Project – 1 : Real-Time Weather Monitoring System

**1. Data Flow Diagram**

Here is a high-level data flow diagram (DFD) illustrating the interaction between the application and the API:

[User] -> [Weather Monitoring System] -> [Weather API]

| |

v v

[Enter Location] [Fetch Weather Data]

| |

v v

[Display Weather Information] <--- [Return Weather Data]

**2. Pseudocode and Implementation**

**Pseudocode:**

START

PROMPT user for location (city name or coordinates)

FETCH weather data from the Weather API using the provided location

IF data fetch is successful THEN

DISPLAY current weather information including temperature, weather conditions, humidity, and wind speed

ELSE

DISPLAY error message

END

**Python Implementation:** First, you need to sign up on OpenWeatherMap (or any weather API provider) to get an API key.

import requests

def get\_weather\_data(location, api\_key):

base\_url = "http://api.openweathermap.org/data/2.5/weather"

params = {

'q': location,

'appid': api\_key,

'units': 'metric' # Use 'imperial' for Fahrenheit

}

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

return response.json()

def display\_weather(data):

if data.get('cod') != 200:

print(f"Error: {data.get('message')}")

return

city = data['name']

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

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

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

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

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

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

print(f"Condition: {weather\_condition}")

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

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

def main():

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

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

weather\_data = get\_weather\_data(location, api\_key)

display\_weather(weather\_data)

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

main()

### 3. Documentation of API Integration and Methods

#### API Integration

* **API Used**: OpenWeatherMap (http://api.openweathermap.org)
* **Endpoint**: /data/2.5/weather
* **Parameters**:
  + q: City name (e.g., "London")
  + appid: Your API key
  + units: Units of measurement (metric for Celsius, imperial for Fahrenheit)

#### Methods

* **get\_weather\_data(location, api\_key)**:
  + Sends a GET request to the OpenWeatherMap API with the provided location and API key.
  + Returns the JSON response.
* **display\_weather(data)**:
  + Parses the JSON response to extract and print weather details such as temperature, weather conditions, humidity, and wind speed.
  + Handles error messages if the API call fails.

### 4. Explanation of Assumptions and Potential Improvements

#### Assumptions

* The user inputs the correct city name.
* The API key is valid and has sufficient quota for the requests.
* The network connection is stable for API calls.

#### Potential Improvements

* **Error Handling**: Improve error handling to cover more scenarios, such as invalid city names, API limits, and network issues.
* **User Interface**: Develop a graphical user interface (GUI) using a library like Tkinter for a more user-friendly experience.
* **Extended Forecast**: Fetch and display extended weather forecasts (e.g., 5-day forecast).
* **Localization**: Allow users to input coordinates (latitude and longitude) instead of city names.
* **Caching**: Implement caching to reduce the number of API calls for frequently requested locations.

This project helps students understand API integration, data parsing, and real-time data handling, which are valuable skills in modern software development.