

**Project Report: React Weather Application**

**Project Title:** Simple Weather Forecast Web Application

**Name:** BIDYACHANDRA MAIBRAM

**Register No.:** 23BTCE203

**Date:** September 8, 2025

**1. Introduction**

This report details the development of a web-based weather application. The primary objective of this project was to create a clean, user-friendly interface where users can fetch and display real-time weather data for any city in the world.

The application is built as a single-page application (SPA) using modern web technologies, focusing on a responsive design that works seamlessly across desktops, tablets, and mobile devices. It leverages the OpenWeatherMap API to source its meteorological data.

**2. Features and Functionality**

The application is designed with simplicity and usability in mind, offering the following core features:

* **City-Based Weather Search:** Users can type the name of any city into a search bar.
* **Real-Time Data Display:** Upon a successful search, the application displays current weather conditions, including:
  + Temperature (in Celsius)
  + "Feels Like" temperature
  + Weather condition (e.g., "clear sky," "light rain")
  + Humidity percentage
  + Wind speed (in m/s)
  + Atmospheric pressure (in hPa)
* **Dynamic Weather Icons:** The UI displays a relevant icon corresponding to the current weather conditions (e.g., sun, clouds, rain).
* **Loading and Error States:** The interface provides clear feedback to the user. It shows a "Searching..." message while fetching data and displays a user-friendly error message (e.g., "City not found") if the API call fails or the city name is invalid.
* **Responsive Design:** The layout automatically adjusts to fit the screen size of the user's device.

**3. Technical Overview**

The project was developed using a modern front-end stack, ensuring a robust and maintainable codebase.

* **React.js:** A popular JavaScript library for building user interfaces. React's component-based architecture was used to structure the application, separating the UI into logical, reusable pieces (App, WeatherCard). State management was handled using React Hooks (useState).
* **Tailwind CSS:** A utility-first CSS framework used for styling the application. It allowed for rapid development of a modern, responsive, and visually appealing design directly within the HTML structure without writing custom CSS files.
* **OpenWeatherMap API:** A third-party service used as the source for all weather data. The application makes HTTP GET requests to this API to fetch data based on the user's city query.
* **Vite:** A modern front-end build tool that provides a faster and leaner development experience for modern web projects.

**4. Code Explanation**

The entire application is encapsulated within a single file, src/App.jsx.

**4.1. State Management (useState)**

Four state variables were used to manage the application's lifecycle:

* city: Stores the current value of the user's input in the search field.
* weatherData: Holds the JSON object returned from the API upon a successful fetch.
* message: Stores any error or informational messages to be displayed to the user.
* loading: A boolean state to track when an API call is in progress, allowing the UI to show a loading indicator.

**4.2. Data Fetching (fetchWeather function)**

This asynchronous function is triggered when the user submits the form.

1. It first prevents the default form submission behavior.
2. It validates that the city input is not empty.
3. It sets the loading state to true and clears any previous data or messages.
4. It constructs the API URL with the user's city and the unique API key.
5. It uses the browser's fetch() API to make the request.
   * **Success:** If the response is successful, it parses the JSON data and updates the weatherData state.
   * **Failure:** If the response indicates an error (e.g., a 404 Not Found), it throws an error.
6. The .catch() block handles any network errors or errors thrown from the response, updating the message state accordingly.
7. The .finally() block ensures the loading state is set back to false after the request is complete, regardless of the outcome.

**4.3. UI Components**

* **App Component:** The main component that contains the primary layout, input form, and logic.
* **WeatherCard Component:** A sub-component responsible for displaying the formatted weather data. It receives the weatherData as a prop and renders the temperature, conditions, icons, and other details. This separation of concerns makes the code cleaner and easier to manage.

**5. Challenges and Resolutions**

The most significant challenge during this project was related to the **OpenWeatherMap API Key**.

* **Problem:** After generating a new API key and integrating it into the application, the API consistently returned a "City not found" error, even for valid city names like "Mumbai" or "London."
* **Diagnosis:**
  1. The application code was reviewed and found to be logically correct.
  2. The API key was tested directly in a browser by calling the API URL. This test also failed, returning a 401 Unauthorized error, which confirmed the problem was with the key itself and not the application code.
  3. Further investigation revealed that new API keys from OpenWeatherMap can take several minutes to a few hours to become active on their servers.
  4. It was also identified that the user needed to **verify their email address** with OpenWeatherMap to fully activate the account and its associated API keys.
* **Resolution:** The solution involved generating a fresh API key, ensuring the account's email was verified, and waiting for the activation period to pass. This resolved the authorization issue and allowed the application to fetch data successfully.

**6. Conclusion**

This project successfully achieved its goal of creating a functional and visually appealing weather application. It demonstrates the effective use of React for state management and UI composition, Tailwind CSS for modern styling, and the integration of a third-party REST API.

The primary takeaway from the development process was the importance of thorough debugging, especially when dealing with external services. By isolating the problem to the API key activation, we were able to solve an issue that initially seemed like a code-related bug. The final application is a robust, single-page tool that provides a valuable service to the user.