**Weather App using ReactJs :**

Notes :

*@layer* base{

*html* {

    font-family: *"Poppins",* sans-serif;  }

}

Explanation :

The code snippet you provided is using Tailwind CSS’s @layer directive to add custom base styles. Here’s a breakdown of what it does:

**Explanation**

* @layer base: This directive allows you to add custom styles to Tailwind’s base layer. The base layer is where you define global styles that apply to the entire document.
* html { font-family: "Poppins", sans-serif; }: This sets the font family for the entire HTML document to “Poppins”, with a fallback to sans-serif.

The **base layer** in Tailwind CSS is where you define global styles that apply to the entire document. It’s one of the three layers in Tailwind’s CSS architecture, alongside the **components** and **utilities** layers. Here’s a quick overview of each:

**Base Layer**

* **Purpose**: To set global styles that affect the entire document.
* **Usage**: Typically used for resetting or normalizing styles, setting default typography, and other global settings.
* **Example**: Setting a global font family, as you did with font-family: "Poppins", sans-serif;.

**Components Layer**

* **Purpose**: To define reusable component styles.
* **Usage**: Used for styling specific components like buttons, cards, modals, etc.
* **Example**: Creating a custom button style that can be reused across your project.

**Utilities Layer**

* **Purpose**: To provide low-level utility classes that can be composed to build custom designs.
* **Usage**: Used for applying individual styles directly in your HTML.

1. <*div* className*=*"mx-auto max-w-screen-lg mt-4 py-5 px-32 bg-gradient-to-br shadow-xl shadow-gray-400 from-cyan-600 to to-blue-700">

Explanation of this is :

Let’s break down the Tailwind CSS classes applied to your <div>:

mx-auto: Centers the <div> horizontally by setting the left and right margins to auto.

max-w-screen-lg: Sets the maximum width of the <div> to the large breakpoint of the screen size.

mt-4: Adds a margin-top of 1rem (16px).

py-5: Adds padding of 1.25rem (20px) on the top and bottom.

px-32: Adds padding of 8rem (128px) on the left and right.

bg-gradient-to-br: Creates a background gradient that goes from the top-left to the bottom-right.

from-cyan-600: Sets the starting color of the gradient to cyan with a shade of 600.

to-blue-700: Sets the ending color of the gradient to blue with a shade of 700.

shadow-xl: Applies an extra-large box shadow.

shadow-gray-400: Sets the color of the shadow to gray with a shade of 400.

These classes combine to create a centered, responsive <div> with a gradient background, padding, and a shadow effect.

1. JAVASCRIPT file content :

const API\_KEY = "17bd6fc1fd9aa13fdb6b5d17b261ded9";

const BASE\_URL = "https://api.openweathermap.org/data/3.0/";

const getWeatherData =(infoType , searchParams)=> { const url = new URL(BASE\_URL + infoType);

url.search = new URLSearchParams({...searchParams,appid:API\_KEY});

return fetch(url) .then((res)=>res.json()) .then((data)=>data); };

export default getWeatherData;

**1. Setting Up Your API Key and Base URL**

**JavaScript**

const API\_KEY = "17bd6fc1fd9aa13fdb6b5d17b261ded9";

const BASE\_URL = "https://api.openweathermap.org/data/3.0/";

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* **API\_KEY**: Think of this as your personal password to access the weather data from OpenWeatherMap. It tells the service that you are allowed to use their data.
* **BASE\_URL**: This is the starting point for all your requests to the weather service. It’s like the main address of a website.

**2. Creating the Function to Get Weather Data**

**JavaScript**

const getWeatherData = (infoType, searchParams) => {

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* **getWeatherData**: This is a function that will fetch weather data for you. It needs two pieces of information:
* + **infoType: What kind of weather data you want (e.g., current weather, forecast).**
  + **searchParams: Details about what you’re looking for (e.g., city name, coordinates).**

**3. Building the URL**

**JavaScript**

**const url = new URL(BASE\_URL + infoType);**

**url.search = new URLSearchParams({ ...searchParams, appid: API\_KEY });**

**AI-generated code. Review and use carefully.**[**More info on FAQ**](https://www.bing.com/new#faq)**.**

* **new URL(BASE\_URL + infoType): This creates a full web address by adding the type of information you want to the base URL. For example, if you want current weather data, it might look like https://api.openweathermap.org/data/3.0/weather.**
* **url.search: This adds the specific details to your web address.**
  + **new URLSearchParams({…searchParams, appid: API\_KEY}): This turns your search parameters (like city name) and your API key into a format that can be added to the web address. It’s like adding details to a search query.**

**4. Fetching the Data**

**JavaScript**

return fetch(url)

.then((res) => res.json())

.then((data) => data);

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* **fetch(url)**: This sends a request to the web address you built. It’s like typing a URL into your browser and hitting enter.
* **.then((res) => res.json())**: When the weather service responds, this line converts the response into a format (JSON) that JavaScript can easily work with.
* **.then((data) => data)**: Finally, this line takes the converted data and makes it available for you to use in your application.

**5. Exporting the Function**

**JavaScript**

export default getWeatherData;

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* **export default getWeatherData**: This makes the getWeatherData function available for use in other parts of your application. It’s like sharing a tool with other parts of your project.

**Summary in Simple Words**

**Summary in Simple Words**

* **API\_KEY: Your password to access weather data.**
* **BASE\_URL: The main address of the weather service.**
* **getWeatherData: A function that fetches weather data.**
* **infoType: The type of weather data you want.**
* **searchParams: Details about what you’re looking for.**
* **URL: The full web address with all the details.**
* **fetch: Sends a request to get the data.**
* **.then: Processes the response to make it usable.**
* **export default: Shares the function with other parts of your project.**

**This function helps you get weather data by building a request, sending it to the weather service, and then processing the response so you can use it in your application. If you have any more questions or need further clarification, feel free to ask!**