# Assignment – 1

Problem 1: Real-Time Weather Monitoring System

1. Data Flow Diagram

A Data Flow Diagram (DFD) helps visualize how data flows within a system. Below is a high-level description of the data flow for a real-time weather monitoring system:

Data Flow Steps:

1. User Input: The user provides input, such as a city name or geographic coordinates.

2. API Request: The application sends a request to the external weather API, passing the user's input (city name or coordinates).

3. Weather API: The API processes the request and fetches the current weather data for the specified location.

4. API Response:The weather API returns the data, including temperature, weather conditions, humidity, and wind speed, to the application.

5. Data Parsing: The application parses the received data to extract relevant information.

6. Display Weather Data:The application displays the parsed weather data to the user.

Data Flow Diagram(DFD):

[User] ---> (Enter Location) ---> [Weather Monitoring System] ---> (Send Request) ---> [Weather API]

[Weather API] ---> (Return Data) ---> [Weather Monitoring System] ---> (Display Data) ---> [User]

```

2. Pseudocode for Weather Monitoring System

Here is a pseudocode outline for the weather monitoring system:

BEGIN

PROMPT user to input a location (city name or coordinates)

IF input is city name:

SET location\_ parameter = city name

ELSE IF input is coordinates:

SET location\_parameter = coordinates

SEND a request to the weather API with location\_parameter

RECEIVE response from the weather API

PARSE the API response to extract:

- temperature

- weather conditions

- humidity

- wind speed

DISPLAY the weather information to the user

END

```

3. Python Implementation

The following is a basic implementation in Python, using the OpenWeatherMap API as an example:

```python

import requests

# Define API Key and base URL

API\_KEY = 'your\_openweathermap\_api\_key'

BASE\_URL = 'http://api.openweathermap.org/data/2.5/weather'

def get\_weather\_data(location):

# Set up parameters for API request

params = {

'q': location,

'appid': API\_KEY,

'units': 'metric' # Use metric units for temperature (Celsius)

}

# Send request to OpenWeatherMap API

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

# Parse response JSON

data = response. Json ()

# Check for successful response

if response.status\_code == 200:

# Extract weather information

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

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

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

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

# Display the weather information

print(f"Location: {location}")

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

print(f"Weather Conditions: {weather\_conditions}")

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

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

else:

# Handle errors (e.g., invalid location)

print("Error: Unable to fetch weather data. Please check the location and try again.")

def main():

# Prompt user for location input

location = input("Enter the city name: ")

# Fetch and display weather data

get\_weather\_data(location)

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

main()

```

4. Documentation of API Integration

API Used: [OpenWeatherMap API] (https://openweathermap.org/api)

Methods Used:

- `requests. Get () `: Sends a GET request to the OpenWeatherMap API with the specified location and API key.

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

Weather Data Fields:

- \*\*`main['temp'] `\*\*: Current temperature in Celsius.

- \*\*`weather [0] ['description'] `\*\*: Weather conditions (e.g., clear sky, rain).

- \*\*`main['humidity'] `\*\*: Current humidity percentage.

- \*\*`wind['speed'] `\*\*: Wind speed in meters per second.

5. Explanation of Assumptions and Potential Improvements

Assumptions:

- The user inputs a valid city name or coordinates.

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

Potential Improvements:

Error Handling: Add more robust error handling for various scenarios, such as network errors, invalid API keys, or incorrect location inputs.

User Interface: Implement a graphical user interface (GUI) using a library like Tkinter or PyQt to improve user interaction.

Location Options: Allow users to search for locations using more diverse parameters, such as ZIP codes or IP addresses.

Cache Data: \*Implement caching to reduce the number of API requests for frequently requested locations.

## Output:

Enter the city name: London

Location: London

Temperature: 18.5°C

Weather Conditions: scattered clouds

Humidity: 77%

Wind Speed: 4.6 m/s