About project :

Weather dashboard – fetching weather info using city name with the help of API (here open weather API)

Technologies used :

React

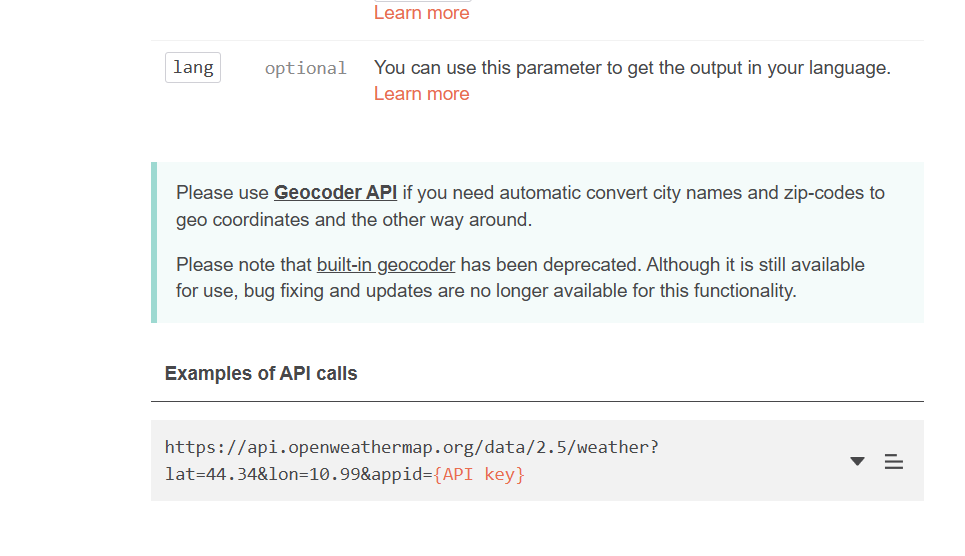
Api Requirements:

1. Go to link - <https://openweathermap.org/current>

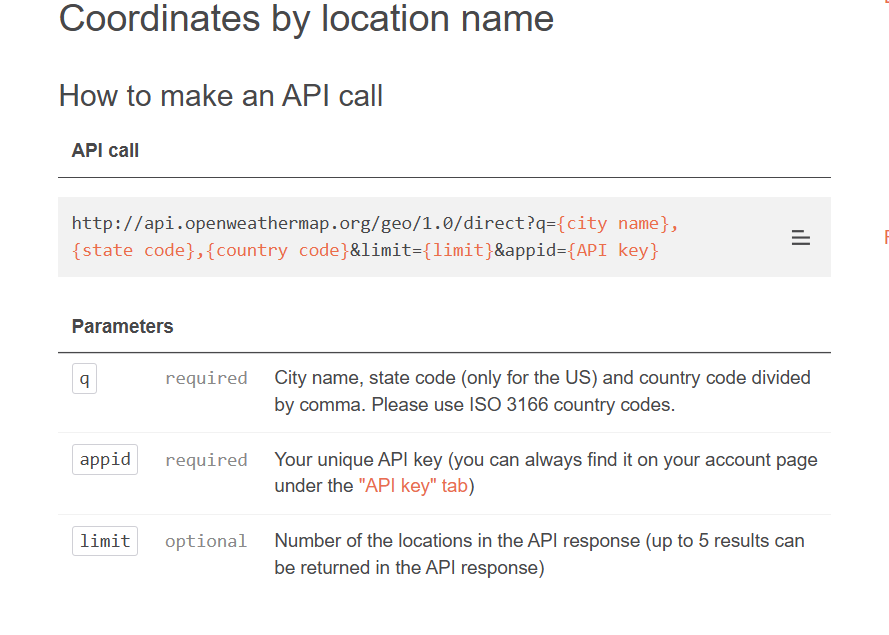
A screenshot of a computer

AI-generated content may be incorrect.

1. Use Geocoder : because we have to search using city name not longitude and latitude . Geocoder – converting addresses or place name into geographical coordinates(longitude and latitude)

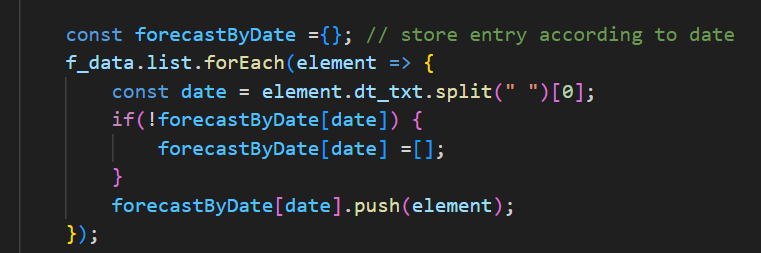






3. Click on “API key” tab to get API key  
 Use the above link of geocoder API

4. Forecast -API :  
 https://api.openweathermap.org/data/2.5/forecast

In this API data is in intervals of 3 hours so we will separate it on the basis of data .   
total 40 records are there : 5 days \* 8 (per day 8 interval)

Requirements:

1) **Install Font Awesome Packages**

Run the following command in your React project directory:

npm install --save @fortawesome/fontawesome-svg-core @fortawesome/free-solid-svg-icons @fortawesome/react-fontawesome

If you need **brand icons** (e.g., Facebook, Twitter), install:

npm install --save @fortawesome/free-brands-svg-icons

For text type : Roboto  
npm install @fontsource/roboto

2) Library :

* **A screenshot of a computer program

  AI-generated content may be incorrect.dayjs** : dayjs is a **lightweight JavaScript library** for handling and formatting dates and times.

Common dayjs Features:

installation : npm install dayjs

* **react-circular-progressbar :** The library used for the Circular Progress Bar(here used in humidity).  
    
  installation : npm install react-circular-progressbar
* **A screen shot of a computer

  AI-generated content may be incorrect.**explanation:   
   import "react-circular-progressbar/dist/styles.css";use to apply default style of progress bar

**value and text**

* value: Represents the progress percentage (0 to 100).
* text: Displays a label inside the progress bar (usually the percentage)

**path (Progress Line)**

* Defines the color and thickness of the progress stroke (the colored part that fills up).
* Common styles:

stroke: "#FFD700" → Sets the color to gold  
 strokeWidth: 8 → Makes the progress line thicker

**trail (Background Line)**

* Defines the color and thickness of the background stroke (the unfilled part).
* Common styles:

A black background with pink and green text

AI-generated content may be incorrect.

stroke: "#eee" → Sets the background color to light gray  
 strokeWidth: 8 → Matches the progress line width

**root (Outer Container)**

* Defines the size of the entire progress bar.
* Common styles:

A black background with pink and white text

AI-generated content may be incorrect.

width: 100px → Sets the diameter of the progress bar  
height: 100px → Maintains a square shape

A screenshot of a computer

AI-generated content may be incorrect.

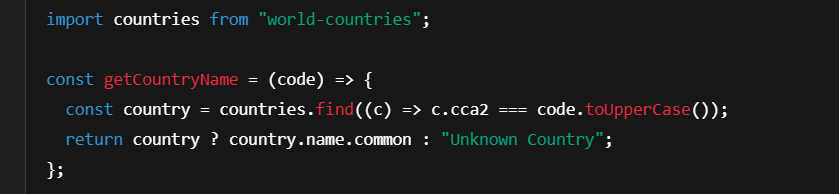
**NOTE :** Dependency Conflict (npm error Fix the upstream dependency conflict): This error typically happens when there are conflicting versions of libraries required by different packages in your project. You can resolve it by either:

* Using --legacy-peer-deps to install the dependencies without resolving peer dependency conflicts.
* Using --force to forcefully install the dependencies despite conflicts.

To install with --legacy-peer-deps (recommended for compatibility):

**npm install --legacy-peer-deps**

* **world-countries**: This package provides country data, including names and ISO codes.  
    
  Installation :   
  **npm install world-countries**



countries.find((c) => c.cca2 === code.toUpperCase());

* countries is an array containing country data
* .find(...) searches for the first country object where the cca2 property (ISO 2-letter country code) matches the provided code.
* code.toUpperCase() ensures the input code is in uppercase (e.g., "us" becomes "US").

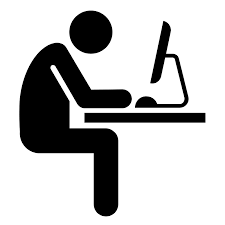
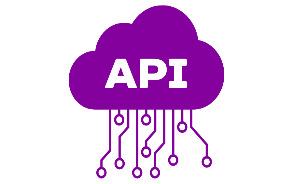
**What is an API** : Application programming interface



It is an interface or bridge that allow software to communicate(exchange data) with each other.

Here we are using **Web APIs** : enable communication over internet using HTTP or HTTPs protocol.

It can be understand as intermediate between use and server .

 request



response



User Server

APIs in response send data which is in JSON format.  
  
**Aync and Await:**



**Asyn** use to write asynchronous code into synchronous manner or it is a way to write asynchronous function (allows program to run other event without waiting for one to finish it).

A screen shot of a computer

AI-generated content may be incorrect.

Asynchronous

**Await :** wait for promise to be resolved before continuing further. Await keyword only be used within an async function .



**promise** : promise is the object represent completion or failure of asynchronous function . It has 3 state



Pending : hasn’t yet completed



Fulfilled : completed successfully

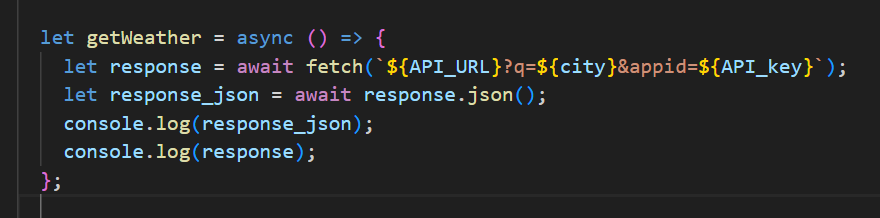


Rejected: failed



Async function always return a promise.

Promises help avoid **callback hell** (nested callbacks) and make the code more readable. (Then and catch is used with it )



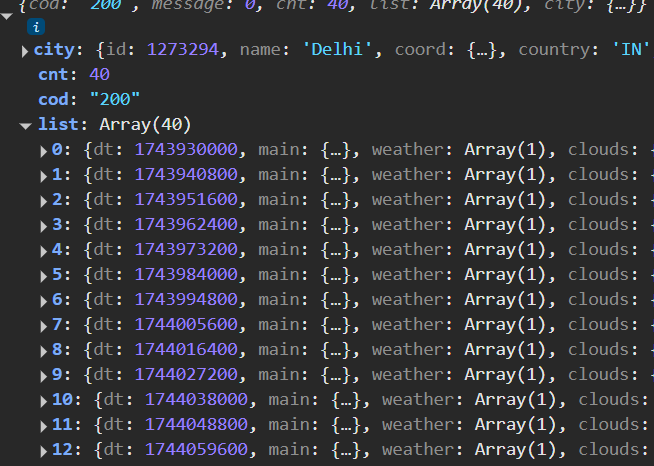
Here fetch function return **HTTP response object** which is stored in response variable. It will give something like this

A screen shot of a computer

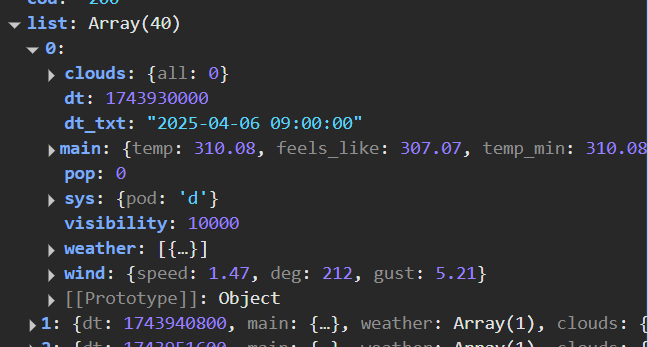
AI-generated content may be incorrect.

Inside body contain weather information but it is not in readable form , so we have use json to read that data. Response\_json will return **JavaScript object** containing the actual weather data from the OpenWeather API

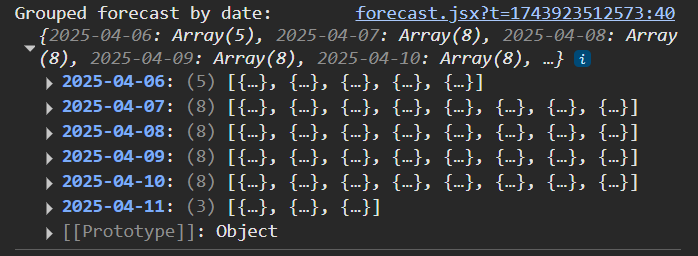
Forecast working:  
Forecast Api return the list that has data of every 3 hour.

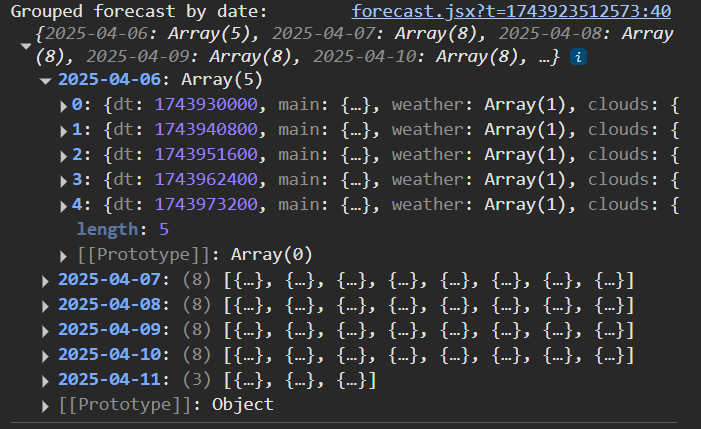
 Total element in the list :40  
 for 1 day : 8 (24/8)  
so, for 5 days = 5\*8=40

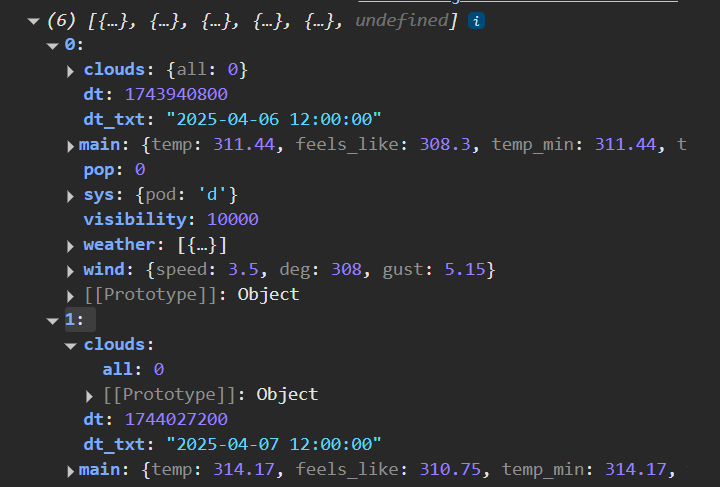
Now we want data forecast of every day , so will divide the list according to date present in the **dt\_txt**  and store it into the **object** (here forecastByDate ) in which key is the date and corresponding value is the list of objects. These objects contain every 3 hour data.

****



****



Now , we will **pick the "12:00:00" forecast entry** for each day from the 8 available entries. And store it in **f\_result** array . it is the array of object .