Operating Systems - Lab
(CL - 2018)
LABORATORY MANUAL
SPRING - 2025

LAB 06
Building, Deploying, and Testing a RESTful API using Flask in a Client-Server Model

_____ ___

_____ _____ ___
STUDENT NAME ROLL NO SEC

_____
LAB ENGINEER'S SIGNATURE & DATE

Marks Awarded:

Lab 06: Building, Deploying, and Testing a RESTful API using Flask in a Client-Server Model

Objective:

The objective of this lab is to introduce students to **APIs (Application Programming Interfaces)** and their role in modern software development. Students will:

- Understand the fundamentals of APIs, including RESTful APIs and HTTP methods.
- Learn how to interact with APIs using tools like **Postman, cURL, or Python requests**.
- Develop a basic API using **Flask/Django** and implement **GET, POST, PUT, DELETE** requests.
- Implement a **Flask server** that fetches real-time weather data from the **OpenWeather API**.
- Develop a **client application** to request weather data from the server.
- Learn how to send HTTP requests, handle responses, and process JSON data.

Introduction:

In the modern software landscape, **APIs** serve as a bridge between different applications, enabling seamless communication and data exchange. APIs define a set of rules that allow software components to interact, whether within the same system or across the internet.There are two main types of APIs:

1. **RESTful APIs (Representational State Transfer):** Based on HTTP, these APIs use standard methods like GET (retrieve data), POST (send data), PUT (update data), and DELETE (remove data).
2. **SOAP APIs (Simple Object Access Protocol):** These use XML-based messaging and are more rigid but highly secure.

This lab will focus on RESTful APIs, where students will first explore existing APIs and then build their own simple API using **Flask/Django**. Understanding how APIs work is essential for **web development, mobile apps, cloud computing, IoT, and microservices**.

Required Materials

- **Software & Tools:**
  - Python 3.x (Download)
  - Flask (Install using pip install flask)
  - Postman (for testing API calls)
  - OpenWeather API Key (Sign up here)
- **Hardware:**

- A computer/laptop with an internet connection. **requests library** (for consuming APIs) – Install using:
- pip install requests
- OpenWeather API Key (Sign up here)

Lab Instructions

Step 1: Install Required Libraries

Before running the server and client code, install the required Python libraries:

pip install flask requests

Step 2: Implement the Server Code

1. Open a new Python file and name it server.py.
2. Copy and paste the following code:

```
from flask import Flask, request, jsonify
import requests

app = Flask(__name__)

# Replace with your OpenWeather API key
API_KEY = 'your_api_key_here'
BASE_URL = 'http://api.openweathermap.org/data/2.5/weather'

@app.route('/weather', methods=['GET'])
def get_weather():
    city = request.args.get('city')

    if not city:
        return jsonify({'error': 'City parameter is required'}), 400

    # Query the weather API
    params = {
        'q': city,
        'appid': API_KEY,
        'units': 'metric'  # Use 'imperial' for Fahrenheit
    }
    response = requests.get(BASE_URL, params=params)

    if response.status_code != 200:
        return jsonify({'error': 'Could not retrieve weather data'}),
response.status_code

    weather_data = response.json()

    # Extract relevant information
    result = {
        'city': weather_data['name'],
        'temperature': weather_data['main']['temp'],
        'description': weather_data['weather'][0]['description'],
    }

    return jsonify(result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

3. **Run the Server** using the command:
4. python server.py
5. The server will start on **http://127.0.0.1:5000**.

Step 3: Implement the Client Code

1. Create another Python file named client.py.
2. Copy and paste the following client-side code:

```
import requests

def get_weather(city):
    url = 'http://127.0.0.1:5000/weather'
    params = {'city': city}

    response = requests.get(url, params=params)

    if response.status_code == 200:
        weather_data = response.json()
        print(f"City: {weather_data['city']}")
        print(f"Temperature: {weather_data['temperature']}°C")
        print(f"Description: {weather_data['description']}")
    else:
        print(f"Error: {response.json().get('error', 'Unknown error occurred')}")

if __name__ == "__main__":
    city_name = input("Enter city name: ")
    get_weather(city_name)
```

3. **Run the Client Code** in another terminal window:
4. python client.py
5. Enter a city name (e.g., **Multan**) when prompted.
6. The client will display the current **temperature** and **weather description** of the city.

Step 4: Testing with Postman

1. Open **Postman**.
2. Create a **new GET request** to:
3. http://127.0.0.1:5000/weather?city=Multan
4. Click **Send** and check if the API returns weather details in JSON format.

Server Output:

When running `server.py`, you should see:

```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
>>> runfile('C:/Users/User/Desktop/t/server.py', wdir='C:/Users/User/Desktop/t')
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [02/Sep/2024 16:46:48] "GET /weather?city=Multan HTTP/1.1" 200 -
runfile('C:/Users/User/Desktop/t/server.py', wdir='C:/Users/User/Desktop/t')
127.0.0.1 - - [02/Sep/2024 16:49:26] "GET /weather?city=Karachi HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2024 16:49:47] "GET /weather?city=Lahore HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2024 16:50:04] "GET /weather?city=Hubco HTTP/1.1" 404 -
127.0.0.1 - - [02/Sep/2024 16:50:23] "GET /weather?city=sindh HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2024 16:50:42] "GET /weather?city=Lodhran HTTP/1.1" 200 -
127.0.0.1 - - [02/Sep/2024 16:51:00] "GET /weather?city=shujabad HTTP/1.1" 404 -
127.0.0.1 - - [02/Sep/2024 16:51:12] "GET /weather?city=Bahawalpur HTTP/1.1" 200 -
```

Client Output (Example for Multan):

```
Enter city name: Multan
City: London
Temperature: 12.3°C
Description: clear sky
```

```
>>> runfile('C:/Users/User/Desktop/t/client.py', wdir='C:/Users/User/Desktop/t')
Enter city name: Karachi
City: Karachi
Temperature: 31.9°C
Description: clear sky
>>> runfile('C:/Users/User/Desktop/t/client.py', wdir='C:/Users/User/Desktop/t')
Enter city name: Lahore
City: Lahore
Temperature: 31.99°C
Description: smoke
```

Postman Response (Example JSON Output):

```
{
    "city": "London",
    "temperature": 12.3,
    "description": "clear sky"
}
```

Home Task Assignments

Task 1: Extend the API Response

Modify the Flask server (`server.py`) to include additional weather details in the response:

- **Humidity**
- **Wind Speed**
- **Pressure**

These details are available in the OpenWeather API response under `main` (for pressure, humidity) and `wind` (for wind speed).

Expected JSON Response Example:

```
{
    "city": "New York",
    "temperature": 18.5,
    "description": "few clouds",
    "humidity": 70,
    "wind_speed": 4.6,
    "pressure": 1015
}
```

Task 2: Implement Error Handling for Invalid City Names

- Modify the server code to handle cases where the user enters an invalid city name.
- The API should return a custom error message instead of the default OpenWeather error.

Expected Output (for an invalid city):

```
{
  "error": "Invalid city name. Please enter a valid city."
}
```

Hint:
Check the OpenWeather API response status code. If 404, return a custom message.

Task 3: Fetch Weather Data for Multiple Cities

Modify the client (`client.py`) to allow the user to enter multiple city names (comma-separated) and fetch the weather data for all of them.

✓ Example Input:

```
Enter city names (comma-separated): London, Paris, New York
```

✓ Expected Output:

```
City: London
Temperature: 15°C
Description: clear sky

City: Paris
Temperature: 17°C
Description: scattered clouds

City: New York
Temperature: 18°C
Description: few clouds
```

✓ Hint: Use a loop to iterate over multiple city names and make multiple API requests.

Task 4: Deploy the API Online

➢ Deploy the API Online using Ngrok

✓ Steps:

1. Create a GitHub repository for the project and upload server.py.
2. Run the Flask API locally on port 5000.
3. Install and start Ngrok to expose the local server.
4. Copy the public API URL generated by Ngrok.
5. Modify the client (client.py) to use the Ngrok URL instead of 127.0.0.1:5000.

Lab Report: