

***PREDICTIVE ANALYSIS LAB PROJECT***

***REPORT***

|  |  |
| --- | --- |
| NAME | SAPID |
| 1. SIDDHARTH VERMA | 500107403 |
| 2. ROHAN CHANDWASAR | 500105552 |

WEATHER PREDICTION MODEL

### **Introduction**

The project is a weather forecasting web application built with Python using the **Flask** web framework. It leverages the **OpenWeatherMap API** to provide real-time weather updates and a five-day forecast for any city entered by the user. The application also includes error handling for invalid inputs and ensures a user-friendly interface.

### **Objective**

The primary goal of this project is to:

1. Display current weather details such as temperature, weather conditions, wind speed, and daily high/low temperatures for a specific city.
2. Provide a five-day weather forecast with key metrics for each day.
3. Ensure error handling by redirecting to an error page when invalid or unrecognized city names are entered.

### **Data Sources**

* **OpenWeatherMap API**
  + Current weather data endpoint: <https://api.openweathermap.org/data/2.5/weather>
  + Forecast data endpoint: <https://api.openweathermap.org/data/2.5/forecast>
  + Geocoding API for city-to-coordinates translation: <http://api.openweathermap.org/geo/1.0/direct>

### **Project Flow**

#### **1. Home Page**

* Users are presented with a form to search for a city's weather.
* Upon entering a city name and submitting the form, the app routes the request to /get\_weather/<city>.

#### **2. City Weather Page**

* **Input Processing:**
  + The city name is formatted for consistency.
  + Geocoding API translates the city name into latitude and longitude coordinates.
* **Weather Data Retrieval:**
  + Current weather is fetched using the latitude and longitude from the Geocoding API.
  + A five-day forecast is retrieved, with specific data points (e.g., temperatures at midday) parsed for display.
* **Error Handling:**
  + If the city name is invalid or the API response lacks data, the user is redirected to the /error page.
* **Output Display:**
  + Weather data for the current day and a summary for the next five days is rendered on a dedicated page using HTML templates.

#### **3. Error Page**

* Displays a user-friendly message indicating an invalid city input.
* Encourages users to return to the home page to try again.

### **Key Features**

#### **Data Collection**

* Geocoding API ensures accurate data by converting city names to geographical coordinates.
* The Weather API provides:
  + Current temperature, conditions, and wind speed.
  + High and low temperatures for the day.
  + Five-day forecasts with daily summaries.

#### **Data Processing**

* **Formatting:** City names are capitalized using Python's string.capwords.
* **Date Handling:** datetime is used to format and display the current date and calculate the next four weekdays.
* **Error Management:** Redirects to an error page if no valid data is returned by the Geocoding API.

### **User Interface**

* **Home Page:** Features a search bar for entering city names.
* **City Weather Page:**
  + Displays current weather with visual clarity.
  + Includes a five-day forecast with temperatures and weather conditions summarized by day.
* **Error Page:** Informs users of invalid inputs in a user-friendly manner.

### **Tools and Technologies Used**

* **Programming Language:** Python
* **Framework:** Flask (for routing and rendering HTML templates)
* **APIs:** OpenWeatherMap for weather data
* **Libraries:**
  + requests for API integration.
  + datetime for date calculations.
  + dotenv for environment variable management.
* **Templates:** HTML for front-end rendering.

### **Strengths**

1. **Real-Time Data:** Utilizes live API data for accurate and up-to-date weather information.
2. **Error Handling:** Handles invalid inputs gracefully, enhancing user experience.
3. **Scalable Design:** The codebase can be easily extended to include additional features such as hourly forecasts or alerts.

### **Areas for Improvement**

1. **Enhance UI:** Add CSS and JavaScript to improve user interactivity and aesthetics.
2. **Cache Responses:** Implement caching to reduce API calls and improve performance for repeated city searches.
3. **Mobile Responsiveness:** Optimize templates for better display on mobile devices.

### **Conclusion**

This weather forecasting application effectively integrates API data with a user-friendly interface to deliver accurate and timely weather updates. Future iterations can focus on visual enhancements and additional features like weather alerts or localization for a global audience.