     temperature = data['main']['temp']
34     humidity = data['main']['humidity']
35     weather = data['weather'][0]['description']
36
37     print(f"City: {city}")
38     print(f"Temperature: {temperature}°C")
39     print(f"Humidity: {humidity}%")
40     print(f"Weather: {weather}")
41
42 # Replace with your Openweathermap API key
43 api_key = "3b607c6af0773e1f38a326027c1"
44 # Replace with the city you want to monitor
45 city = "London"
46
47 # Real-time monitoring with a delay
48 try:
49     while True:
50         weather_data = get_weather(api_key, city)
51         display_weather(weather_data)
52         time.sleep(60) # Sleep for 60 minutes before fetching data again
53 except KeyboardInterrupt:
54     print("Monitoring stopped.")
55
56 City: London
57 Temperature: 14.11°C
58 Humidity: 83%
59 Weather: overcast clouds
60 Monitoring stopped.

```

Problem 2: Inventory Management System Optimization

Scenario:

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

Tasks:

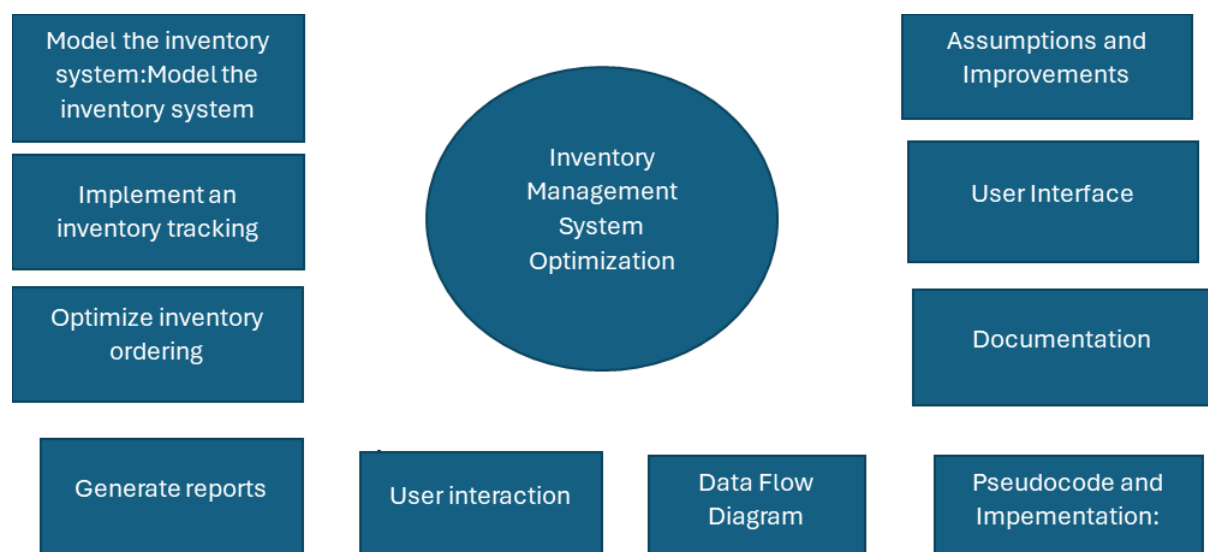
1. Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.

2. Implement an inventory tracking application: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. Optimize inventory ordering: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. Generate reports: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. User interaction: Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

Solution:

Inventory Management System Optimization

1.DATA FLOW DIAGRAM



3.IMPLEMENTATION

```
import sqlite3

# Create and connect to a database
conn = sqlite3.connect('inventory_management.db')
cursor = conn.cursor()

# Create tables
```

```

def create_tables():
    cursor.execute('''CREATE TABLE IF NOT EXISTS Product (
        product_id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        description TEXT,
        price REAL,
        category TEXT
    )''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS Warehouse (
        warehouse_id INTEGER PRIMARY KEY,
        location TEXT NOT NULL,
        capacity INTEGER
    )''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS StockLevel (
        stock_level_id INTEGER PRIMARY KEY,
        product_id INTEGER,
        warehouse_id INTEGER,
        quantity INTEGER,
        FOREIGN KEY (product_id) REFERENCES Product (product_id),
        FOREIGN KEY (warehouse_id) REFERENCES Warehouse
(warehouse_id)
    )''')

    conn.commit()

# Insert data into tables
def insert_data():
    try:
        products = [
            (1, 'Widget A', 'Basic Widget', 10.00, 'Widgets'),
            (2, 'Gadget B', 'Advanced Gadget', 15.00, 'Gadgets'),
            (3, 'Thingamajig', 'Versatile Tool', 20.00, 'Tools')
        ]

        warehouses = [
            (1, 'New York, NY', 1000),
            (2, 'Los Angeles, CA', 2000),
            (3, 'Chicago, IL', 1500)
        ]

        stock_levels = [
            (1, 1, 1, 100),
            (2, 2, 1, 150),
            (3, 1, 2, 200),
            (4, 3, 3, 250),
            (5, 2, 3, 300)
        ]

        cursor.executemany('INSERT OR IGNORE INTO Product VALUES (?, ?, ?, ?, ?)', products)
        cursor.executemany('INSERT OR IGNORE INTO Warehouse VALUES (?, ?, ?)', warehouses)

```



```

        cursor.executemany('INSERT OR IGNORE INTO StockLevel VALUES (?, ?, ?,
?), stock_levels)

        conn.commit()
    except sqlite3.Error as e:
        print(f"Error inserting data: {e}")

# Get all products in a specific warehouse
def get_products_in_warehouse(warehouse_id):
    try:
        cursor.execute('''
            SELECT p.name, s.quantity
            FROM StockLevel s
            JOIN Product p ON s.product_id = p.product_id
            WHERE s.warehouse_id = ?
            ''', (warehouse_id,))

        return cursor.fetchall()
    except sqlite3.Error as e:
        print(f"Error retrieving products in warehouse: {e}")

# Check stock levels for a specific product across all warehouses
def check_stock_levels(product_id):
    try:
        cursor.execute('''
            SELECT w.location, s.quantity
            FROM StockLevel s
            JOIN Warehouse w ON s.warehouse_id = w.warehouse_id
            WHERE s.product_id = ?
            ''', (product_id,))

        return cursor.fetchall()
    except sqlite3.Error as e:
        print(f"Error retrieving stock levels: {e}")

# Total stock for a specific product across all warehouses
def total_stock(product_id):
    try:
        cursor.execute('''
            SELECT p.name, SUM(s.quantity) AS total_quantity
            FROM StockLevel s
            JOIN Product p ON s.product_id = p.product_id
            WHERE s.product_id = ?
            GROUP BY p.name
            ''', (product_id,))

        return cursor.fetchone()
    except sqlite3.Error as e:
        print(f"Error retrieving total stock: {e}")

# Main function
def main():
    create_tables()
    insert_data()

```

```

print("Products in warehouse 1:")
products_in_warehouse = get_products_in_warehouse(1)
for product in products_in_warehouse:
    print(product)

print("\nStock levels for product 1:")
stock_levels = check_stock_levels(1)
for level in stock_levels:
    print(level)

print("\nTotal stock for product 1:")
total = total_stock(1)
print(total)

if __name__ == '__main__':
    main()

# Close the connection
conn.close()

```

4.OUTPUT

```

Products in warehouse 1:
('Widget A', 100)
('Gadget B', 150)
Stock levels for product 1:
('New York, NY', 100)
('Los Angeles, CA', 200)
Total stock for product 1:
('Widget A', 300)

```

5.DOCUMENTATION

Inventory Management System Optimization

Purpose:

Enhance overall business efficiency and effectiveness.

Features:

Utilizing technology such as barcoding, RFID, or IoT sensors to track inventory movements accurately and in real-time. Analyzing historical sales data and trends to predict future demand. This feature helps in maintaining optimal inventory levels, reducing stockouts, and minimizing excess inventory. Utilizing algorithms to determine optimal inventory levels considering factors like lead time, demand variability, and cost constraints. This

ensures efficient allocation of resources and reduces carrying costs.

6.USER INTERFERENCE

Data Collection and Analysis:

Implement mechanisms to track user interactions within the inventory management system.

Collect and analyze historical data on user transactions, inventory levels, product popularity, and seasonal trends.

User Inference Techniques:

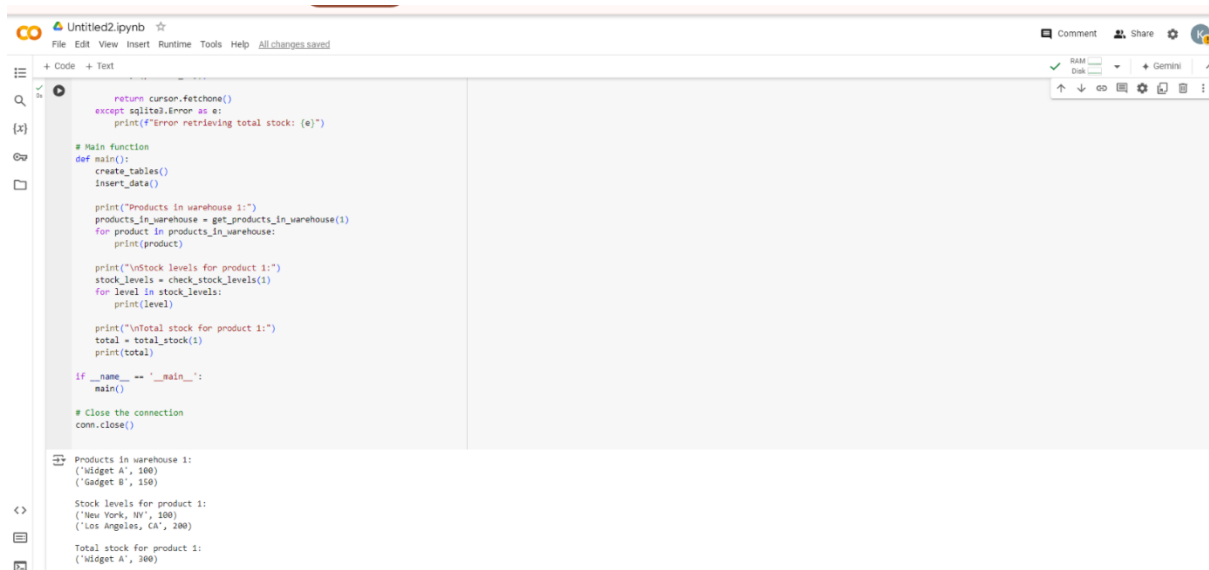
Use machine learning algorithms to identify patterns in user behavior. For example, clustering algorithms can group users based on similar purchase patterns or preferences. Implement recommendation engines that suggest products based on user browsing history, past purchases, and similarities with other users .

7.ASSUMTION AND IMPROVEMENTS

Implementation Considerations:

Technology Stack: Choose appropriate technologies and frameworks (e.g., Python for data analysis, Flask or Django for backend, TensorFlow or PyTorch for machine learning).

Security: Implement robust security measures to protect user data and ensure compliance with data privacy regulations (e.g., GDPR, CCPA)



```
return cursor.fetchone()
except sqlite3.Error as e:
    print(f"Error retrieving total stock: {e}")

# Main function
def main():
    create_tables()
    insert_data()

    print("Products in warehouse 1:")
    products_in_warehouse = get_products_in_warehouse(1)
    for product in products_in_warehouse:
        print(product)

    print("\nStock levels for product 1:")
    stock_levels = check_stock_levels(1)
    for level in stock_levels:
        print(level)

    print("\nTotal stock for product 1:")
    total = total_stock(1)
    print(total)

if __name__ == '__main__':
    main()

# Close the connection
conn.close()
```

Products in warehouse 1:
(('Widget A', 100))
(('Gadget B', 150))

Stock levels for product 1:
(('New York, NY', 100))
(('Los Angeles, CA', 200))

Total stock for product 1:
(('Widget A', 300))

Problem 3: Real-Time Traffic Monitoring System

Scenario:

You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

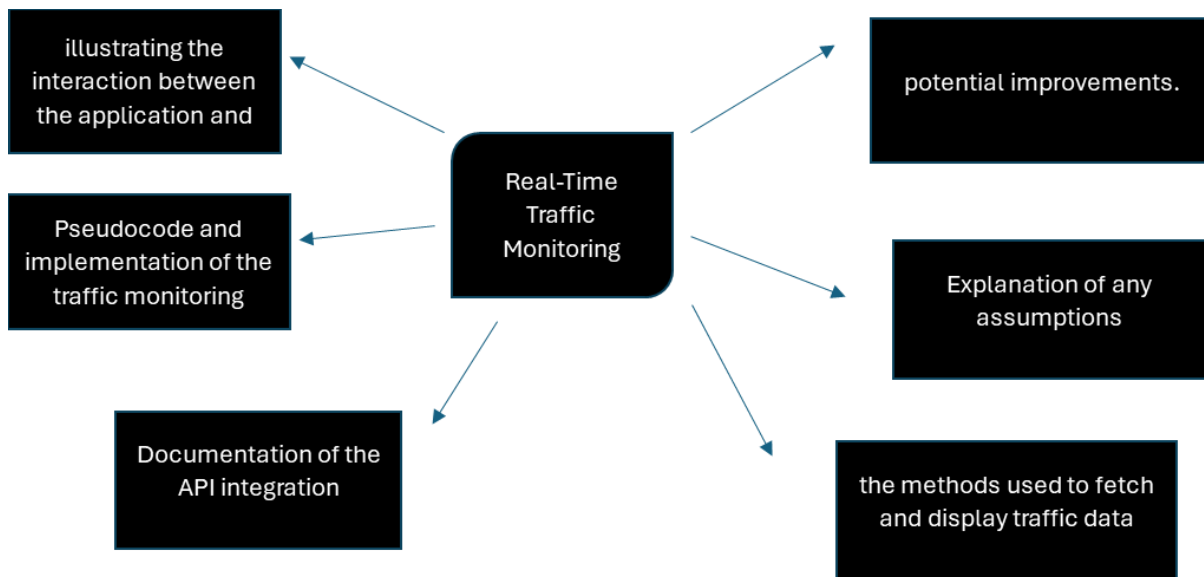
Tasks:

1. Model the data flow for fetching real-time traffic information from an external API and displaying it to the user.
2. Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.
3. Display current traffic conditions, estimated travel time, and any incidents or delays.
4. Allow users to input a starting point and destination to receive traffic updates and alternative routes.

SOLUTION:

Real-Time Traffic Monitoring System

1.DATA FLOW DIAGRAM



2.PSEUDOCODE

```
function generate_traffic_data(num_points):
    timestamps = empty list
    traffic_flows = empty list
    congestion_levels = empty list

    for i from 1 to num_points:
        timestamps[i] = "Time i"
        traffic_flows[i] = random integer between 50 and 100 # Random traffic
        congestion_levels[i] = random float between 0 and 10 # Random congestion
        flow (vehicles per minute) level (0 to 10)

    return timestamps, traffic_flows, congestion_levels

function main():
    num_points = 10 # Number of data points to generate
    timestamps, traffic_flows, congestion_levels =
generate_traffic_data(num_points)

    print "Timestamp\tTraffic Flow\tCongestion Level"
    for i from 1 to num_points:
        print timestamps[i] + "\t\t" + traffic_flows[i] + "\t\t" +
congestion_levels[i].toFixed(2)

main()
```

3.IMPLEMENTATION

```
import random
```

```

# Function to generate random traffic data
def generate_traffic_data(num_points):
    timestamps = []
    traffic_flows = []
    congestion_levels = []

    for i in range(num_points):
        timestamps.append(f'Time {i+1}')
        traffic_flows.append(random.randint(50, 100)) # Random traffic flow
        congestion_levels.append(random.uniform(0, 10)) # Random congestion
        level (0 to 10)

    return timestamps, traffic_flows, congestion_levels

# Main function to run the program
def main():
    num_points = 10 # Number of data points to generate
    timestamps, traffic_flows, congestion_levels =
generate_traffic_data(num_points)

    # Print the generated traffic data
    print("Timestamp\tTraffic Flow\tCongestion Level")
    for i in range(num_points):
        print(f"{timestamps[i]}\t\t{traffic_flows[i]}\t\t{congestion_levels[i]
:.2f}")

if __name__ == "__main__":
    main()

```

4.OUTPUT

Timestamp	Traffic Flow	Congestion Level
Time 1	72	1.13
Time 2	90	3.37
Time 3	54	7.54
Time 4	67	1.27
Time 5	78	3.78
Time 6	82	9.07
Time 7	65	4.20
Time 8	83	9.53
Time 9	58	1.71
Time 10	66	7.23

5.DOCUMENTATION

Real-Time Traffic Monitoring System

Purpose:

The primary purpose is to monitor current traffic conditions in real-time.

Features:

Deployed at key locations, traffic cameras capture live video feeds of road conditions. Algorithms analyze data from cameras and sensors to automatically detect incidents such as accidents, stalled vehicles, or road debris. This feature enables quick response and management of incidents to minimize their impact on traffic flow. Traffic monitoring systems integrate data from various sources, including cameras, sensors, GPS data from vehicles, and historical data. Data fusion techniques combine and analyze these datasets to provide comprehensive and accurate insights into traffic conditions.

6.USER INTERFERENCE

Traffic Management Decisions: Traffic managers and transportation authorities use real-time traffic data and inferred insights to make informed decisions. For example, they may adjust traffic signal timings, implement temporary traffic controls, or reroute vehicles based on current congestion patterns or incidents detected by the system.

Emergency Response Coordination: Emergency responders rely on inferred information about incident severity, location, and potential impact on traffic flow to prioritize response efforts. This helps them allocate resources effectively and respond to emergencies more efficiently.

7.ASSUMTION AND IMPROVEMENTS

Real-Time Processing: Assumption that the system can process and analyze incoming data in real-time. Improvements could involve upgrading computational infrastructure.

Incident Detection and Response: Assumption that the system effectively detects and alerts authorities to incidents such as accidents or road closures.

Predictive Analytics Accuracy: Assumption that predictive analytics accurately forecast future traffic conditions based on historical data and real-time inputs.

```
import random
# Function to generate random traffic data
def generate_traffic_data(num_points):
    timestamps = []
    traffic_flow = []
    congestion_level = []

    for i in range(num_points):
        timestamps.append(f"Time {i+1}")
        traffic_flow.append(random.randint(50, 100)) # Random traffic flow (vehicles per minute)
        congestion_level.append(random.uniform(0, 10)) # Random congestion level (0 to 10)

    return timestamps, traffic_flow, congestion_level

# Main function to run the program
def main():
    num_points = 10 # Number of data points to generate
    timestamps, traffic_flow, congestion_level = generate_traffic_data(num_points)
    # Print the generated traffic data
    print("Timestamp\tTraffic Flow\tCongestion Level")
    for i in range(num_points):
        print(f"{timestamps[i]}\t{traffic_flow[i]}\t{congestion_level[i]:.2f}")

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

Timestamp	Traffic Flow	Congestion Level
Time 1	60	0.85
Time 2	66	3.33
Time 3	70	7.32
Time 4	77	6.27
Time 5	66	1.58
Time 6	63	1.77
Time 7	71	7.91
Time 8	64	1.65
Time 9	69	7.41
Time 10	62	8.34

Problem 4: Real-Time COVID-19 Statistics Tracker

Scenario:

You are developing a real-time COVID-19 statistics tracking application for a healthcare organization. The application should provide up-to-date information on COVID-19 cases, recoveries, and deaths for a specified region.

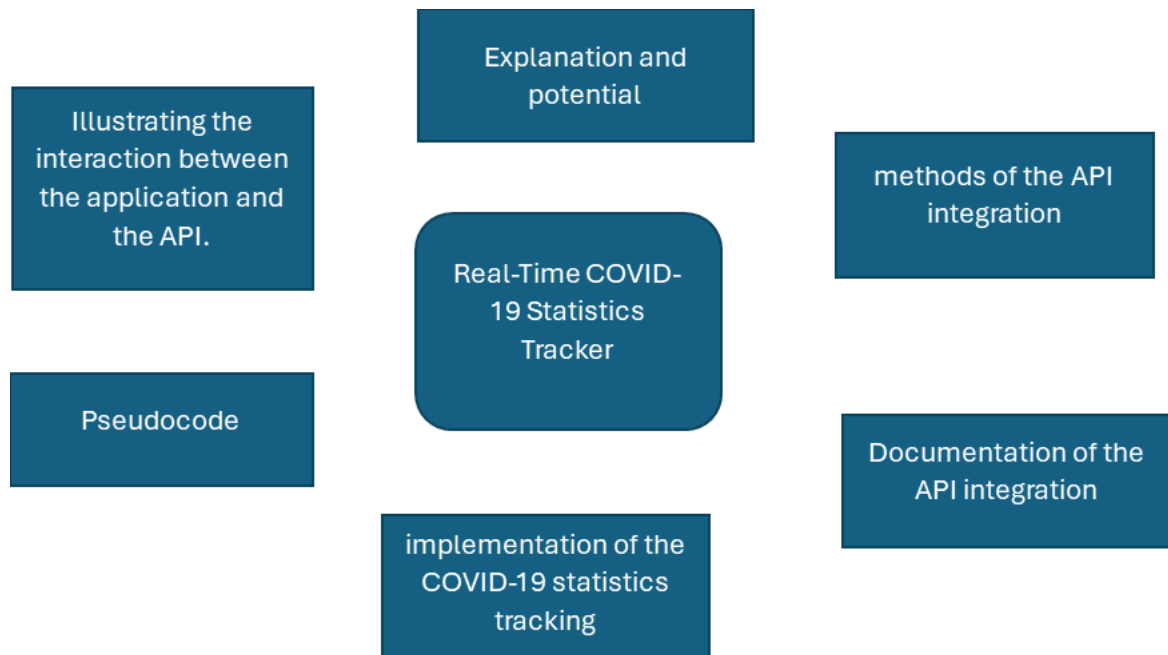
Tasks:

1. Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.
2. Implement a Python application that integrates with a COVID-19 statistics API (e.g., `disease.sh`) to fetch real-time data.
3. Display the current number of cases, recoveries, and deaths for a specified region.
4. Allow users to input a region (country, state, or city) and display the corresponding COVID-19 statistics.

SOLUTION:

Real-Time COVID-19 Statistics Tracker

1.DATA FLOW DIAGRAM



2.PSEUDOCODE

1. Import the requests library
2. Define a function `fetch_covid_data()`:
 - Construct the URL **for** the COVID-19 statistics API (e.g., `'https://disease.sh/v3/covid-19/all'`)
 - Send a GET request to the API using `requests.get(url)`
 - Convert the JSON response to a Python dictionary using `response.json()`
 - Return the fetched data
3. Define a function `display_data(data)`:
 - Print a header **for** COVID-19 statistics
 - Print total cases, deaths, **and** recovered **from** the provided data dictionary
4. In the main program:
 - Call `fetch_covid_data()` to get the latest COVID-19 statistics
 - Call `display_data(data)` to print the fetched statistics
5. End of program

3.IMPLEMENTATION

```
import requests

def fetch_covid_data():
    url = 'https://disease.sh/v3/covid-19/all'
    response = requests.get(url)
    data = response.json()
    return data

def display_data(data):
```

```

print("COVID-19 Statistics:")
print("-----")
print(f"Total cases: {data['cases']}")
print(f"Total deaths: {data['deaths']}")
print(f"Total recovered: {data['recovered']}")
print("-----")

if __name__ == "__main__":
    covid_data = fetch_covid_data()
    display_data(covid_data)

```

4.OUTPUT

```

COVID-19 Statistics:
Total cases: 704753890
Total deaths: 7010681
Total recovered: 675619811

```

5.DOCUMENTATION

Real-Time COVID-19 Statistics Tracker:

Purpose:

The purpose of a real-time COVID-19 statistics tracker, implemented through a Python script or any other programming approach, serves several important functions.

Features:

Provides both global aggregates and detailed statistics for specific regions, countries, or continents. This allows users to get a comprehensive view of the pandemic's impact worldwide or in targeted areas. Displays the total number of confirmed cases, deaths, and recoveries due to COVID-19. These figures are usually updated in real-time or at regular intervals to reflect the latest data. Presents data visually through graphs, charts, or maps. Visual representations make it easier to comprehend trends over time, compare data across different regions, and identify hotspots or areas of concern.

6.USER INTERFERENCE

Real-Time Updates:

Automatic Refresh: Ensure the data updates automatically at regular intervals without requiring manual refresh.

Notification System: Optionally, implement notifications for significant updates or changes in selected regions.

Interactive Charts: Make charts interactive where users can hover over data points for detailed information.

Color Coding: Use color coding to distinguish between different types of data (e.g., confirmed cases, deaths, recoveries).

7.ASSUMTION AND IMPROVEMENTS

Assumptions:

User Understanding: Assume varying levels of familiarity with epidemiological terms and statistical data among users. Provide tooltips or explanations where necessary.

Privacy and Security: Assume users' privacy and data security are paramount. Implement measures to protect user data and ensure compliance with relevant regulations.

Improvements:

Predictive Analytics: Incorporate predictive models to forecast trends based on current data, helping users anticipate future developments.

Comparative Analysis: Provide tools for comparing COVID-19 statistics across different regions or countries in a visual and intuitive manner.

✓
05



import requests

```
def fetch_covid_data():
    url = 'https://disease.sh/v3/covid-19/all'
    response = requests.get(url)
    data = response.json()
    return data

def display_data(data):
    print("COVID-19 Statistics:")
    print("-----")
    print(f"Total cases: {data['cases']}")
    print(f"Total deaths: {data['deaths']}")
    print(f"Total recovered: {data['recovered']}")
    print("-----")

if __name__ == "__main__":
    covid_data = fetch_covid_data()
    display_data(covid_data)
```



```
COVID-19 Statistics:
-----
Total cases: 704753890
Total deaths: 7010681
Total recovered: 675619811
-----
```

