1)Problem 1: Real-Time Weather Monitoring System

Scenario:

To 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.

dATA FLOW DIAGRAM:-

User Input (Location) -> Python Application -> Weather API (e.g., OpenWeatherMap)

^ |

| v

Display Weather Data <- JSON Response <- API Request

# Import necessary libraries

import requests

import json

# Function to fetch weather data from API

def fetch\_weather\_data(location):

api\_key = 'your\_api\_key\_here' # Replace with your actual API key

base\_url = 'https://api.openweathermap.org/data/2.5/weather'

# Handling different location input types (city name or coordinates)

if location.isdigit(): # Assuming input is coordinates (latitude, longitude)

params = {'lat': location.split(',')[0], 'lon': location.split(',')[1], 'appid': api\_key, 'units': 'metric'}

else: # Assuming input is city name

params = {'q': location, 'appid': api\_key, 'units': 'metric'}

try:

response = requests.get(base\_url, params=params)

data = response. json()

if response. Status code == 200:

return data # Return JSON data received from API

else:

print(f" Error fetching data: {data['message']}")

return None

except requests. exceptions. Request Exception as e:

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

return None

# Function to display weather data

def display\_weather\_data(data):

if data:

# Extracting relevant weather information

temperature = data['main']['temp']

weather\_desc = data['weather'][0]['description']

humidity = data['main']['humidity']

wind\_speed = data['wind']['speed']

# Displaying weather information

print(f"Temperature: {temperature} °C")

print(f"Weather: {weather\_desc}")

print(f"Humidity: {humidity}%")

print(f"Wind Speed: {wind\_speed} m/s")

else:

print("No weather data available.")

# Main function to run the application

def main():

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

weather\_data = fetch\_weather\_data(location)

display\_weather\_data(weather\_data)

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

main()

Documentation

API Integration

This Python application integrates with the OpenWeatherMap API to fetch real-time weather data. It uses the requests library to send HTTP requests to the API endpoint (https://api.openweathermap.org/data/2.5/weather) with parameters such as API key (appid), location (q for city name or lat, lon for coordinates), and units (metric for Celsius).

Methods Used

Fetching Weather Data (fetch\_weather\_data function):

Constructs the API request based on user input (city name or coordinates).

Sends an HTTP GET request to the API and handles the JSON response.

Checks for errors and prints appropriate error messages.

Displaying Weather Data (display\_weather\_data function):

Extracts relevant weather information such as temperature, weather description, humidity, and wind speed from the JSON response.

Prints the weather information in a readable format.

Assumptions

Input Handling: The application assumes the user inputs either a city name or coordinates in the format latitude,longitude.

API Key: You need to replace 'your\_api\_key\_here' with your actual OpenWeatherMap API key for the application to function properly.

Potential Improvements

Error Handling: Enhance error handling to cover more edge cases (e.g., network errors, invalid input formats).

User Interface: Develop a graphical user interface (GUI) or web interface for better user interaction.

Caching: Implement caching mechanisms to reduce the number of API calls for repetitive requests within a short period.

Additional Weather Parameters: Expand the application to fetch and display more detailed weather parameters provided by the API.

This system provides a basic framework for fetching and displaying weather data from an external API, which can be expanded and improved based on specific requirements and additional features desired for the weather monitoring system.