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:

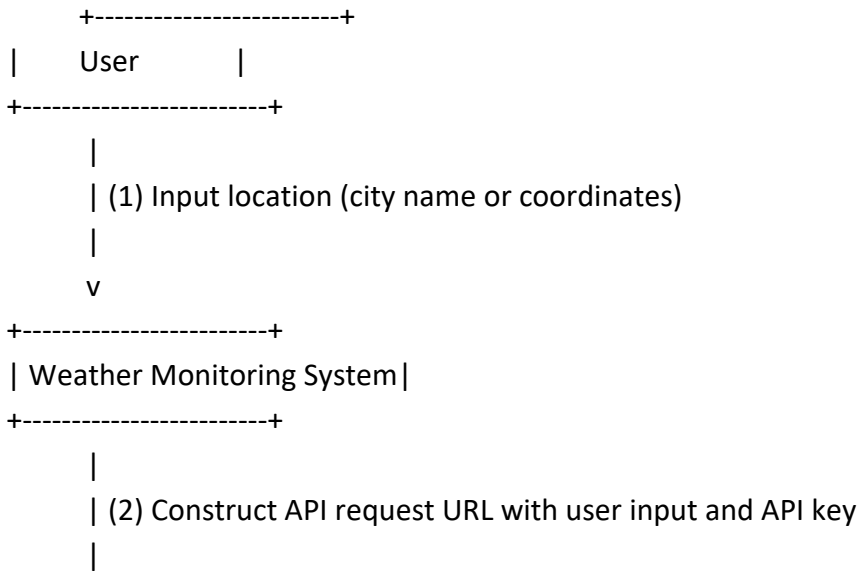
1. **Model the data flow for fetching weather information from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a weather API (e.g., OpenWeatherMap) to fetch real-time weather data.**
3. **Display the current weather information, including temperature,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

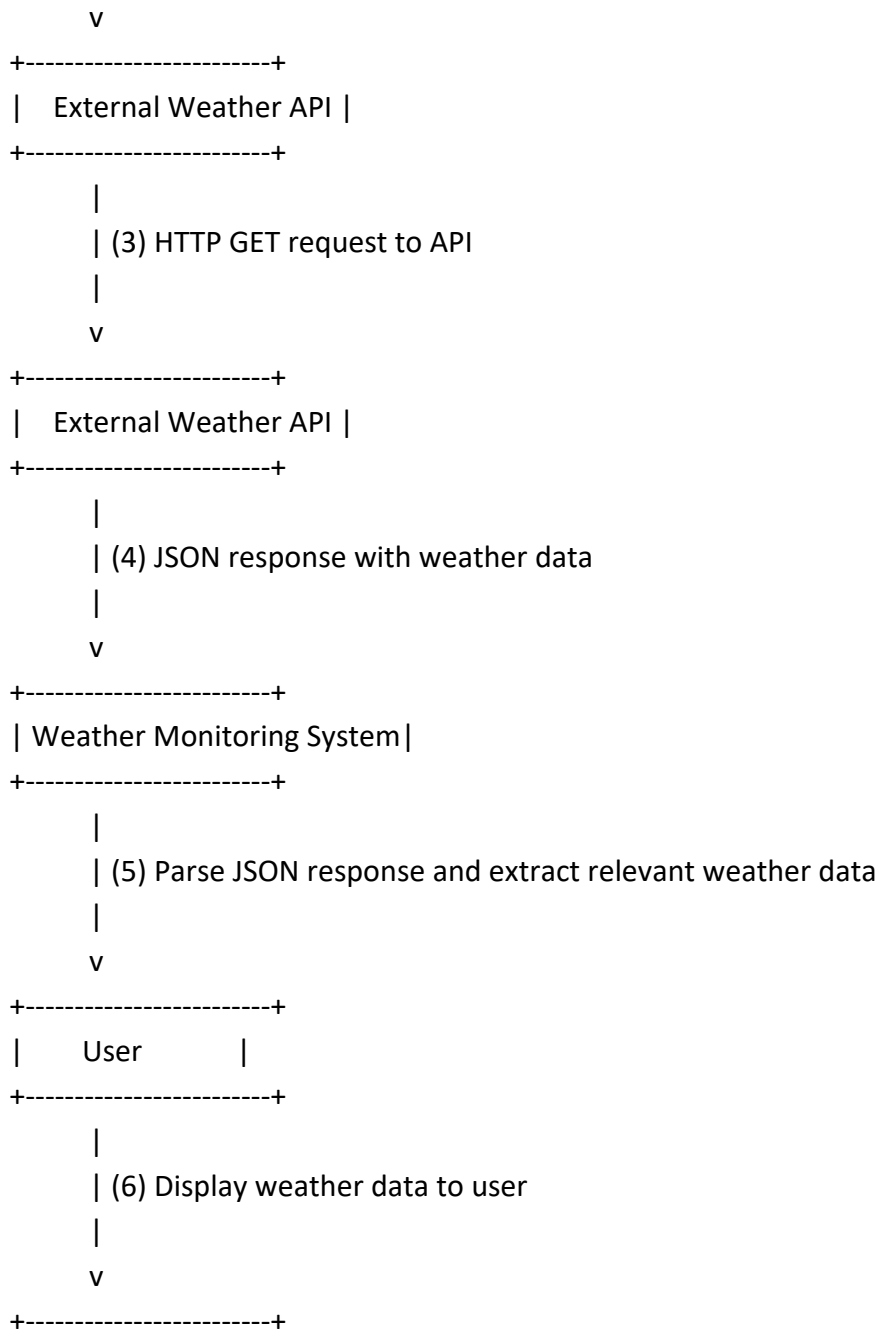
Deliverables:

- **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.**

Answer:

Model Data flow:





Python Code:

```
import requests

def get_weather_data(location):
    api_key = "b11f2fe52244a66eb93ee793f28c2d3b"  # Your provided API key
    base_url = "http://api.openweathermap.org/data/2.5/weather?"
    complete_url = base_url + "q=" + location + "&appid=" + api_key
```

```
    response = requests.get(complete_url)
    return response.json()
def display_weather_data(weather_data):
    if weather_data['cod'] != '404':
        main = weather_data['main']
        wind = weather_data['wind']
        weather_description = weather_data['weather'][0]['description']
        print(f"Temperature: {main['temp']}K")
        print(f"Humidity: {main['humidity']}%")
        print(f"Weather Description: {weather_description}")
        print(f"Wind Speed: {wind['speed']} m/s")
    else:
        print("City Not Found")

if _name_ == "_main_":
    location = input("Enter the city name: ")
    weather_data = get_weather_data(location)
    display_weather_data(weather_data)
```

Pseudocode:

1.Initialize the application

- Import necessary libraries
- Set up the API key and base URL for the weather API

2.Get user input

- Prompt the user to input a location (city name or coordinates)

3.Fetch weather data

- Build the request URL using the user input and API key
- Make an HTTP GET request to the weather API
- Parse the JSON response to extract relevant weather data

4.Display weather information

- Format and display the current weather information: temperature, weather conditions, humidity, and wind speed

5.Error handling

- Handle any errors that may occur during the API request or data parsing

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:

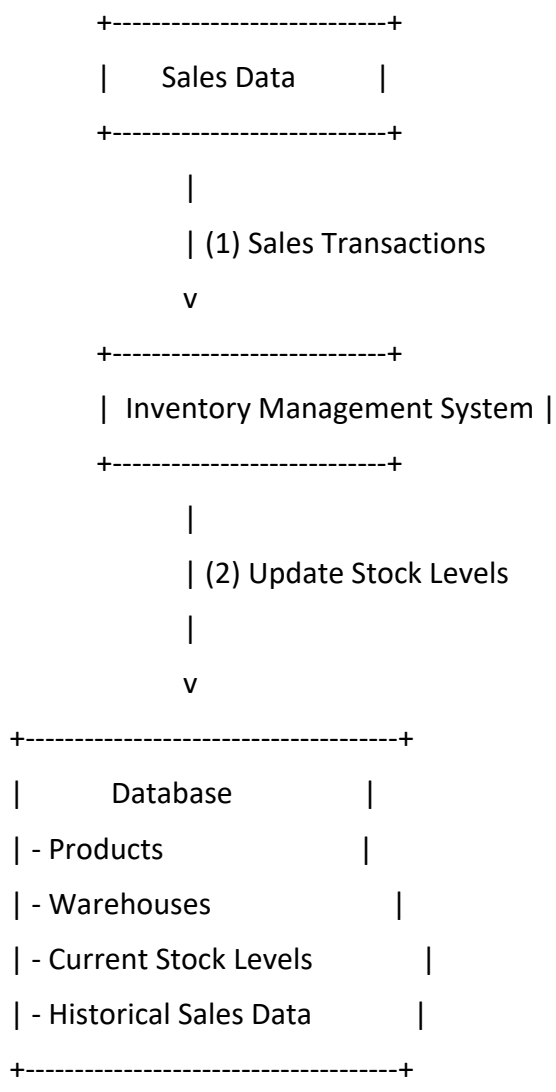
1. **Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.**
2. **Implement an inventory tracking application: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a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

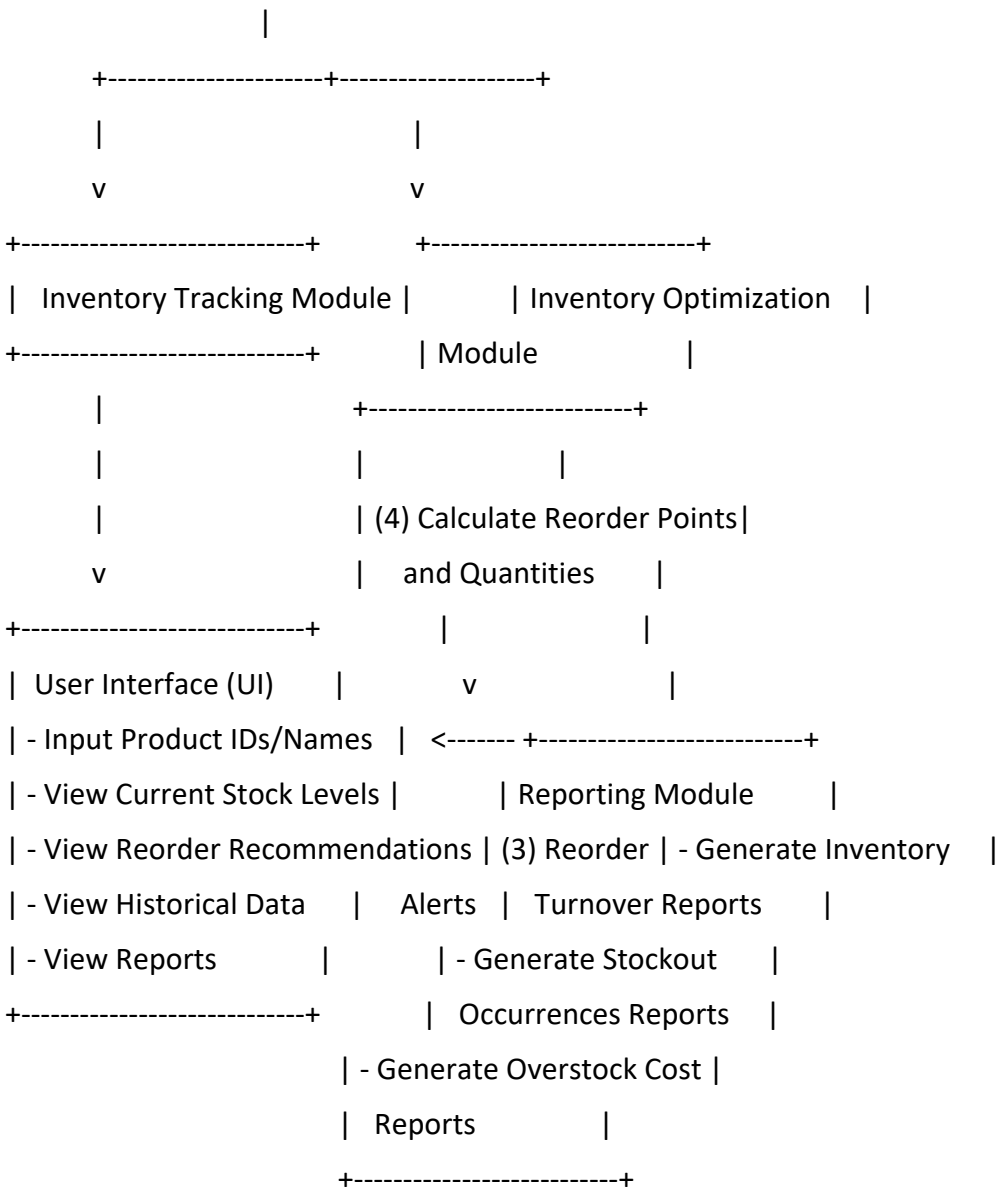
Deliverables:

- **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.**

Answer:

Model Data flow:





Python Code:

```
class Product:
    def _init_(self, product_id, name, category, current_stock, reorder_point, reorder_quantity, lead_time):
        self.product_id = product_id
        self.name = name
        self.category = category
        self.current_stock = current_stock
        self.reorder_point = reorder_point
        self.reorder_quantity = reorder_quantity
        self.lead_time = lead_time
```

```
        self.historical_sales = []
    def add_sales_data(self, sales):
        self.historical_sales.append(sales)
        self.current_stock -= sales  # Update current stock after sale
class Warehouse:
    def _init_(self, warehouse_id, location):
        self.warehouse_id = warehouse_id
        self.location = location
        self.products = {}
    def add_product(self, product):
        self.products[product.product_id] = product
    def track_inventory(self):
        print("\n--- Inventory Tracking ---")
        for product in self.products.values():
            if product.current_stock < product.reorder_point:
                print(f"Reorder Alert for: {product.name}")
                print(f"Current Stock: {product.current_stock}")
                print(f"Recommended Order Quantity: {product.reorder_quantity}")
            else:
                print(f"{product.name} is sufficiently stocked.")
    def generate_report(self):
        print("\n--- Inventory Report ---")
        for product in self.products.values():
            print(f"Product: {product.name}")
            print(f"Current Stock: {product.current_stock}")
            print(f"Turnover Rate: {self.calculate_turnover_rate(product)}")
    def calculate_turnover_rate(self, product):
        total_sales = sum(product.historical_sales)
        average_stock = (product.current_stock + product.current_stock) / 2
        turnover_rate = total_sales / average_stock if average_stock > 0 else 0
        return turnover_rate
def calculate_eoq(annual_demand, ordering_cost, holding_cost):
    if holding_cost > 0:
        return (2 * annual_demand * ordering_cost / holding_cost) ** 0.5
    return 0
def main():
```

```
warehouse = Warehouse(1, "Main Warehouse")
# Example products
product1 = Product(101, "Laptop", "Electronics", 15, 5, 20, 2)
product1.add_sales_data(3)
product1.add_sales_data(4)
product2 = Product(102, "Smartphone", "Electronics", 30, 10, 15, 3)
product2.add_sales_data(5)
product2.add_sales_data(6)
warehouse.add_product(product1)
warehouse.add_product(product2)
while True:
    print("\nOptions:")
    print("1. Track Inventory")
    print("2. Generate Report")
    print("3. Add Sales Data")
    print("4. Calculate EOQ")
    print("5. Exit")
    choice = input("Select an option: ")
    if choice == '1':
        warehouse.track_inventory()
    elif choice == '2':
        warehouse.generate_report()
    elif choice == '3':
        product_id = int(input("Enter Product ID: "))
        sales = int(input("Enter sales data: "))
        if product_id in warehouse.products:
            warehouse.products[product_id].add_sales_data(sales)
            print("Sales data updated.")
        else:
            print("Product not found.")
    elif choice == '4':
        product_id = int(input("Enter Product ID: "))
        if product_id in warehouse.products:
            annual_demand = sum(warehouse.products[product_id].historical_sales)
            ordering_cost = 50  # Example ordering cost
            holding_cost = 2    # Example holding cost per unit
```

```
            eoq = calculate_eoq(annual_demand, ordering_cost, holding_cost)
            print(f"Optimal Order Quantity (EOQ) for {warehouse.products[product_id].name}:
{eoq:.2f}")
        else:
            print("Product not found.")
    elif choice == '5':
        break
    else:
        print("Invalid option, try again.")
if _name_ == "_main_":
  main()
```

Pseudocode:

1. Define class Product:
    - Initialize with product_id, name, current_stock
    - Initialize empty list for historical_sales
2. Define class Warehouse:
    - Initialize with warehouse_id, name
    - Initialize empty dictionary for inventory
3. Define class InventorySystem:
    - Initialize with empty dictionaries for products and warehouses
4. Define method add_product(product):
    - Add product to products dictionary
5. Define method add_warehouse(warehouse):
    - Add warehouse to warehouses dictionary
6. Define method update_stock(product_id, warehouse_id, quantity):
    - Retrieve warehouse by warehouse_id
    - Update inventory for product_id in warehouse
    - Retrieve product by product_id
    - Update product current_stock
7. Define method record_sale(product_id, quantity):
    - Retrieve product by product_id
    - Decrease product current_stock by quantity

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.**

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.**

Answer:

Data Flow Model:

Start
|
V
[User Input: Starting Point, Destination]
|
V
[Send Request to Traffic API]
|
V
[Receive Traffic Data]
|
V
[Process Traffic Data]
|
V
[Display Traffic Conditions, Estimated Travel Time, Incidents]
|
V
[Optionally Display Alternative Routes]

|
V
End

Python code:

```
 import requests
API_KEY = 'YOUR_GOOGLE_MAPS_API_KEY'
BASE_URL = 'https://maps.googleapis.com/maps/api/directions/json'
def fetch_traffic_data(start, destination):
   params = {
      'origin': start,
      'destination': destination,
      'key': API_KEY,
      'departure_time': 'now'
   }
   response = requests.get(BASE_URL, params=params)
   data = response.json()
   if data['status'] == 'OK':
      route = data['routes'][0]
      legs = route['legs'][0]
      traffic_data = {
         'traffic_conditions': legs['traffic_speed_entry'],
         'travel_time': legs['duration_in_traffic']['text'],
         'incidents': route['warnings']
      }
      return traffic_data
   else:
      raise Exception('Error fetching traffic data: ' + data['status'])
def display_traffic_data(traffic_data):
   print("Current Traffic Conditions: ")
   for condition in traffic_data['traffic_conditions']:
      print(f"  - Speed: {condition['speed']} km/h")
   print(f"Estimated Travel Time: {traffic_data['travel_time']}")
   print("Incidents or Delays: ")
```

```
    if traffic_data['incidents']:
        for incident in traffic_data['incidents']:
            print(f"  - {incident}")
    else:
        print("  - No incidents or delays reported.")
def main():
    start = input("Enter the starting point: ")
    destination = input("Enter the destination: ")
    try:
        traffic_data = fetch_traffic_data(start, destination)
        display_traffic_data(traffic_data)
    except Exception as e:
        print(f"An error occurred: {e}")
if __name__ == "__main__":
    main()
```

Pseudocode:

1. Define constants for the API key and base URL of the traffic monitoring API.

2. Create a function `fetch_traffic_data(start, destination)`:

- Construct the API request URL with the start and destination points.
- Send a request to the API and get the response.
- Parse the response to extract traffic data (conditions, travel time, incidents).
- Return the extracted traffic data.

3. Create a function `display_traffic_data(traffic_data)`:

- Print current traffic conditions.
- Print estimated travel time.
- Print any incidents or delays.
- Suggest alternative routes if traffic is heavy.

4. Create a function `main()`:

- Prompt the user for a starting point and destination.
- b. Call `fetch_traffic_data(start, destination)` to get real-time traffic data.
- c. Call `display_traffic_data(traffic_data)` to display the information.

5. Execute the `main()` function.

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.

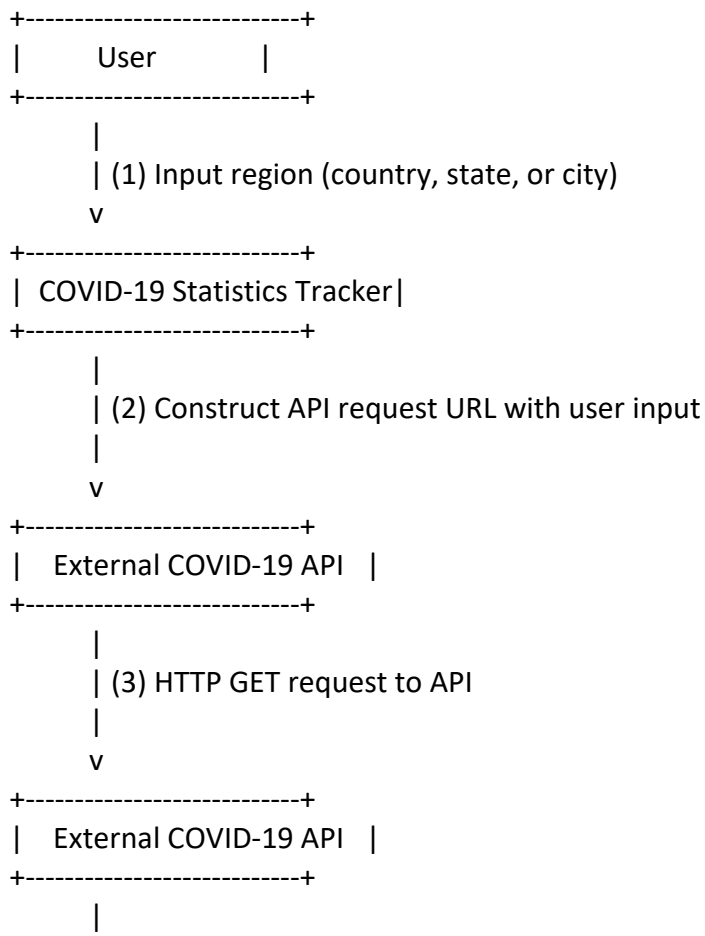
Tasks:

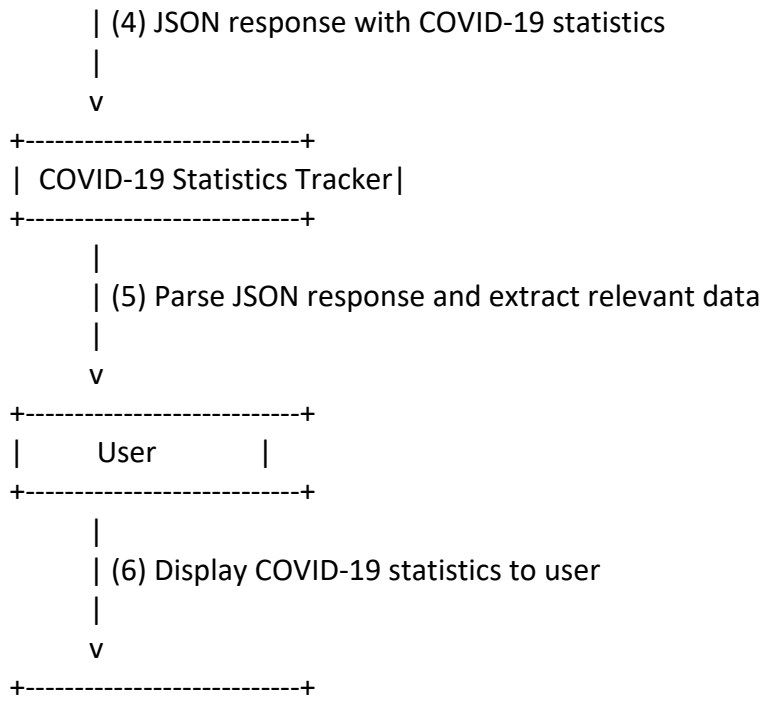
1. **Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.**
2. **Implement a Python application that integrates with a COVID-19 statistics API (e.g., disease.sh) to fetch real-time data.**
3. **Display the current number of cases, recoveries, and deaths 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.**

Answer :





Python Code:

```
import requests
def fetch_covid_data(region):
    API_URL = f"https://disease.sh/v3/covid-19/countries/{region}"
    response = make_api_call(API_URL)
    if response.status_code == 200:
        return response.json()
    else:
        return f"Error fetching data: {response.status_code}"
def make_api_call(url):
    headers = {"Accept": "application/json"}
    return requests.get(url, headers=headers)
def display_statistics(data):
    print(f"COVID-19 Statistics for {data['country']}:")
    print(f"Total Cases: {data['cases']}")
    print(f"Total Recoveries: {data['recovered']}")
    print(f"Total Deaths: {data['deaths']}")
```

```
def main():
    region = input("Enter the region (country, state, or city): ")
    covid_data = fetch_covid_data(region)
    if isinstance(covid_data, dict):
        display_statistics(covid_data)
    else:
        print(covid_data)
if _name_ == "_main_":
    main()
```

Pseudocode:

1. Define class CovidStatsTracker:

- Initialize with api_key and base_url
- Define method get_covid_stats(region):

2.Construct request URL using base_url, region, and API key

- Send HTTP GET request to the API
- If response is successful:
- Parse JSON response
- Extract current cases, recoveries, and deaths
- Return extracted data

- Else:
  - Return None

3.Define method display_stats(data):

- If data is not None:
  - Print current cases, recoveries, and deaths
- Else:
  - Print error message