Weather App

Project report

**Weather Application Report**

**Overview**

The WeatherApp is a Python program that interacts with the OpenWeatherMap API to provide weather information for user-specified cities. It features robust error handling, a modular design, and a user-friendly interface. The application demonstrates the effective use of APIs, exception handling, and object-oriented programming principles.

**Features**

1. **Weather Data Retrieval**:
   * Fetches weather details such as temperature, humidity, wind speed, and pressure using OpenWeatherMap's API.
   * Supports dynamic city-based input from users.
2. **Error Handling**:
   * Includes comprehensive error-handling mechanisms for common issues like network failures, API timeouts, or invalid city names.
3. **Data Parsing**:
   * Extracts relevant weather data from the API response and organizes it into a user-friendly format.
4. **User Interaction**:
   * Allows users to input city names and view weather details interactively.
   * Provides an option to exit the application gracefully.
5. **Formatted Output**:
   * Displays weather information in a clean and easy-to-read format, including:
     + Current temperature
     + Feels-like temperature
     + Weather description
     + Humidity percentage
     + Wind speed
     + Atmospheric pressure

**Implementation Details**

1. **API Integration**:
   * Utilizes the requests library to interact with the OpenWeatherMap API.
   * The base\_url is set to "http://api.openweathermap.org/data/2.5/weather".
   * Requests include city name, API key, and metric units for temperature.
2. **Class Structure**:
   * The app is encapsulated in a WeatherApp class, promoting modularity and reusability.
   * Methods include:
     + \_\_init\_\_: Initializes the app with the API key and base URL.
     + get\_weather\_data: Fetches raw weather data from the API.
     + parse\_weather\_data: Extracts and formats weather information.
     + display\_weather: Outputs formatted weather data to the console.
     + run: Manages user input and app control flow.
3. **Error Handling**:
   * Catches and handles specific exceptions like:
     + HTTPError: Issues with the API response (e.g., invalid city).
     + ConnectionError: Network issues.
     + Timeout: API response delays.
   * Generic exceptions are also managed to avoid unexpected crashes.

**Example**

1. **User Input**:
   * The user enters a city name (e.g., *London*).
   * If the city name is invalid, an error message is displayed.
2. **Output**:
   * For a valid input, the app displays:
   * ------------------------------
   * Weather in London:
   * Temperature: 15°C
   * Feels Like: 13°C
   * Description: Clear sky
   * Humidity: 60%
   * Wind Speed: 5 m/s
   * Pressure: 1015 hPa
   * ------------------------------
   * For invalid input (e.g., non-existent city), it shows:
   * Failed to parse data. The city may not exist or the data may be incomplete.

**Key Advantages**

1. **User-Friendly**:
   * Interactive prompts and clear error messages make the app accessible to non-technical users.
2. **Robustness**:
   * Extensive error handling ensures the app runs smoothly, even with unexpected inputs or API failures.
3. **Modularity**:
   * Separate methods for data retrieval, parsing, and display enhance maintainability and readability.