Weather Web Application Development Using Python and Flask

Overview

In this project, we will be developing a web application using Python and Flask that allows users to retrieve current weather data for any city of interest. The application will interact with the OpenWeatherMap API to fetch real-time weather information and display it to the user in a user-friendly manner.

Project Goals

Build a Flask Web Application: Utilize the Flask framework to create a web application with dynamic routes and HTML templates.

Integrate with OpenWeatherMap API: Implement functionality to fetch weather data from the OpenWeatherMap API based on user input.

Enhance User Experience: Design a simple and intuitive user interface for inputting city names and displaying weather information.

Best Practices: Follow best practices in development, including environment setup, version control.

Steps Performed

Step 1: Set Up Development Environment

1. Create a Virtual Environment
2. Install Required Packages
3. Create a Requirements File

Step 2: Configuration and Environment Setup

4. Create .gitignore File
5. Setup Environment Variables

Step 3: Project Structure

6.Organize Folders

Step 4: Retrieve weather data

7. Create weather.py File. Develop a Python script to interact with the OpenWeatherMap API and retrieve current weather data for a specified city.

Step 5: Build the Web Application

8. Create server.py File. Develop the server-side code using Flask to serve HTML templates and handle requests for weather data

Step 6: Version Control and Deployment

9. Upload to Git
10. Deploy the Web App

Implementation Details

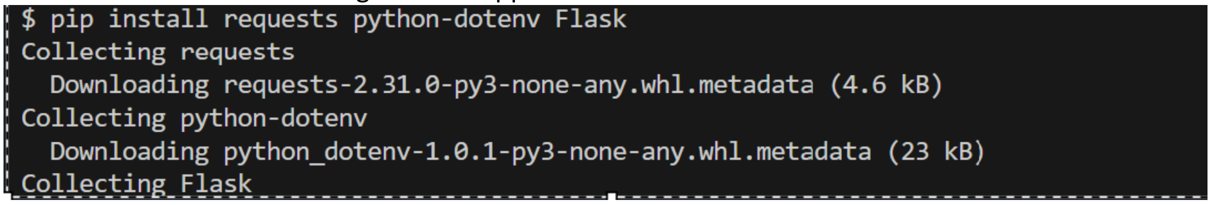
Step 1: Set Up Development Environment

1.Create a Virtual Environment

Set up a virtual environment in your preferred Integrated Development Environment (IDE) such as VS Code.

2.Install Required Packages

Install the necessary packages using pip. This includes python-dotenv for managing environment variables and Flask for building the web application.

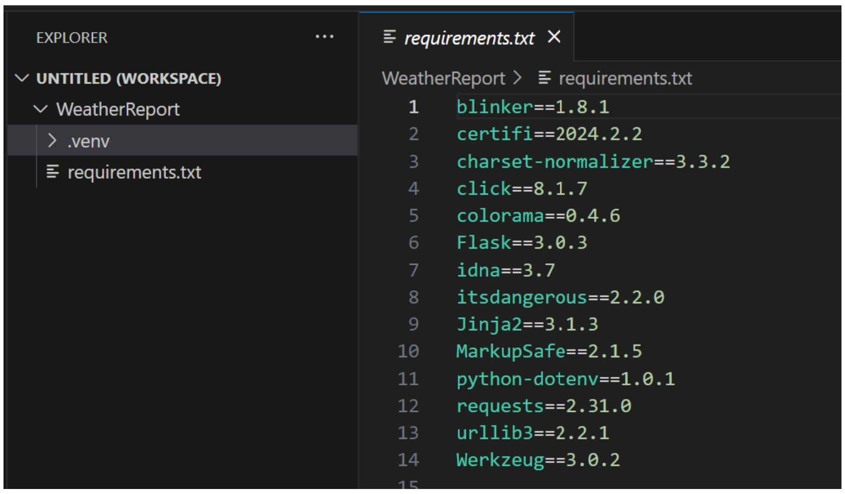


```
$ pip install requests python-dotenv Flask
Collecting requests
  Downloading requests-2.31.0-py3-none-any.whl.metadata (4.6 kB)
Collecting python-dotenv
  Downloading python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Collecting Flask
```

3.Create a Requirements File

Generate a requirements.txt file containing a list of all installed packages. This file ensures consistent dependencies across different environments.

$ pip freeze > requirements.txt

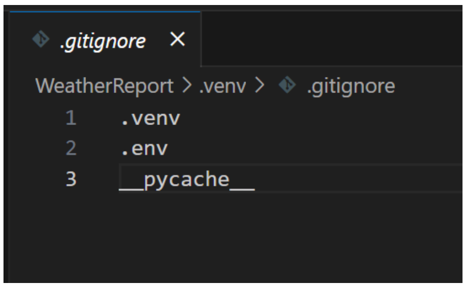


```
blinker==1.8.1
certifi==2024.2.2
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
Flask==3.0.3
idna==3.7
itsdangerous==2.2.0
Jinja2==3.1.3
MarkupSafe==2.1.5
python-dotenv==1.0.1
requests==2.31.0
urllib3==2.2.1
Werkzeug==3.0.2
```

Step 2: Configuration and Environment Setup

4.Create .gitignore File

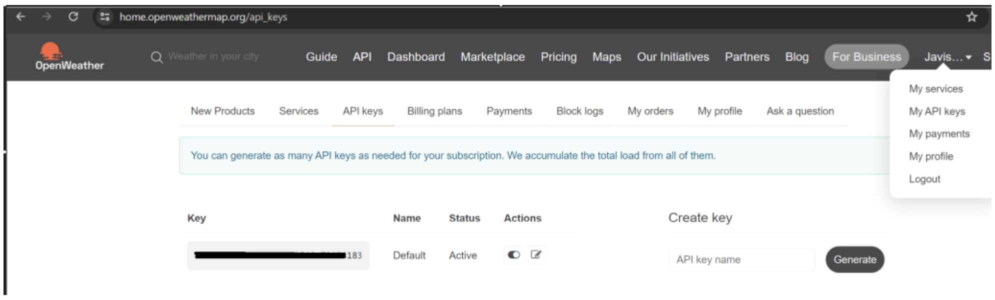
Create a .gitignore file to exclude sensitive files and directories such as .venv (virtual environment) and .env (environment variables) from version control.



```
.venv
.env
__pycache__
```

5.Set Up Environment Variables

Obtain an API key from OpenWeatherMap and store it securely as an environment variable named API_KEY. Ensure to add .env file to .gitignore for security purposes.



home.openweathermap.org/api_keys

You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
[redacted]183	Default	Active	

Create key

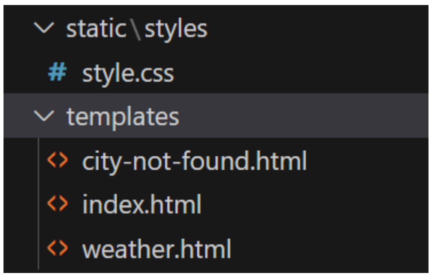
Step 3: Project Structure

6.Organize Folders

Create static and templates folders in your project directory.

The static folder will contain static files like CSS for styling.

The templates folder will store HTML templates for rendering dynamic content.



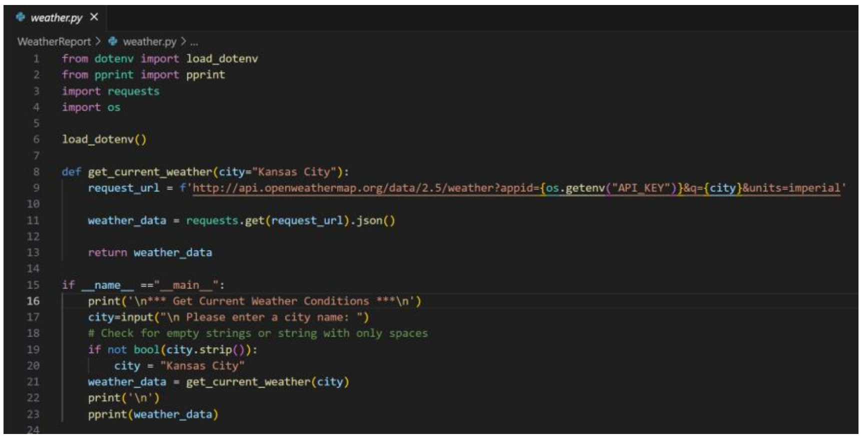
```
static\styles
  style.css
templates
  city-not-found.html
  index.html
  weather.html
```

Step 4: Retrieve Weather Data

7.Create weather.py File

Develop a Python script to interact with the OpenWeatherMap API and retrieve current weather data for a specified city.

The get_current_weather function sends a request to the OpenWeatherMap API and retrieves the current weather data for the specified city.The API key is stored securely using the python-dotenv library to load environment variables from a .env file.



weather.py

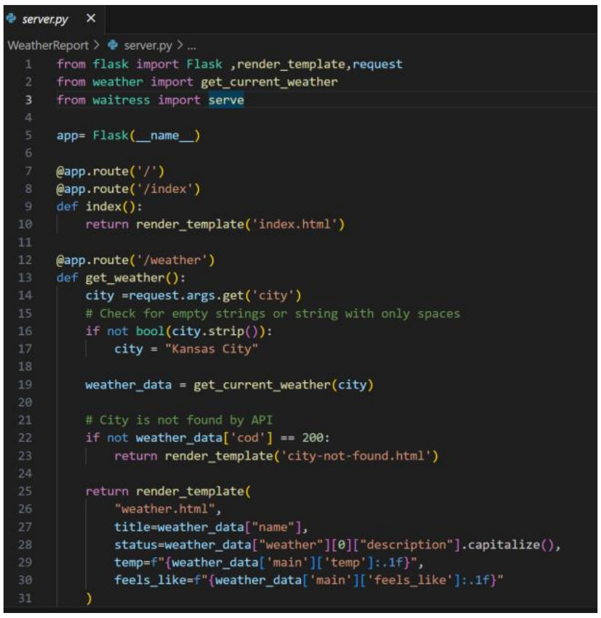
WeatherReport > weather.py > ...

```python
from dotenv import load_dotenv
from pprint import pprint
import requests
import os

load_dotenv()

def get_current_weather(city="Kansas City"):
    request_url = f'http://api.openweathermap.org/data/2.5/weather?appid={os.getenv("API_KEY")}&q={city}&units=imperial'

    weather_data = requests.get(request_url).json()

    return weather_data

if __name__ =="__main__":
    print('\n*** Get Current Weather Conditions ***\n')
    city=input("\n Please enter a city name: ")
    # Check for empty strings or string with only spaces
    if not bool(city.strip()):
        city = "Kansas City"
    weather_data = get_current_weather(city)
    print('\n')
    pprint(weather_data)
```

Step 5: Build the Web Application

8.Create server.py File

Develop the server-side code using Flask to serve HTML templates and handle requests for weather data.



server.py

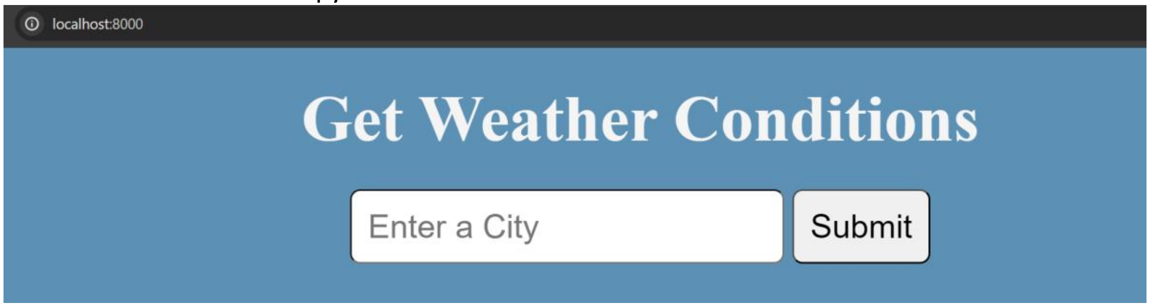
WeatherReport > server.py > ...

```python
from flask import Flask ,render_template,request
from weather import get_current_weather
from waitress import serve

app= Flask(__name__)

@app.route('/')
@app.route('/index')
def index():
    return render_template('index.html')

@app.route('/weather')
def get_weather():
    city =request.args.get('city')
    # Check for empty strings or string with only spaces
    if not bool(city.strip()):
        city = "Kansas City"

    weather_data = get_current_weather(city)

    # City is not found by API
    if not weather_data['cod'] == 200:
        return render_template('city-not-found.html')

    return render_template(
        "weather.html",
        title=weather_data["name"],
        status=weather_data["weather"][0]["description"].capitalize(),
        temp=f"{weather_data['main']['temp']:.1f}",
        feels_like=f"{weather_data['main']['feels_like']:.1f}"
    )
```

The server.py file contains the server-side code for a weather app developed using Flask. This file defines routes to serve HTML templates and handle requests to retrieve weather data for a specified city.

- The server.py file defines a Flask application (app) and specifies routes to serve different pages.
- The /index route renders the index.html template, which serves as the homepage of the weather app.
- The /weather route handles requests to retrieve weather data for a specific city. It retrieves the city name from the query parameters, calls the get_current_weather function to fetch weather data from the OpenWeatherMap API, and renders the weather.html template with the retrieved data.
- If the city is not found by the API (weather_data['cod'] != 200), the city-not-found.html template is rendered to inform the user.
- The server is started using the serve function from the waitress library, enabling it to handle multiple concurrent connections efficiently.

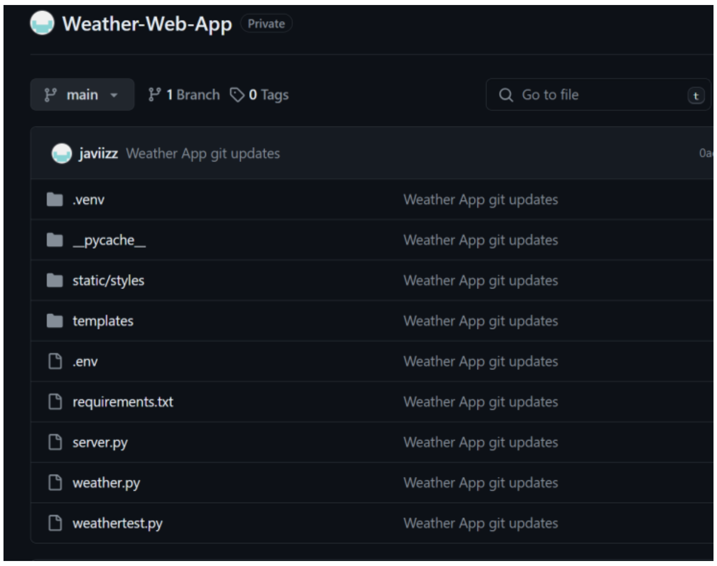
Execution result of server.py



Step 6: Version Control and Deployment

9.Upload to Git

Upload the project files to a version control system like Git to track changes.



Weather-Web-App Private

main 1 Branch 0 Tags Go to file

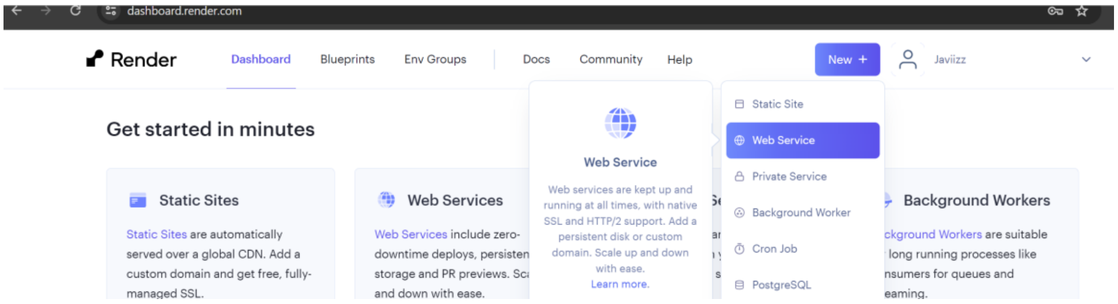
javiizz Weather App git updates

.venv	Weather App git updates
__pycache__	Weather App git updates
static/styles	Weather App git updates
templates	Weather App git updates
.env	Weather App git updates
requirements.txt	Weather App git updates
server.py	Weather App git updates
weather.py	Weather App git updates
weathertest.py	Weather App git updates

10.Deploy the Web App

Deploy the Flask web application to a hosting platform such as Render.com to make it accessible over the internet.

Create a new Web Service in render.com



dashboard.render.com

Render Dashboard Blueprints Env Groups Docs Community Help New + Javiizz

Get started in minutes

Static Sites

Static Sites are automatically served over a global CDN. Add a custom domain and get free, fully-managed SSL.

Web Services

Web Services include zero-downtime deploys, persisten storage and PR previews. Sc and down with ease.

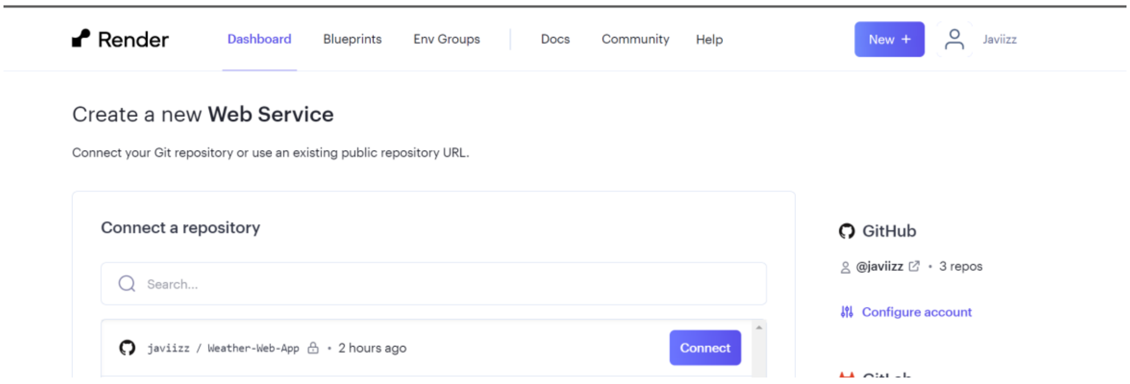
Web Service

Web services are kept up and running at all times, with native SSL and HTTP/2 support. Add a persistent disk or custom domain. Scale up and down with ease. Learn more.

Background Workers

ckground Workers are suitable long running processes like nsumers for queues and eaming.

Connect with Git and select the repository.



Render Dashboard Blueprints Env Groups Docs Community Help New + Javiizz

Create a new Web Service

Connect your Git repository or use an existing public repository URL.

Connect a repository

Search...

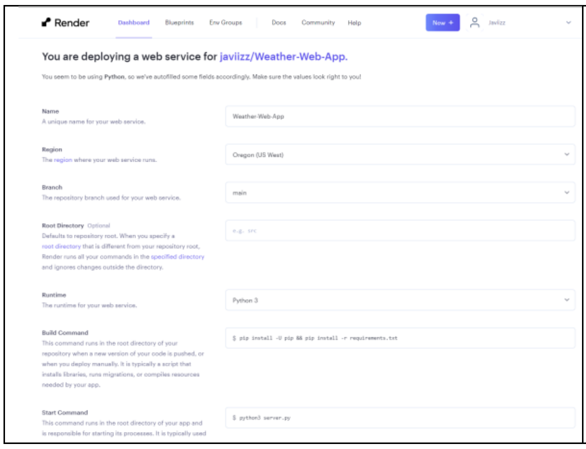
javiizz / Weather-Web-App • 2 hours ago Connect

GitHub

@javiizz • 3 repos

Configure account

Enter the details for hosting the app and click on Deploy



You are deploying a web service for javiizz/Weather-Web-App.

You seem to be using **Python**, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name
A unique name for your web service.
Weather-Web-App

Region
The region where your web service runs.
Oregon (US West)

Branch
The repository branch used for your web service.
main

Root Directory Optional
Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.
e.g. src

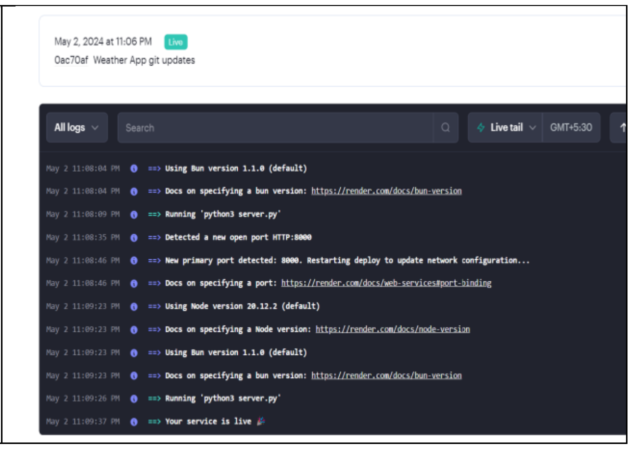
Runtime
The runtime for your web service.
Python 3

Build Command
This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.

```
$ pip install -U pip && pip install -r requirements.txt
```

Start Command
This command runs in the root directory of your app and is responsible for starting its processes. It is typically used

```
$ python3 server.py
```

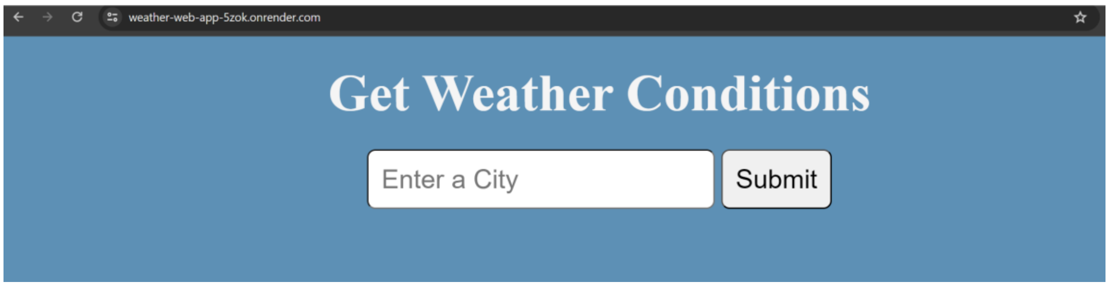


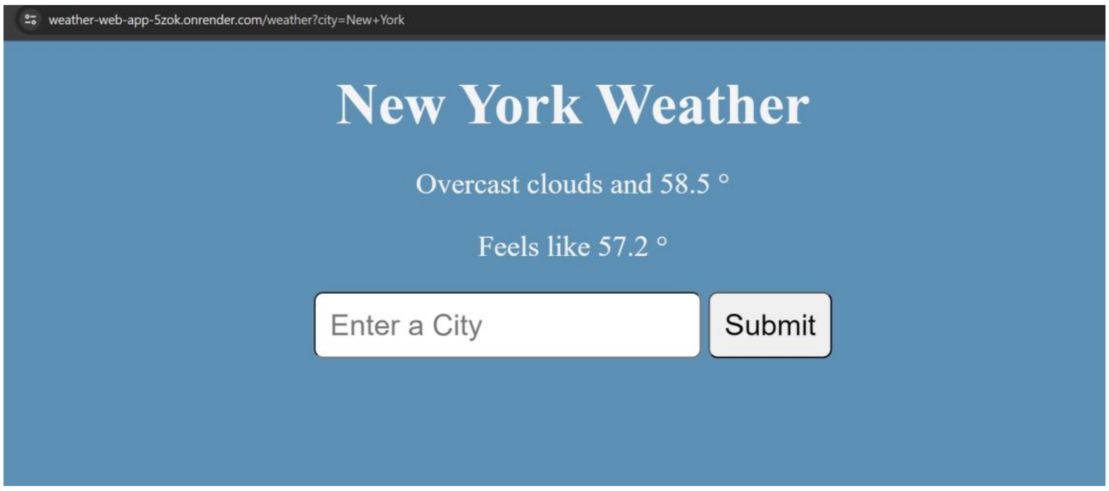
May 2, 2024 at 11:06 PM
0ac70af Weather App git updates

```
May 2 11:08:04 PM  ==> Using Bun version 1.1.0 (default)
May 2 11:08:04 PM  ==> Docs on specifying a bun version: https://render.com/docs/bun-version
May 2 11:08:09 PM  ==> Running 'python3 server.py'
May 2 11:08:35 PM  ==> Detected a new open port HTTP:8000
May 2 11:08:46 PM  ==> New primary port detected: 8000. Restarting deploy to update network configuration...
May 2 11:08:46 PM  ==> Docs on specifying a port: https://render.com/docs/web-services#port-binding
May 2 11:09:23 PM  ==> Using Node version 20.12.2 (default)
May 2 11:09:23 PM  ==> Docs on specifying a Node version: https://render.com/docs/node-version
May 2 11:09:23 PM  ==> Using Bun version 1.1.0 (default)
May 2 11:09:23 PM  ==> Docs on specifying a bun version: https://render.com/docs/bun-version
May 2 11:09:26 PM  ==> Running 'python3 server.py'
May 2 11:09:37 PM  ==> Your service is live 🎉
```

Deployed Weather web app

https://weather-web-app-5zok.onrender.com




weather-web-app-5zok.onrender.com/weather?city=New+York
New York Weather
Overcast clouds and 58.5 °
Feels like 57.2 °
Enter a City
Submit

Conclusion

Throughout this project, we've learned valuable concepts and techniques in web development, including working with APIs, building dynamic web applications with Flask, and deploying applications to hosting platforms.By leveraging the Flask framework and integrating with the OpenWeatherMap API, we've created a user-friendly application that allows users to easily access current weather data for any city.The project demonstrates the power and flexibility of Python for web development, as well as the simplicity and elegance of Flask as a web framework