**A purple circle with dots and lines

Description automatically generated**

**NAME:**

**Basit Abbas**

**ROLL NUMBER:**

**SU92-BSDSM-F23- 013**

**PROGRAM:**

**BS DATA SCIENCE**

**SUMBITTED TO:**

**SIR RASIKH**

***TASK : 08***

**Weather App Report**

**Overview**

The Weather App is a simple web application built using Flask and HTML/CSS that allows users to retrieve weather information for a specified city. It fetches real-time weather data using the OpenWeatherMap API and displays it in a user-friendly interface.

**Components**

**1. Backend (app.py)**

* Uses Flask to create a web server.
* Fetches weather data from OpenWeatherMap API.
* Processes the API response and extracts relevant weather details such as temperature, description, and an icon representing the weather.
* Renders the weather data onto the frontend using the Flask render\_template function.

**2. Frontend (index.html)**

* Provides a simple and clean user interface.
* Includes an input field for users to enter a city name and a button to fetch weather data.
* Displays weather information such as city name, temperature (in Celsius), weather description, and an icon.
* Includes basic CSS styling for a better user experience.

**Key Features**

* **Real-time Weather Data**: Fetches live weather updates from OpenWeatherMap API.
* **User-Friendly Interface**: Simple form for city input and a visually appealing weather display.
* **Error Handling**: Displays an error message if an invalid city is entered.
* **Responsive Design**: The page layout adjusts well to different screen sizes.

**Issues and Recommendations**

1. **Incorrect Template Directory Name**
   * Issue: The app.py file sets template\_folder="temeplate", which seems to be a typo.
   * Fix: Change it to template\_folder="templates" to ensure Flask correctly finds index.html.
2. **API Key Exposure**
   * Issue: The API key is hardcoded in app.py, which is a security risk.
   * Fix: Store the API key in an environment variable instead.
3. **Lack of User Input Validation**
   * Issue: The app does not validate user input before sending the API request.
   * Fix: Add basic validation to check for empty or invalid input before making a request.
4. **No Caching Mechanism**
   * Issue: Every request fetches fresh data, increasing API usage.
   * Fix: Implement a caching mechanism to reduce redundant API calls.
5. **Limited Error Handling**
   * Issue: The app only checks for a 200 response status but does not handle cases like API rate limits or network errors.
   * Fix: Improve error handling by checking for various response codes and displaying appropriate messages.

**Conclusion**

This Weather App is a well-structured project demonstrating Flask's capabilities for fetching and displaying real-time data. Implementing the suggested improvements will enhance its security, performance, and user experience.