**Weather Forecast Website using Flask and OpenWeather API**

**Task Given:**

Fetch data from a specific website using an API. The retrieved data, provided in JSON format, should be stored in a structured format such as CSV.

**Task Description**

The Weather Forecast Website is a web-based application developed using Python’s Flask framework. It integrates the OpenWeather API to fetch real-time weather information for a given city. The app displays weather data such as temperature, humidity, wind speed, and weather conditions in a user-friendly format using HTML and CSS. It also supports dynamic display of weather-related images based on conditions like "Few Clouds", "Rain", etc.

|  |  |
| --- | --- |
| **Technology Used** | **Purpose** |
| Python Core | Backend logic |
| Flask | Web framework |
| OpenWeather API | Real-time weather data |
| HTML & CSS | Frontend UI |
| CSV | User input history |

**How the Application Works**

**1. User Interface (Frontend)**

* The app starts at the / route.
* The user enters a city name in a form.
* On submission (POST method), the backend processes this city name.

**2. Weather API Integration**

* The app constructs a request to **OpenWeatherMap** API using the given city:

**url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API\_KEY}&units=metric"**

* The requests library sends the API call and gets the response in JSON format.
* Example API response includes:
  + weather[0].main: condition (e.g., Clear, Rain)
  + weather[0].description: more details (e.g., light rain)
  + main.temp: temperature in Celsius

**3. Data Extraction & Formatting**

* The response is parsed to extract:
  + city name
  + temperature
  + description
  + condition (e.g., Clear, Clouds)
* A corresponding image URL is selected from a predefined dictionary:

**image\_url = weather\_images.get(condition, default\_image)**

**4. Rendering Results (Backend to Frontend)**

* The weather\_data dictionary is passed to the index.html Jinja2 template:

python

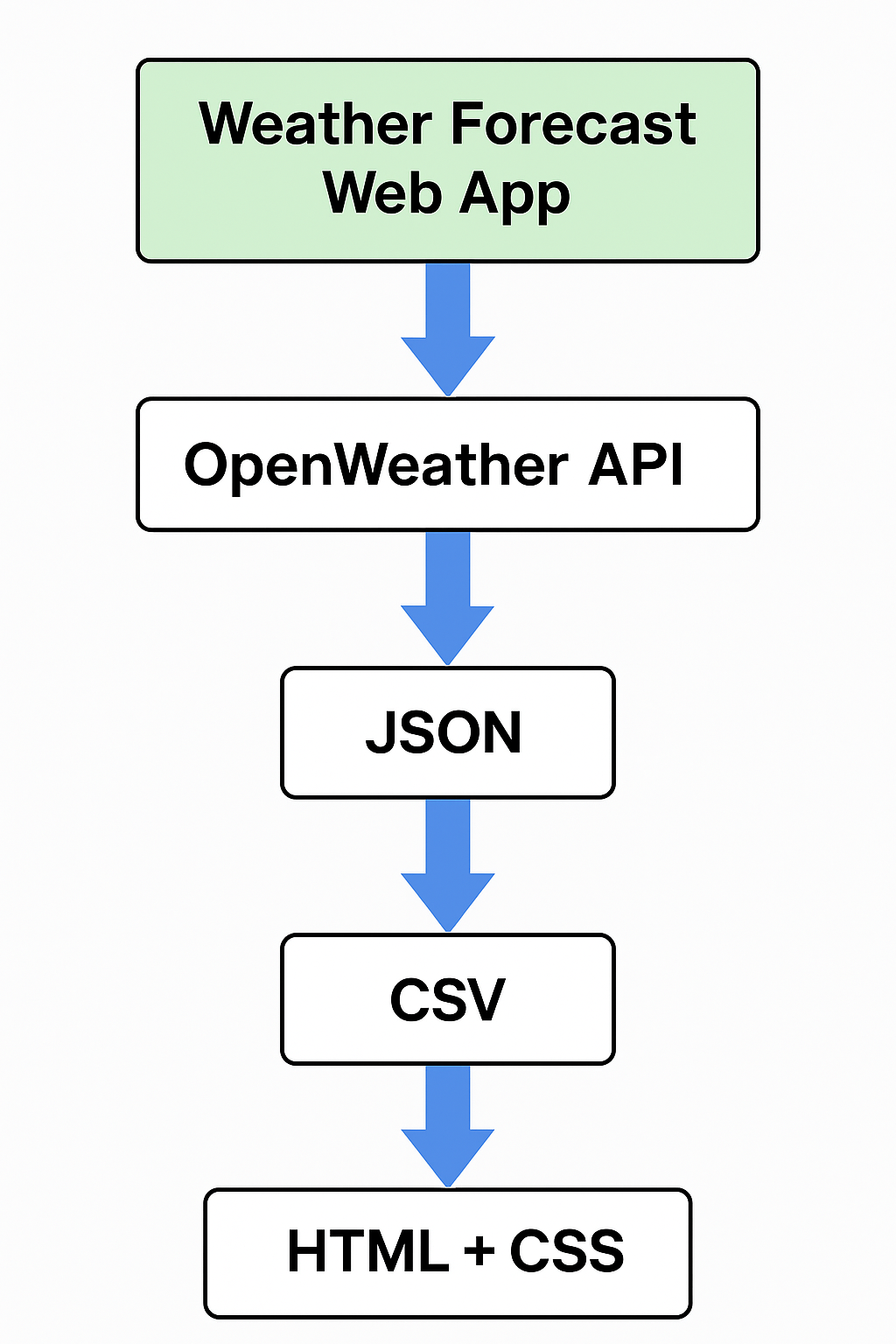
Copy code

return render\_template("index.html", weather\_data=weather\_data)

* This template displays:
  + The temperature
  + The weather condition description
  + The image icon (based on condition)

**How the API Integration Was Done (Simplified View)**

| **Step** | **What Happens** |
| --- | --- |
| User Input | User types city name in form on web page |
| Send API Request | Flask app sends GET request to OpenWeatherMap API with API key + city |
| Get Response | API returns JSON containing weather details |
| Process JSON | Flask extracts condition, temperature, and description |
| Select Image | Based on condition, app picks a matching image URL from weather\_images |
| Display on Webpage | Flask sends data to template and shows result with icon and weather details |

**Workflow:**

**Additional Notes**

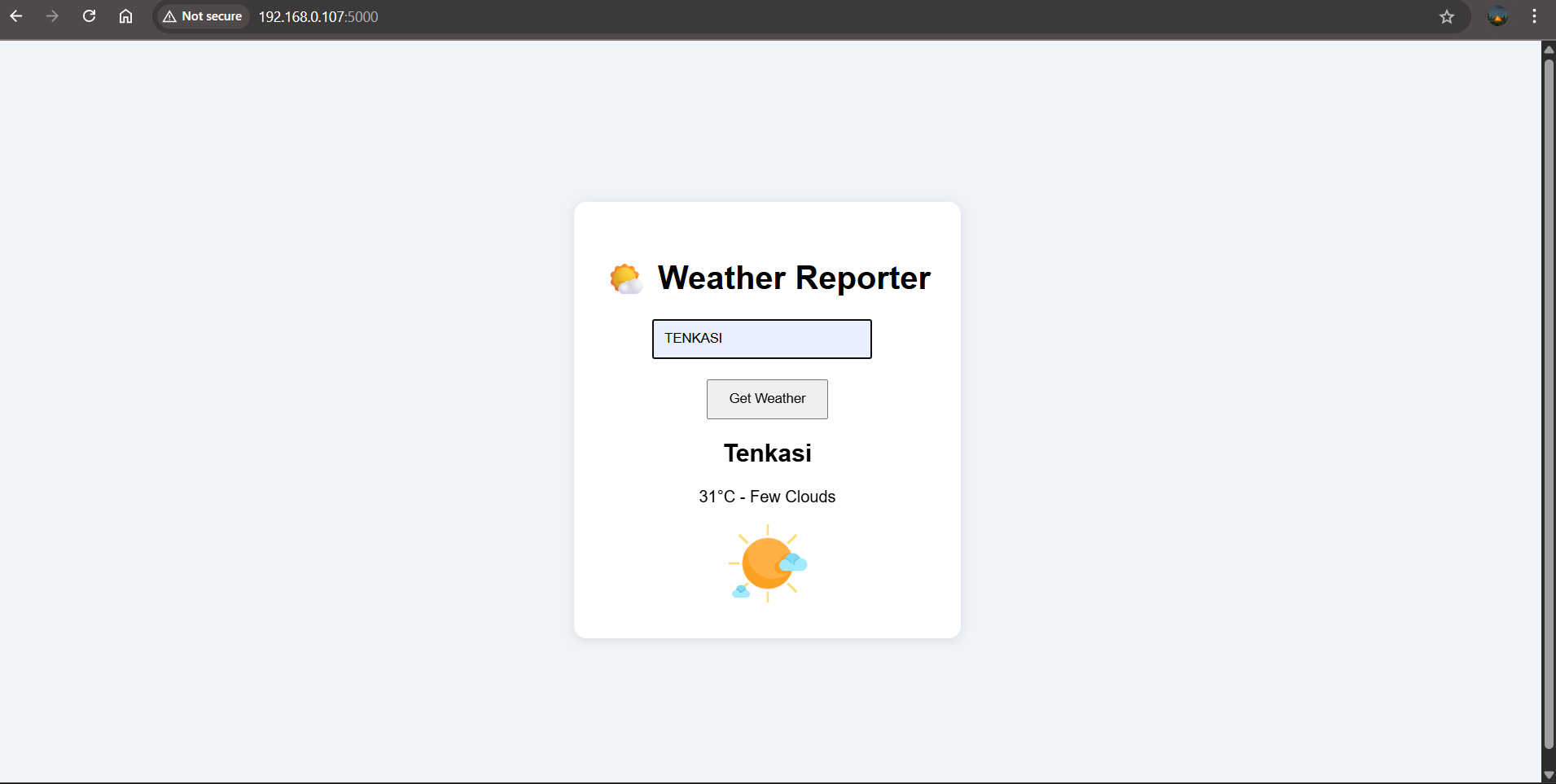
* **Security**: API key is stored directly in code; for production, use environment variables.
* **Cross-Platform**: os.chmod handles file permissions on non-Windows systems.
* **CSV Logging**: You could log weather queries into **weather\_data.csv** if desired.
* **Fallback Image**: A base64 image is used if condition doesn't match known types.

**Reference API Link:**

* <https://home.openweathermap.org/api_keys>

**Reference Snaps**

* Data is fetched using API and shows the current weather of the given city.



* The application retrieves real-time weather data using an API and displays the current conditions for the specified city.

