Weather app by using API(application programming interface)

Here at Nordic APIs, we strive to post content for API users at every experience level. We cover plenty of expert-level industry analysis, but sometimes it’s important to check in with new API users, so you can see how APIs can empower your programming **in practice**.

To this end, we’re going to show you how to build a simple weather app that consumes APIs to determine the user’s geolocation and then fetch their local weather data. We’ve created a [GitHub repository](https://github.com/For3stpunk/weather-app-demo) and spun up a [live demo](https://for3stpunk.github.io/weather-project/) too, so you can try it out for yourself and check your work!

For this tutorial, we’ll be using:

* Basic HTML
* Basic JavaScript
* Basic CSS

Over the course of this project, you’ll learn how to use the Fetch API and call a third-party API from within a JavaScript web application. We’ll be constructing this mostly from scratch.

**Styling The App**

For this tutorial, we’ll be creating a simple, minimal app that still looks appealing. The background image will change depending on the weather conditions, and all results will be returned using the [Lora Google font](https://fonts.google.com/specimen/Lora).

To start, create a blank file using your text editor of choice. Title it as index.html and then input the following:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Weather app</title>

<link rel="stylesheet" href="styles.css" />

<!-- Google Fonts -->

<link rel="preconnect" href="<https://fonts.gstatic.com>">

<link href="<https://fonts.googleapis.com/css2?family=Lora:ital,wght@0,700;1,600&display=swap>" rel="stylesheet">

</head>

<body>

<div class="container">

<img src="" alt="" srcset="" id="weather-icon">

<div id="location">Unable to Fetch Weather</div>

<div class="desc">No Information Available.</div>

<div class="weather">

<div class="c">Error</div>

<div class="circle"></div>

<div class="f">Error</div>

</div>

<div class="info">

<h4>Sunrise: <span class="sunrise">No Information Available</span></h4>

<h4>Sunset: <span class="sunset">No Information Available</span></h4>

</div>

</div>

<script src="scripts.js"></script>

</body>

</html>

index.html will be the template for your web app. Right now, it shows no information, but soon enough, it will be connected to real-time weather data.

In this block of code, we’re using multiple divs with classes and IDs, which will be important for interacting with the DOM. It also calls the styles.css and scripts.js files. All of this is wrapped in a container, which also contains an image tag with a blank src attribute, a location ID, and a div containing default text that reads “Unable To Fetch Weather.”

Finally, temperatures are loaded into a div called weather, which returns the current temperature in both Fahrenheit and Celsius. There are also two additional divs for sunrise and sunset.

Now we’re going to style the returns using CSS flexbox. We’ll start by restoring to the default settings.

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

We’ll then make it so the whole app uses the Lora font.

body {

font-family: 'Lora', serif;

}

Now let’s style the container. We’ll set the dimensions to 100 viewport height and 100 viewport width. We’ll then invoke CSS flexbox using the display: flex; command. We’ll center the results using justify-content:center and align-items:center.

We’ll make the background a radial gradient using an online gradient generator.

The end result will look like this.

.container {

height: 100vh;

width: 100vw;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

background: rgb(251, 242, 133);

background: radial-gradient(

circle,

rgba(251, 242, 133, 0.6334908963585435) 0%,

rgba(224, 196, 91, 0.8407738095238095) 35%,

rgba(230, 224, 113, 1) 100%

);

}

Now you’re going to do the same thing for the weather section:

.weather {

display: flex;

align-items: center;

margin: 15px 0;

font-size: 1.5rem;

}

We’ll finish up by styling the location div, description, and background.

#location {

font-size: 3rem;

font-weight: 800;

font-style: italic;

}

.desc {

font-size: 1.25rem;

text-transform: capitalize;

}

And:

.circle {

background-color: black;

border-radius: 50px;

height: 15px;

width: 15px;

margin: 0 15px;

}

Now let’s move on to working with the APIs.

**Getting Weather Data From APIs**

For this tutorial, we’ll be using the [OpenWeatherMap](https://openweathermap.org/) to retrieve weather data. OpenWeatherMap offers a free API for real-time weather data that you can use in your development projects. Of course, there are plenty of [other weather APIs](https://nordicapis.com/6-best-free-and-paid-weather-apis/) on the market.

First, go to OpenWeatherApp and select ‘Sign In’ in the top-right corner. Create an account. Once your account is created, navigate to the ‘API’ tab at the top of the page. Locate the ‘Current Weather Data API’ section and select ‘Subscribe’. Choose the Free option.

Once you’re subscribed, find the dropdown menu beneath your user name in the top-right corner. Select ‘My API Keys’. You’ll see your API key, which you can use to retrieve real-time weather data.

Now we can go back to building our weather app.

**Coding The JavaScript**

Now you’re going to create the scripts.js file we called earlier. The first thing you’re going to do in scripts.js is to create a variable called API where you’ll store your API key.

// OpenWeatherMap API. Do not share it publicly.

const api = '66869939e0b182ac2d6\*\*\*\*\*\*\*\*\*\*\*'; //Replace with your API

Next, you’re going to create an event listener that runs a function every time the page is loaded.

window.addEventListener('load', () => {});

Inside the function, we will create two variables, lat and long for the user’s latitude and longitude, and then use the navigator object to retrieve the user’s geolocation. That block of code will look like this:

window.addEventListener('load', () => {

let long;

let lat;

// Accessing Geolocation of User

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition((position) => {

});

}

});

The if(navigator.geolocation) method checks and sees if an object can be retrieved by the browser. If it is, a method called getCurrentPosition is used from the geolocation property. The position variable is then passed into a function.

If you were to consult the console log at this point, you’d see your current latitude and longitude. Your browser will ask your permission before displaying those variables, however.

The geolocation property has numerous variables, but we’ll limit ourselves to the latitude and longitude, which we’ll save inside long and lat variables.

window.addEventListener('load', () => {

let long;

let lat;

// Accesing Geolocation of User

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition((position) => {

// Storing Longitude and Latitude in variables

long = position.coords.longitude;

lat = position.coords.latitude;

});

}

Once you’ve got the latitude and longitude, you can use those to query the OpenWeatherMap API. Every API has a root endpoint. OpenWeatherMap’s is <https://api.openweathermap.org/data/2.5/weather>, which can then be modified following a ?. The end result might look something like this:

https://[api.openweathermap.org/data/2.5/weather?lat=$](http://api.openweathermap.org/data/2.5/weather?lat=$)%7Blat%7D&lon=$%7Blong%7D&appid=$%7Bapi%7D&units=metric

You can read more about formatting OpenWeatherMap queries [here](https://openweathermap.org/current).

The returned data is stored inside a JSON object called base.

If you were to look at the console log at this point, you’d see a static URL. If you were to open this site, you’d see all of the data that’s available. This is the URL we’ll be using with the Fetch API JavaScript object to retrieve the data from the API and return it to the app. A return.json() object translates the response into a JSON object. Then we’ll extract the data that we need using a technique called [destructuring](https://nordicapis.com/5-api-design-tips-beyond-rest/).

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition((position) => {

// Storing Longitude and Latitude in variables

long = position.coords.longitude;

lat = position.coords.latitude;

const base = `<https://api.openweathermap.org/>data/2.5/weather?lat=${lat}&lon=${long}&appid=${api}&units=metric`;

// Using fetch to get data

fetch(base)

.then((response) => {

return response.json();

})

.then((data) => {

const { temp } = data.main;

const place = data.name;

const { description, icon } = data.weather[0];

const { sunrise, sunset } = data.sys;

});

});

}

With this code, we’re extracting the temp data and storing it to a temp variable. The place-name is stored as name. The description and icon code is contained in the weather array. Finally, the sys object stores the times for sunrise and sunset.

A few of these items may need some further clarification. The first is the icon code. OpenWeatherMap contains a few icons for different weather conditions (you can see the full list [here](https://openweathermap.org/weather-conditions)). Once you have the icon code, you can insert it into:

http://[openweathermap.org/img](http://openweathermap.org/img)/wn/$%7Bicon%7D@2x.png

Then, you’ll get the appropriate icon for the weather conditions.

The time is the other detail that needs some fine-tuning. The OpenWeatherMap returns the time in epoch format. It needs to be converted into GMT before we can use it in our app. The formula for converting epoch time to GMT is as follows:

const timeInGMT = new Date(epochTime \* 1000);

Both the sunrise and sunset times need to be converted. We’re also going to include a URL in the blank img variable we created at the beginning. Finally, we’re going to convert the temperature from Celsius into Fahrenheit as we want our app to display both.

The formula for converting Celsius to Fahrenheit:

fahrenheit = (celcius \* 9) / 5 + 32

The final result should look something like this:

window.addEventListener('load', () => {

let long;

let lat;

// Accesing Geolocation of User

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition((position) => {

// Storing Longitude and Latitude in variables

long = position.coords.longitude;

lat = position.coords.latitude;

const base = `<https://api.openweathermap.org/data/2.5/weather?lat=>${lat}&lon=${long}&appid=${api}&units=metric`;

// Using fetch to get data

fetch(base)

.then((response) => {

return response.json();

})

.then((data) => {

console.log(data);

const { temp } = data.main;

const place = data.name;

const { description, icon } = data.weather[0];

const { sunrise, sunset } = data.sys;

const iconUrl = `<http://openweathermap.org/img/wn/>${icon}@2x.png`;

const fahrenheit = (temp \* 9) / 5 + 32;

// Converting Epoch(Unix) time to GMT

const sunriseGMT = new Date(sunrise \* 1000);

const sunsetGMT = new Date(sunset \* 1000);

});

});

}

});

Finally, we’ll finish up by accessing the DOM elements and changing the variables.

const iconImg = document.getElementById('weather-icon');

const loc = document.querySelector('#location');

const tempC = document.querySelector('.c');

const tempF = document.querySelector('.f');

const desc = document.querySelector('.desc');

const sunriseDOM = document.querySelector('.sunrise');

const sunsetDOM = document.querySelector('.sunset');

Now we can easily change the icon image using a selector and the .src property.

iconImg.src = iconUrl;

Then we’ll use the toFixed(2) method to restrict the results to two decimal points. Then we’ll use the toLocaleDateString() and toLocaleTimeString() methods to convert GMT to the local time. Lastly, the textContent property sets the inside text, so it’s passed on the right side.

The final JavaScript code should look something like this:

// Openweathermap API. Do not share it publicly.

const api = 'caa7a056213fc43a1\*\*\*\*\*\*\*\*\*\*\*\*'; //Replace with your API

const iconImg = document.getElementById('weather-icon');

const loc = document.querySelector('#location');

const tempC = document.querySelector('.c');

const tempF = document.querySelector('.f');

const desc = document.querySelector('.desc');

const sunriseDOM = document.querySelector('.sunrise');

const sunsetDOM = document.querySelector('.sunset');

window.addEventListener('load', () => {

let long;

let lat;

// Accesing Geolocation of User

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition((position) => {

// Storing Longitude and Latitude in variables

long = position.coords.longitude;

lat = position.coords.latitude;

const base = `<https://api.openweathermap.org/data/2.5/weather?lat=>${lat}&lon=${long}&appid=${api}&units=metric`;

// Using fetch to get data

fetch(base)

.then((response) => {

return response.json();

})

.then((data) => {

const { temp } = data.main;

const place = data.name;

const { description, icon } = data.weather[0];

const { sunrise, sunset } = data.sys;

const iconUrl = `<http://openweathermap.org/img/wn/>${icon}@2x.png`;

const fahrenheit = (temp \* 9) / 5 + 32;

// Converting Epoch(Unix) time to GMT

const sunriseGMT = new Date(sunrise \* 1000);

const sunsetGMT = new Date(sunset \* 1000);

// Interacting with DOM to show data

iconImg.src = iconUrl;

loc.textContent = `${place}`;

desc.textContent = `${description}`;

tempC.textContent = `${temp.toFixed(2)} °C`;

tempF.textContent = `${fahrenheit.toFixed(2)} °F`;

sunriseDOM.textContent = `${sunriseGMT.toLocaleDateString()}, ${sunriseGMT.toLocaleTimeString()}`;

sunsetDOM.textContent = `${sunsetGMT.toLocaleDateString()}, ${sunsetGMT.toLocaleTimeString()}`;

});

});

}

});

And that’s it!

If you’ve been following along and writing your own code, all you have to do is select the index.html file, and your new API-driven weather app will load in your browser. You can also clone the GitHub repository we linked above. Then all you need to do is include your own API key in the ‘scripts.js’ file, and you’ll have a working weather web app in a matter of moments!

You can also see live demo to see how it works for yourself! By using this link

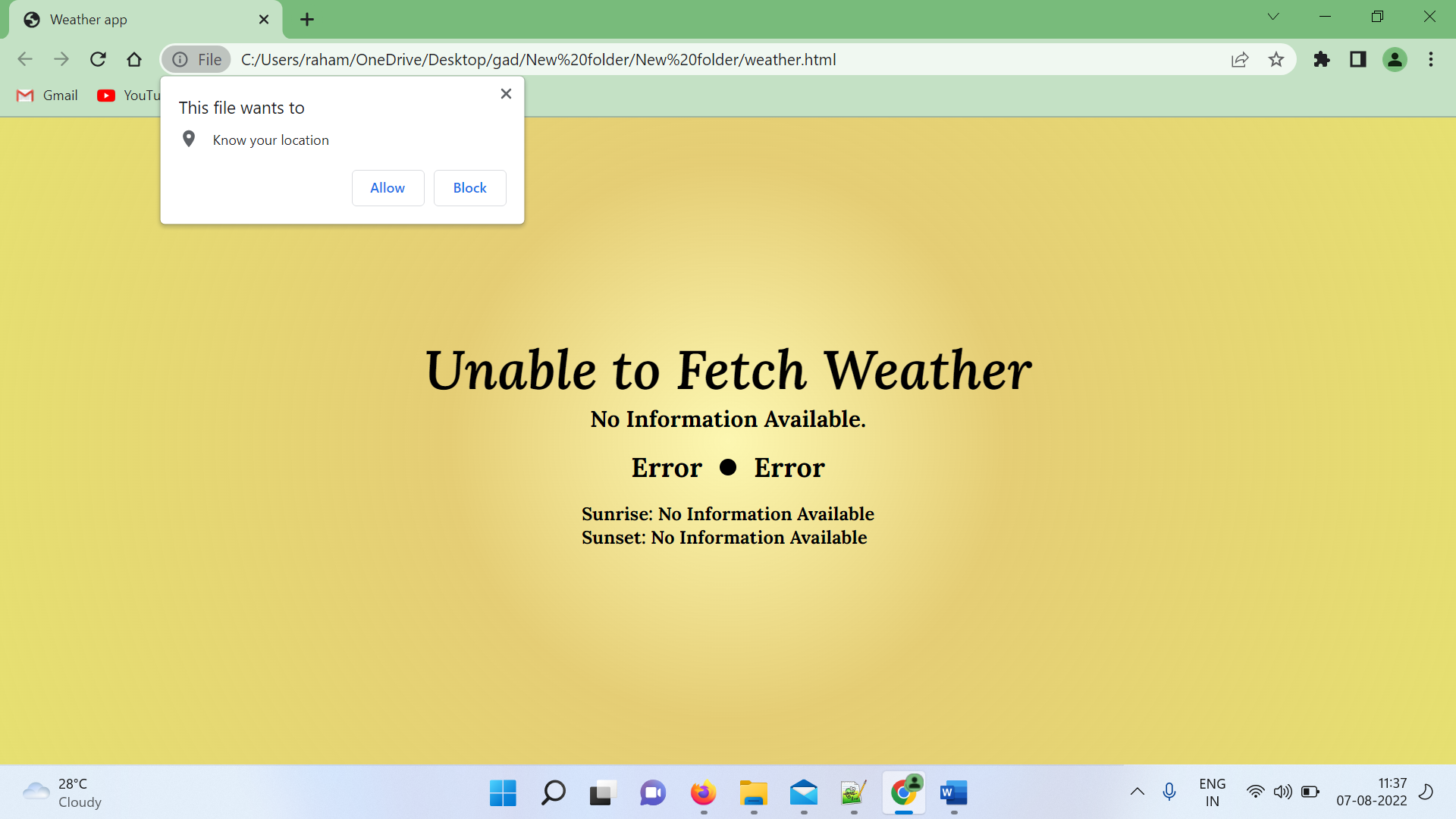
**How To Build An API-Driven Weather App: Final Thoughts**

As you can see, it’s easy to work with APIs once you know even a little about what you’re doing. For one, there are a ton of built-in resources and libraries that make integrating with and consuming APIs a breeze for every major programming language you can think of (and most minor ones, too). In this tutorial, we saw the Fetch API method in action. We also saw some of its endless potential applications.

By writing this [in JavaScript](https://nordicapis.com/10-top-web-development-frameworks-for-2022/), you should hopefully have a better notion of how easy it is to destructure JSON objects, which make up the bulk of API assets. Finally, this app should give you an idea of how easy, and utterly effective, taking raw data and structuring and styling it with CSS can be.

Keep in mind this project is simply meant for demonstration purposes. Real, high-use production applications should obfuscate their API key.

First enable the location to fetch the data



After enable location it will take the current info of your location weather

