

Assignment: python programming for DLP

Name : M.Vaishnavi

Register Number : 192372203

Department : CSE(AI)

Date of submission :

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:

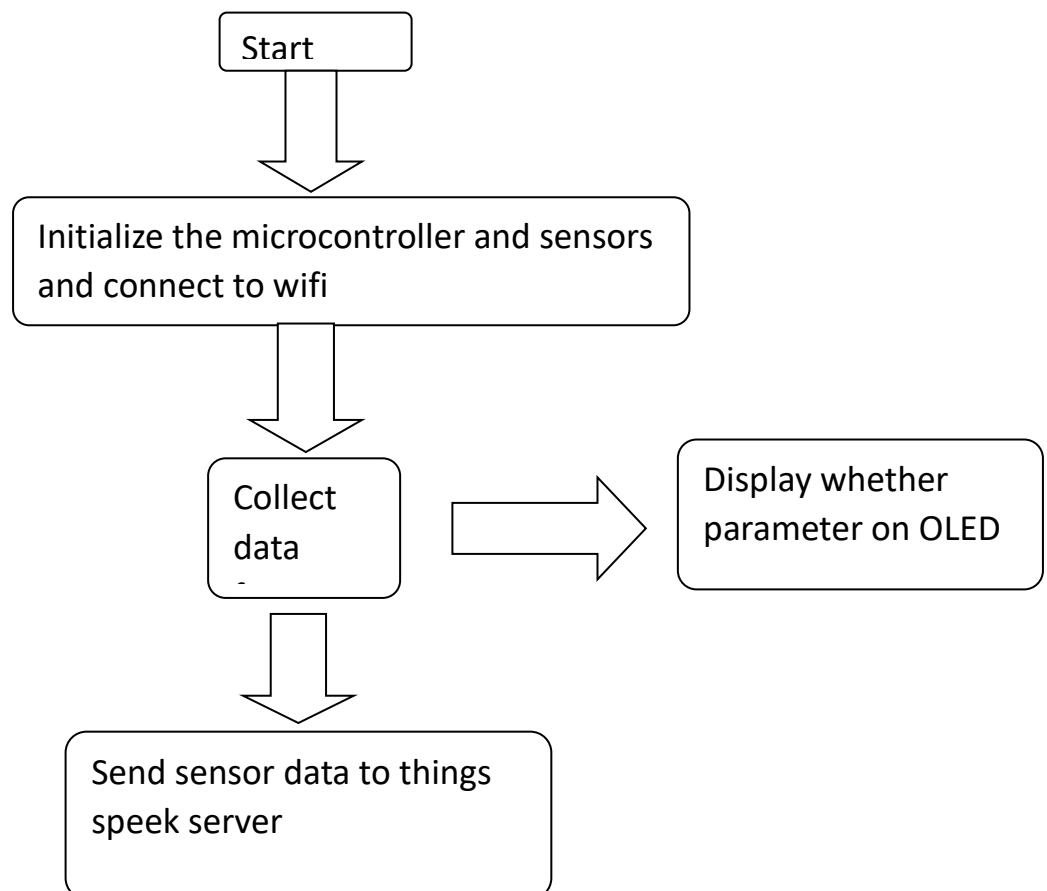
1. Model the data flow for fetching weather information from an external API and displaying it to the user.
2. Implement a Python application that integrates with a weather API (e.g., OpenWeatherMap) to fetch real-time weather data.
3. Display the current weather information, including temperature, weather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

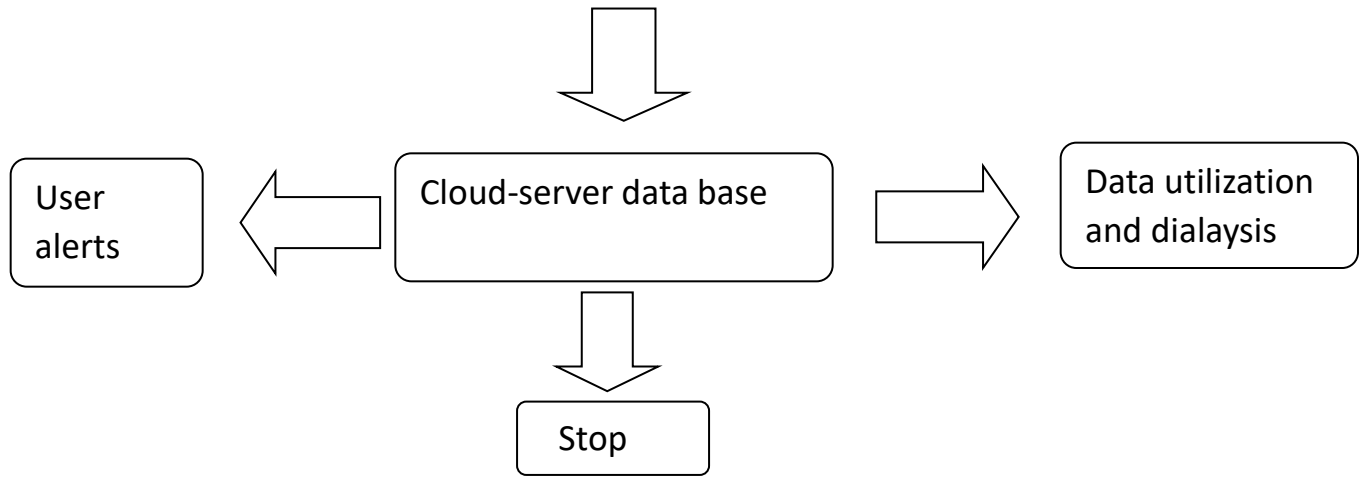
4. Allow users to input the location (city name or coordinates) and display the corresponding weather data.

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the weather monitoring system.
- Documentation of the API integration and the methods used to fetch and display weather data.
- Explanation of any assumptions made and potential improvements.

FLOW CHART





IMPLIMENTATION:

```
import requests
```

```
def fetch_weather_data(api_key, location):
```

```
    base_url =
```

```
    "https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid"
```

```
    params = {
```

```
        'q': location,
```

```
        'appid': api_key,
```

```
        'units': 'metric'
```

```
    }
```

```
    try:
```

```
        response = requests.get(base_url, params=params)
```

```
        data = response.json()
```

```
        if data["cod"] == 200:
```

```
            weather_info = {
```

```
                'location': data['name'],
```

```
                'temperature': data['main']['temp'],
```

```
                'weather': data['weather'][0]['description'],
```

```
                'humidity': data['main']['humidity'],
```

```

        'wind_speed': data['wind']['speed']
    }

    return weather_info

else:

    return None

except Exception as e:

    print(f"Error fetching weather data: {e}")

    return None

def display_weather(weather_info, location):

    if weather_info:

        print(f"Weather in {location}:")

        print(f"Temperature: {weather_info['temperature']} °C")

        print(f"Weather: {weather_info['weather']}")

        print(f"Humidity: {weather_info['humidity']}%")

        print(f"Wind Speed: {weather_info['wind_speed']} m/s")

    else:

        print(f"Failed to fetch weather data for {location}")

def main():

    api_key = "ed7c18d0f1024da78bf89f147ccd9bca"

    location = input("Enter city name or coordinates (latitude,longitude): ")

    weather_info = fetch_weather_data(api_key, location)

    display_weather(weather_info, location)

if __name__ == "__main__":

    main()

```

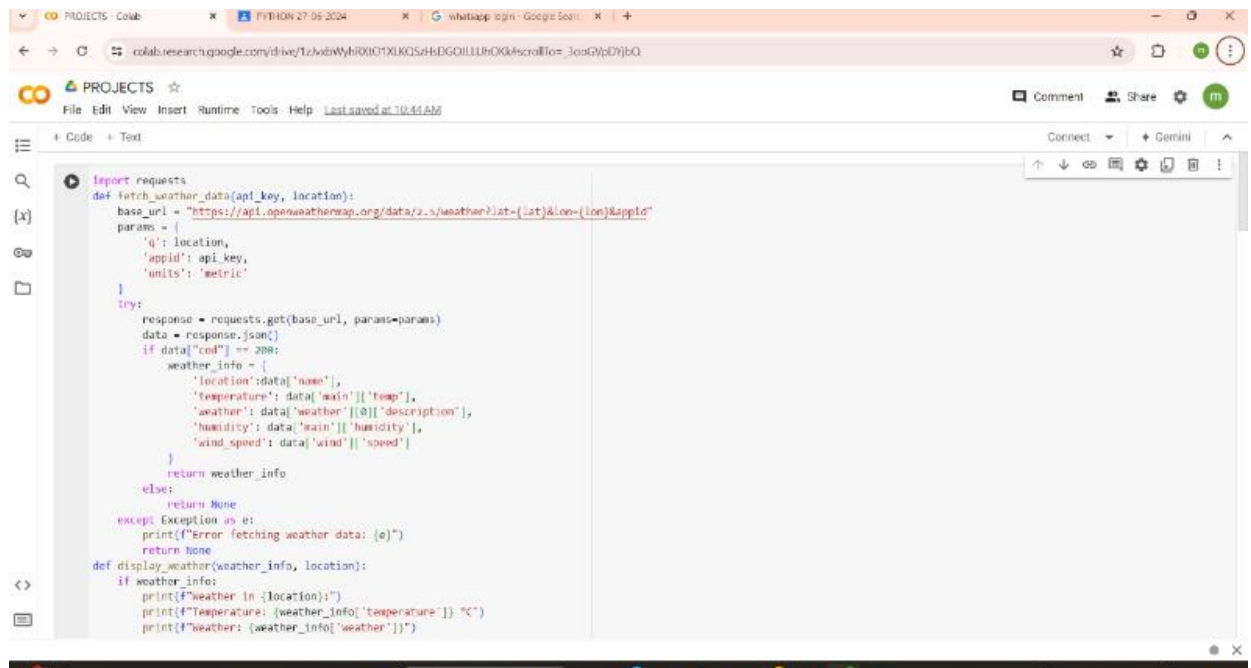
INPUT:

Enter city name or coordinates (latitude,longitude): CHENNAI

OUTPUT:

Weather in CHENNAI:

- * Temperature: 29.24 °C
- * Weather: broken clouds
- * Humidity: 78%
- * Wind Speed: 5.14 m/s



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'PROJECTS - Colab', 'PYTHON 27.05.2024', 'whatsapp login - Google Search', and a new tab. The address bar shows a Google Drive link. The notebook's toolbar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', 'Comment', 'Share', and a 'Last saved at 10:44 AM' indicator. The code editor displays a Python script that uses the 'requests' library to fetch weather data from the OpenWeatherMap API. The script defines a function 'fetch_weather_data' that takes an API key and a location as input. It constructs a URL with the location's latitude and longitude, and a parameters dictionary with the location, API key, and units. It then makes a GET request and processes the JSON response to extract temperature, weather description, humidity, and wind speed. A second function, 'display_weather', is defined to print the extracted data in a formatted manner. The script also includes error handling for network issues.

```
import requests
def fetch_weather_data(api_key, location):
    base_url = "https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={api_key}"
    params = {
        'q': location,
        'appid': api_key,
        'units': 'metric'
    }
    try:
        response = requests.get(base_url, params=params)
        data = response.json()
        if data["cod"] == 200:
            weather_info = {
                'location': data['name'],
                'temperature': data['main']['temp'],
                'weather': data['weather'][0]['description'],
                'humidity': data['main']['humidity'],
                'wind_speed': data['wind']['speed']
            }
            return weather_info
        else:
            return None
    except Exception as e:
        print(f"Error fetching weather data: {e}")
        return None
def display_weather(weather_info, location):
    if weather_info:
        print(f"Weather in {location}:")
        print(f"Temperature: {weather_info['temperature']} °C")
        print(f"Weather: {weather_info['weather']}")
```

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 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: Allowing inventory turnover and profitability.

Tasks:

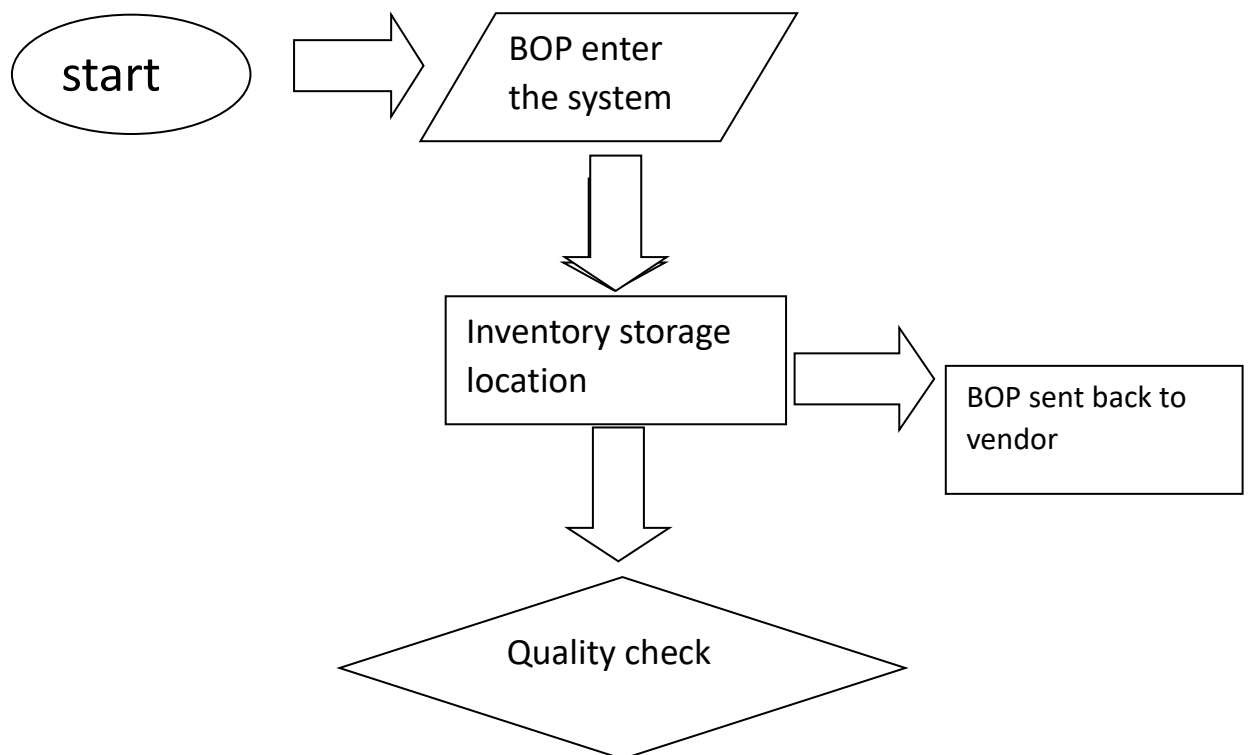
1. Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. Implement an inventory tracking application: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. Optimize inventory ordering: Implement algorithms to

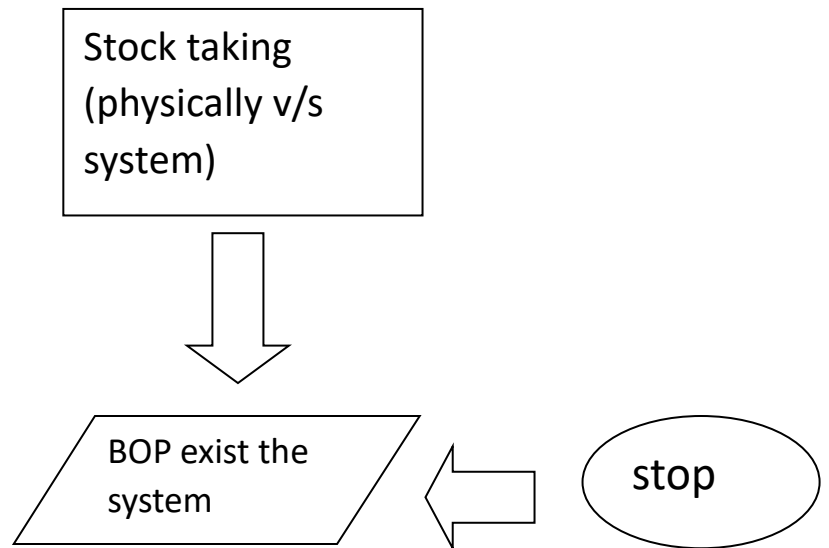
calculate users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

Deliverables:

- Data Flow Diagram: Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).
- Pseudocode and Implementation: Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.
- Documentation: Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).
- User Interface: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.
- Assumptions and Improvements: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

FLOW CHART





IMPLEMENTATION:

class Product:

```
def __init__(self, id, name, category, price, supplier):
```

```
    self.id = id
```

```
    self.name = name
```

```
    self.category = category
```

```
    self.price = price
```

```
    self.supplier = supplier
```

```
    self.stock_level = 0
```

class Product:

```
def __init__(self, id, name, category, price, supplier):
```

```
    self.id = id
```

```
    self.name = name
```

```
    self.category = category
```



```
self.price = price
```

```
self.supplier = supplier
```

```
self.stock_level = 0
```

```
def update_stock(self, quantity):
```

```
    self.stock_level += quantity
```

```
class InventoryManagementSystem:
```

```
    def __init__(self):
```

```
        self.products = {}
```

```
    def add_product(self, product):
```

```
        self.products[product.id] = product
```

```
    def track_stock_level(self, product_id):
```

```
        return self.products.get(product_id, None).stock_level if product_id in self.products  
    else None
```

```
    def alert_low_stock(self, product_id, threshold):
```

```
        if product_id in self.products:
```

```
            if self.products[product_id].stock_level < threshold:
```

```
                print("Low stock alert!") # Added a placeholder action
```

```
# Example usage
```

```
if __name__ == "__main__":
```

```
    inventory_system = InventoryManagementSystem()
```

```
# Adding products
```

```
product1 = Product(1, "Laptop", "Electronics", 1200, "Supplier A")
```

```
product2 = Product(2, "Printer", "Office Supplies", 300, "Supplier B")
```

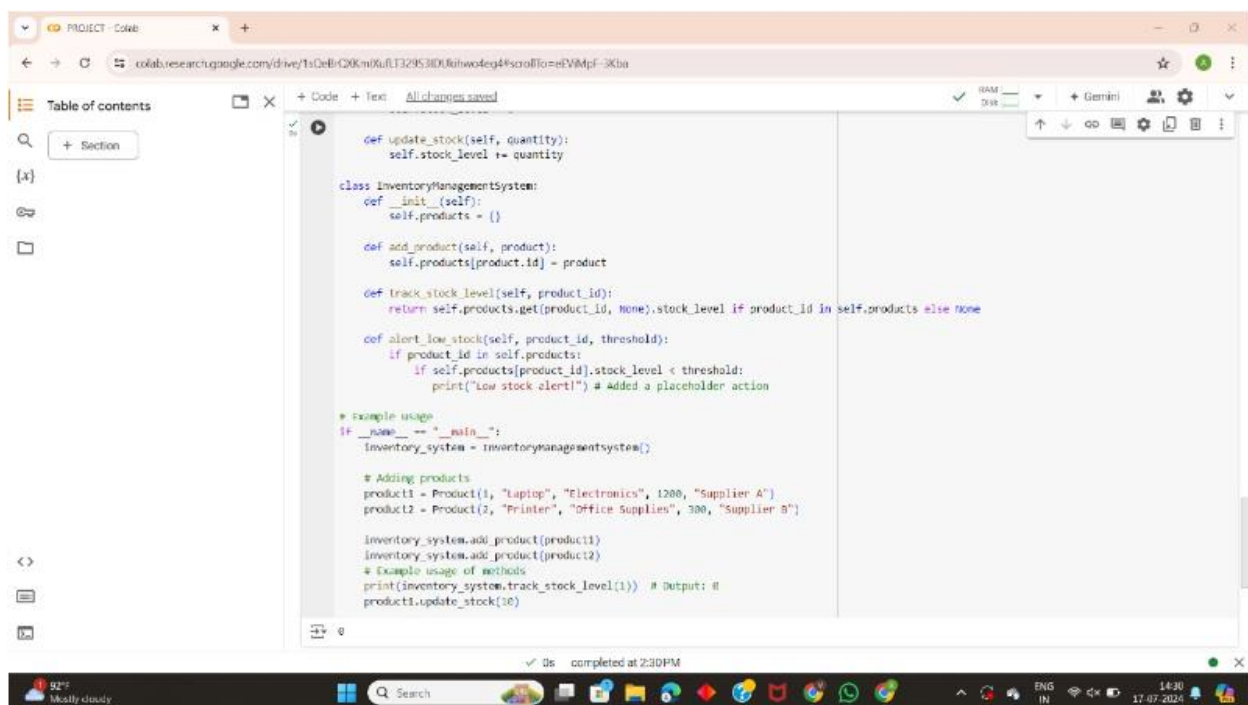
```
inventory_system.add_product(product1)
```

```
inventory_system.add_product(product2)
```

```
# Example usage of methods
```

```
print(inventory_system.track_stock_level(1)) # Output: 0
```

```
product1.update_stock(10)
```



The screenshot shows a Google Colab notebook with a table of contents on the left and a code editor on the right. The code defines an `InventoryManagementSystem` class with methods for adding products, tracking stock levels, and alerting low stock. It also includes example usage code that creates an instance of the class, adds two products, and prints the stock level of the first product.

```
def update_stock(self, quantity):
    self.stock_level += quantity

class InventoryManagementSystem:
    def __init__(self):
        self.products = {}

    def add_product(self, product):
        self.products[product.id] = product

    def track_stock_level(self, product_id):
        return self.products.get(product_id, None).stock_level if product_id in self.products else None

    def alert_low_stock(self, product_id, threshold):
        if product_id in self.products:
            if self.products[product_id].stock_level < threshold:
                print("Low stock alert!") # Added a placeholder action

# Example usage
if __name__ == "__main__":
    inventory_system = InventoryManagementSystem()

    # Adding products
    product1 = Product(1, "Laptop", "Electronics", 1200, "Supplier A")
    product2 = Product(2, "Printer", "Office Supplies", 300, "Supplier B")

    inventory_system.add_product(product1)
    inventory_system.add_product(product2)

    # Example usage of methods
    print(inventory_system.track_stock_level(1)) # Output: 0
    product1.update_stock(10)
```

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.

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the traffic monitoring system.
- Documentation of the API integration and the methods used to fetch and display traffic

data.

- Explanation of any assumptions made and potential improvements.

IMPLEMENTATION:

```
def display_traffic_data(traffic_data):  
    if traffic_data:  
        try:  
            # Attempt to extract relevant information from traffic_data  
            routes = traffic_data.get('routes', [])  
            if routes:  
                legs = routes[0].get('legs', [])  
                if legs:  
                    duration = legs[0]['duration_in_traffic']['text']  
                    incidents = legs[0].get('traffic_speed_entry', [])  
  
                    print(f"Estimated travel time: {duration}")  
                    if incidents:  
                        print("Incidents or delays:")  
                        for incident in incidents:  
                            print(f"- {incident['incident_description']}")  
                    else:  
                        print("No incidents or delays reported.")  
                else:  
                    print("No legs found in the route.")  
            else:  
                print("No routes found.")  
        except KeyError as e:  
            print(f"KeyError: {e}. Incorrect data structure in API response.")
```

else:

```
print("No traffic data available.")
```

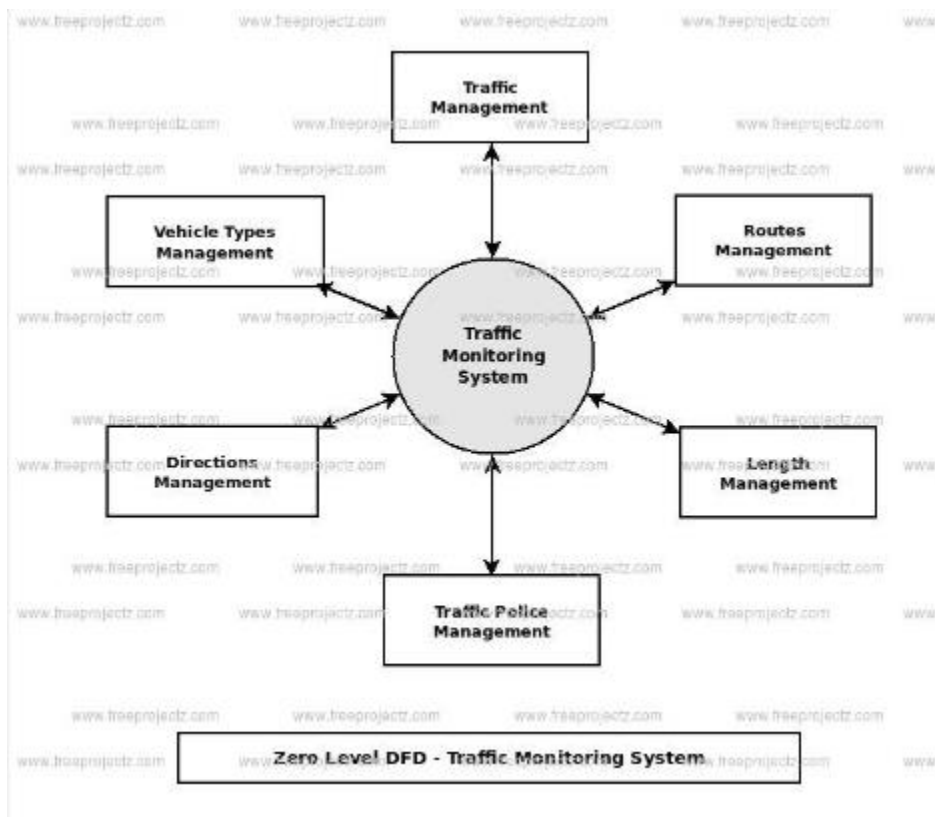
```
def display_traffic_data(traffic_data):
    if traffic_data:
        try:
            # Attempt to extract relevant information from traffic_data
            routes = traffic_data.get('routes', [])
            if routes:
                legs = routes[0].get('legs', [])
                if legs:
                    duration = legs[0]['duration_in_traffic']['text']
                    incidents = legs[0].get('traffic_speed_entry', [])

                    print(f"Estimated travel time: {duration}")
                    if incidents:
                        print("Incidents or delays:")
                        for incident in incidents:
                            print(f"- {incident['incident_description']}")
                    else:
                        print("No incidents or delays reported.")
                else:
                    print("No legs found in the route.")
            else:
                print("No routes found.")
        except KeyError as e:
            print(f"KeyError: {e}. Incorrect data structure in API response.")
    else:
        print("No traffic data available.")
```

```
[ ] class Product:
    def __init__(self, id, name, category, price, supplier):
        self.id = id
        self.name = name
```

✓ 0s completed at 10:20 PM

Windows taskbar with search bar and icons for File Explorer, Mail, Task View, Google Chrome, Spotify, Word, and a custom icon.



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.

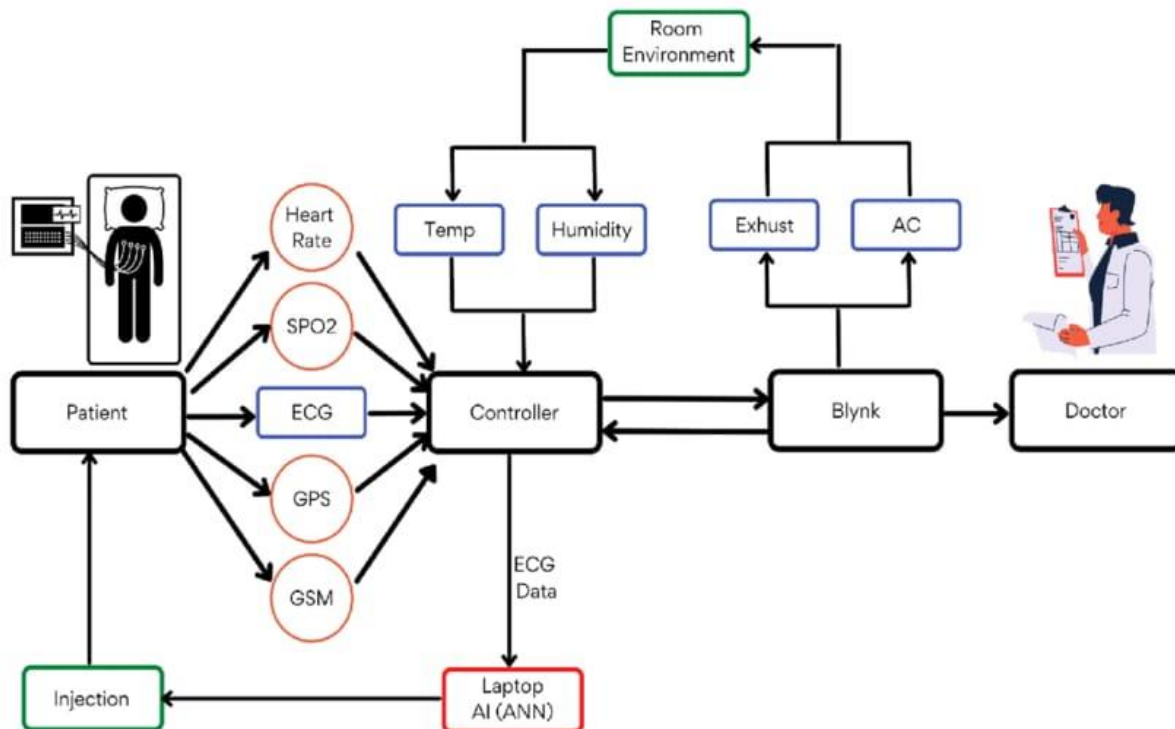
Tasks:

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.

Deliverables:

- Data flow diagram illustrating the interaction between the application and the API.
- Pseudocode and implementation of the COVID-19 statistics tracking application.
- Documentation of the API integration and the methods used to fetch and display COVID 19 data.
- Explanation of any assumptions made and potential improvements.

FLOW CHART:



IMPLEMENTATION:

```
import requests
```

```
def fetch_covid_statistics(region):
```

```
    endpoint = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"
```

```
    response = requests.get(endpoint)
```

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

```
        return response.json()
```

```
    else:
```

```
        return None
```

```
def display_statistics(statistics):
```

```
    if statistics is not None:
```

```
        print("COVID-19 Statistics:")
```

```
        print(f"Cases: {statistics['cases']}")
```



```
        print(f"Recoveries: {statistics['recovered']}")

        print(f"Deaths: {statistics['deaths']}")

    else:

        print("Failed to fetch COVID-19 statistics.")

def main():

    region = input("Enter a country, state, or city: ")

    statistics = fetch_covid_statistics(region)

    display_statistics(statistics)

if __name__ == "__main__":

    main()
```

INPUT:

Enter a region(eg,world,USA,Germany): Hungary

OUTPUT:

COVID-19 Statistics for hungary:

Cases: 2230232

Recovered: 2152155

Deaths: 49048

PROJECTS · Colab

PYTHON 27.06.2024

whatsapp login - Google Search

+

colab.research.google.com/drive/1xvdiWYhR00CTXKQ5shhDGOILLHOKGkscraTic-3osGvpD7yjbD

PROJECTS

File Edit View Insert Runtime Tools Help Last saved at 10:44 AM

+ Code + Text

Connect + Gemini

import requests

def fetch_covid_statistics(region):

endpoint = "https://disease.sh/v3/covid-19/historical/all?lastdays=all"

response = requests.get(endpoint)

if response.status_code == 200:

return response.json()

else:

return None

def display_statistics(statistics):

if statistics is not None:

print("COVID-19 Statistics:")

print(f"Cases: {statistics['cases']}")

print(f"Recoveries: {statistics['recovered']}")

print(f"Deaths: {statistics['deaths']}")

else:

print("Failed to fetch COVID-19 statistics.")

def main():

region = input("Enter a country, state, or city: ")

statistics = fetch_covid_statistics(region)

display_statistics(statistics)

if __name__ == "__main__":

main()

<>

Enter a country, state, or city: china

COVID-19 Statistics:

Cases: {'1/22/20': 557, '1/23/20': 637, '1/24/20': 844, '1/25/20': 1437, '1/26/20': 2120, '1/27/20': 2929, '1/28/20': 5580, '1/29/20': 6106, '1/30/20': 8237, '1/31/20': 9927, '2/1/20': 11437, '2/2/20': 13000, '2/3/20': 14500, '2/4/20': 16000, '2/5/20': 17500, '2/6/20': 19000, '2/7/20': 20500, '2/8/20': 22000, '2/9/20': 23500, '2/10/20': 25000, '2/11/20': 26500, '2/12/20': 28000, '2/13/20': 29500, '2/14/20': 31000, '2/15/20': 32500, '2/16/20': 34000, '2/17/20': 35500, '2/18/20': 37000, '2/19/20': 38500, '2/20/20': 40000, '2/21/20': 41500, '2/22/20': 43000, '2/23/20': 44500, '2/24/20': 46000, '2/25/20': 47500, '2/26/20': 49000, '2/27/20': 50500, '2/28/20': 52000, '2/29/20': 53500, '3/1/20': 55000, '3/2/20': 56500, '3/3/20': 58000, '3/4/20': 59500, '3/5/20': 61000, '3/6/20': 62500, '3/7/20': 64000, '3/8/20': 65500, '3/9/20': 67000, '3/10/20': 68500, '3/11/20': 70000, '3/12/20': 71500, '3/13/20': 73000, '3/14/20': 74500, '3/15/20': 76000, '3/16/20': 77500, '3/17/20': 79000, '3/18/20': 80500, '3/19/20': 82000, '3/20/20': 83500, '3/21/20': 85000, '3/22/20': 86500, '3/23/20': 88000, '3/24/20': 89500, '3/25/20': 91000, '3/26/20': 92500, '3/27/20': 94000, '3/28/20': 95500, '3/29/20': 97000, '3/30/20': 98500, '3/31/20': 100000}

Recoveries: {'1/22/20': 30, '1/23/20': 32, '1/24/20': 39, '1/25/20': 42, '1/26/20': 56, '1/27/20': 65, '1/28/20': 108, '1/29/20': 127, '1/30/20': 145, '1/31/20': 225, '2/1/20': 287, '2/2/20': 350, '2/3/20': 412, '2/4/20': 474, '2/5/20': 536, '2/6/20': 598, '2/7/20': 660, '2/8/20': 722, '2/9/20': 784, '2/10/20': 846, '2/11/20': 908, '2/12/20': 970, '2/13/20': 1032, '2/14/20': 1094, '2/15/20': 1156, '2/16/20': 1218, '2/17/20': 1280, '2/18/20': 1342, '2/19/20': 1404, '2/20/20': 1466, '2/21/20': 1528, '2/22/20': 1590, '2/23/20': 1652, '2/24/20': 1714, '2/25/20': 1776, '2/26/20': 1838, '2/27/20': 1900, '2/28/20': 1962, '2/29/20': 2024, '3/1/20': 2086, '3/2/20': 2148, '3/3/20': 2210, '3/4/20': 2272, '3/5/20': 2334, '3/6/20': 2396, '3/7/20': 2458, '3/8/20': 2520, '3/9/20': 2582, '3/10/20': 2644, '3/11/20': 2706, '3/12/20': 2768, '3/13/20': 2830, '3/14/20': 2892, '3/15/20': 2954, '3/16/20': 3016, '3/17/20': 3078, '3/18/20': 3140, '3/19/20': 3202, '3/20/20': 3264, '3/21/20': 3326, '3/22/20': 3388, '3/23/20': 3450, '3/24/20': 3512, '3/25/20': 3574, '3/26/20': 3636, '3/27/20': 3698, '3/28/20': 3760, '3/29/20': 3822, '3/30/20': 3884, '3/31/20': 3946}

23°C Mostly cloudy

Search

ENG IN 23:27 15.07.2024