Weather Forecast Service Using HTTP Server-Client Model

**Submitted By**:  
[B.V.PRAVEEN ,CS22B2033]

**Aim:**

The aim of this project is to implement a client-server architecture using Python to retrieve real-time weather data from the OpenWeather API. The project demonstrates:

* Establishing communication between a client and a server using HTTP.
* Handling dynamic inputs and processing responses from external APIs.
* Parsing and displaying the relevant weather data like temperature, humidity, wind speed, latitude, and longitude in a user-friendly format.

**Introduction**

* **Overview:**The project involves creating a networked application where a client communicates with a server to fetch weather data for a given city. The server queries the OpenWeather API to retrieve the information.
* **Importance and Relevance:**  
  Weather forecasting applications are widely used in various domains like travel, agriculture, and disaster management. Understanding how such systems work provides insights into real-world client-server communication.
* **Problem Statement:**  
  The task is to simulate a small-scale weather application where the server fetches data from an external API, processes the data, and sends it back to the client for display.

**System design**

**Architecture**:  
 The system follows a client-server architecture:

* **Client**: Sends a city name as a request to the server.
* **Server**: Receives the request, queries the OpenWeather API for weather details, and sends the processed data back to the client.

[Client] --> (Request: city) --> [Server] --> [OpenWeather API]

^--- (Response: weather details) <--------|

**Protocol/Concept Details:**

* Uses HTTP/1.1 for data transmission.
* Implements error handling for cases like invalid city names or API errors.
* Parses JSON responses to extract specific weather details.

**Tools and Technologies:**

* Programming Language: Python.
* Libraries: socket, requests, json, logging.
* External Service: OpenWeather API for real-time weather data.

**Implementation Details**

**Step-by-Step Methodology:**

1. **Server Setup:**

* A Python server listens on port 8080 for incoming client requests.
* Parses the client’s request to extract the city name and queries the OpenWeather API using the provided API key.

1. **Client Setup**:

* The client sends a city name to the server over a TCP socket connection.
* Displays the weather details received from the server.

1. **API Integration**:

* The server sends an HTTP GET request to the OpenWeather API.
* Retrieves data in JSON format and extracts fields like temperature, humidity, latitude, longitude, etc.

1. **Error Handling**:
   * Logs errors like invalid API keys or city names.
   * Handles network issues gracefully.
2. **Testing**:
   * Tested with valid and invalid city names.
   * Verified the response format and connection stability.

**Testing and Results**

**Testing Scenarios**:

* + Valid city names (e.g., London, New York):

Verified correct data is returned.

* + Invalid city names (e.g., xyz123): Checked error handling.
  + Network disconnection: Verified graceful failure of the application.

**Results:**

* The application successfully retrieves and displays weather data for valid city names. For invalid cities or errors, appropriate error messages are logged.

**Sample Output**:

**Client Side:**

**Server Side:**

**Discussions**

**Challenges Faced:**

* + Understanding the OpenWeather API structure and handling errors like invalid API keys.
  + Parsing JSON responses and managing edge cases for invalid inputs.
  + Configuring logging for debugging network issues.

**Limitations:**

* + Only supports single-word city names (multi-word cities require additional formatting).
  + No support for concurrent clients in the current setup.

**Future Enhancements**

* Add support for concurrent client requests using threading or asynchronous programming.
* Improve input validation to handle special characters or multi-word city names.
* Enhance the user interface by integrating a graphical front-end using Tkinter.
* Include additional weather parameters like UV index or hourly forecasts.

**References**

* OpenWeather API Documentation: https://openweathermap.org/api
* Python socket Library Documentation: <https://docs.python.org/3/library/socket.html>
* Python requests Library Documentation: https://docs.python-requests.org/en/latest/
* "Computer Networking: Principles, Protocols, and Practice" by Olivier Bonaventure.