**THE SUPERIOR UNIVERSITY LAHORE** 

**LAB#8**

**Semester: 4th Se~~ctio~~n: AI (B)**

**Faculty of Computer Science and Information Technology Deadline:**

**Subject: PAI LAB Total Marks:**

**Name: M.Hassan Shahid**

**Roll No: 059**

***Instructions:***

* Copying of the assignment willresult in failure.
* Assignment should be submitted in word or pdf.

# Flask Weather App

## Introduction

This project is a simple weather application built using Flask for the backend and HTML, CSS, and JavaScript for the front end. The goal is to let users enter the name of a city and see the current weather data fetched from the OpenWeatherMap API. It’s a great starting point for understanding how to combine server-side processing with an interactive web interface.

## Tools and Technologies

* **Python and Flask:** We use Python for the logic and Flask as the web framework to create a RESTful API.
* **Requests Library:** This Python library makes it easy to send HTTP requests to external services—in this case, the OpenWeatherMap API.
* **OpenWeatherMap API:** This service provides up-to-date weather information. (Remember to sign up for an API key!)
* **HTML, CSS, and JavaScript:** These are used to build the user interface, style the page, and handle user interactions.
* **Jinja2:** Flask’s templating engine, which helps in rendering HTML files.

## Code Explanation

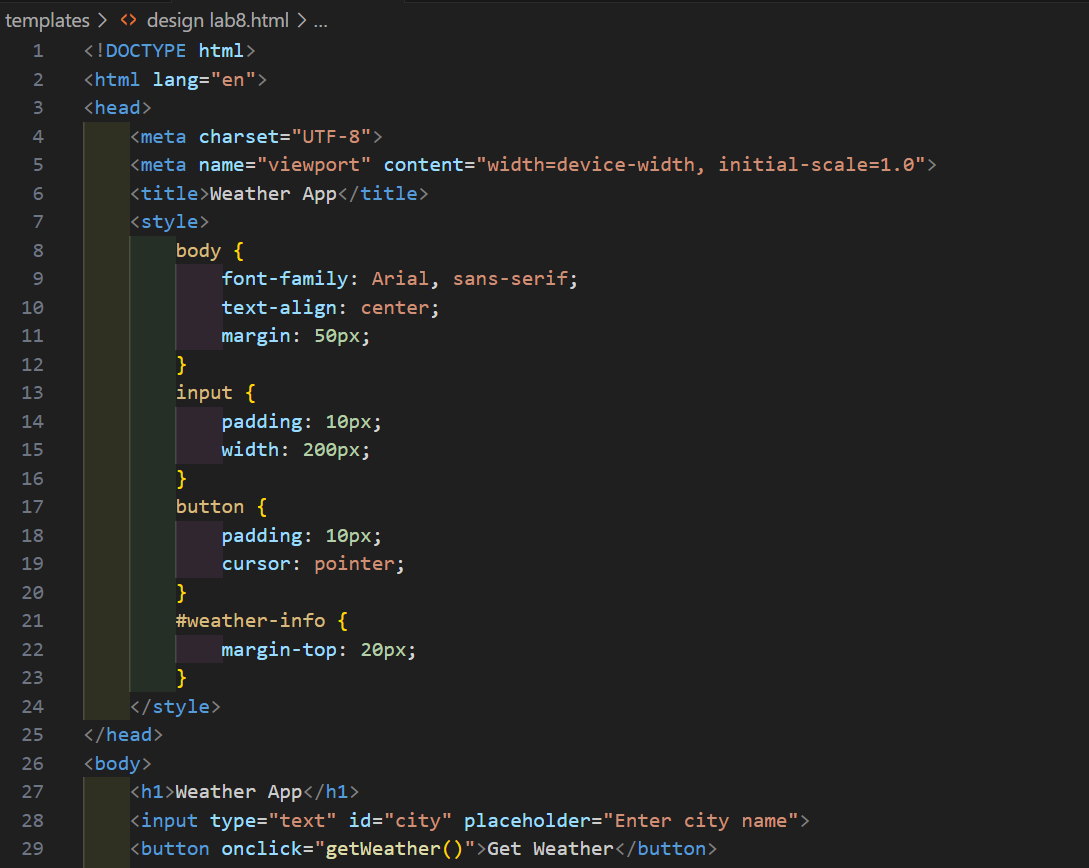
### Backend (Flask Code)

backend code for our Flask application. This code creates two endpoints: one for serving the main page and another for fetching weather data based on a city name.

#### RESULTS:C:\Users\Admin\Pictures\Screenshots\Screenshot 2025-03-17 173938.pngExplanation

* **Starting the Server:** The code begins by importing the necessary modules and initializing a Flask application. It then sets up the API key and base URL for OpenWeatherMap.
* **Home Route:** The root route (/) is responsible for rendering an HTML page called index.html. This is the entry point for users.
* **Weather Route:** The /weather endpoint listens for GET requests and expects a city name as a query parameter. When a city is provided, the app sends a request to OpenWeatherMap, retrieves the weather data, and returns the relevant details (city name, temperature, weather description, and humidity) in JSON format.
* **Error Handling:** If a city is not provided or if the external API call fails, the server responds with an error message and an appropriate HTTP status code.
* **Running the App:** Finally, the app is set to run in debug mode so you can easily track any issues during development.

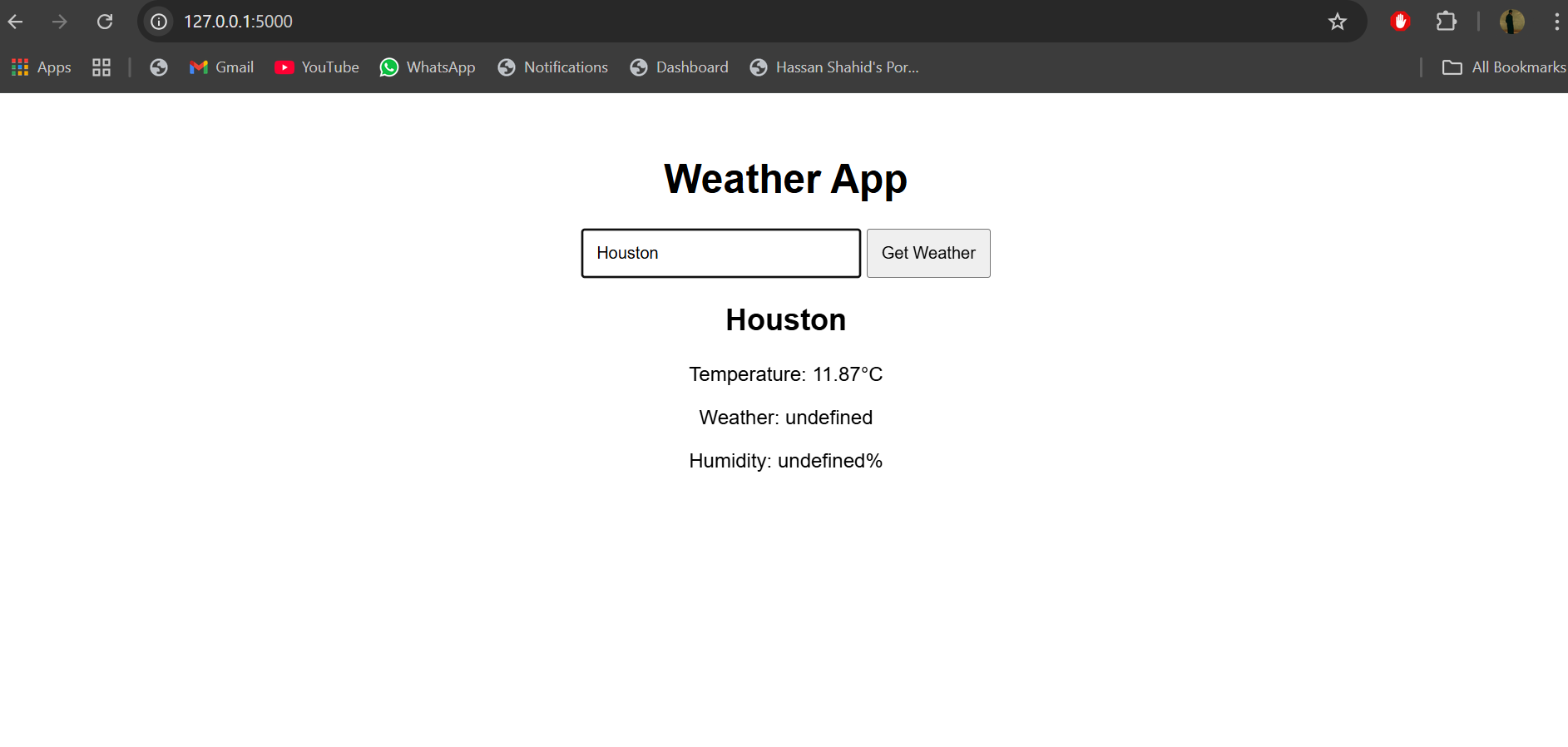
### Frontend (HTML, CSS, JavaScript)



### 

#### Explanation

* **Structure:** The HTML provides a clean layout with a header, an input field for entering the city name, a button to trigger the weather fetch, and a section where the weather information is displayed.
* **Styling:** Simple CSS is used to center the content, style the input, and provide spacing. This keeps the interface neat and user-friendly.
* **Interactivity:** The JavaScript function getWeather() reads the city entered by the user. It then sends an asynchronous request to the Flask backend using the Fetch API.
* **Dynamic Display:** Once the backend returns the data, JavaScript updates the webpage by injecting the weather details into the designated section. If there’s an error (for example, if the city isn’t found), it displays an error message instead.

**RESULTS**

## Conclusion:

This Flask Weather App is a straightforward yet powerful demonstration of how to build a full-stack web application. By combining a Python-based Flask backend with a dynamic front end, the project shows you how to:

* Set up a RESTful API to communicate with an external weather service.
* Render HTML pages using Flask's templating system.
* Handle user inputs and display live data using modern web technologies like JavaScript and Fetch API.

This project is not only a learning exercise but also a foundation you can build on. Future enhancements might include better error handling, more detailed weather reports, or even additional features such as forecasting. The experience gained here is invaluable for building more complex web applications in the future.