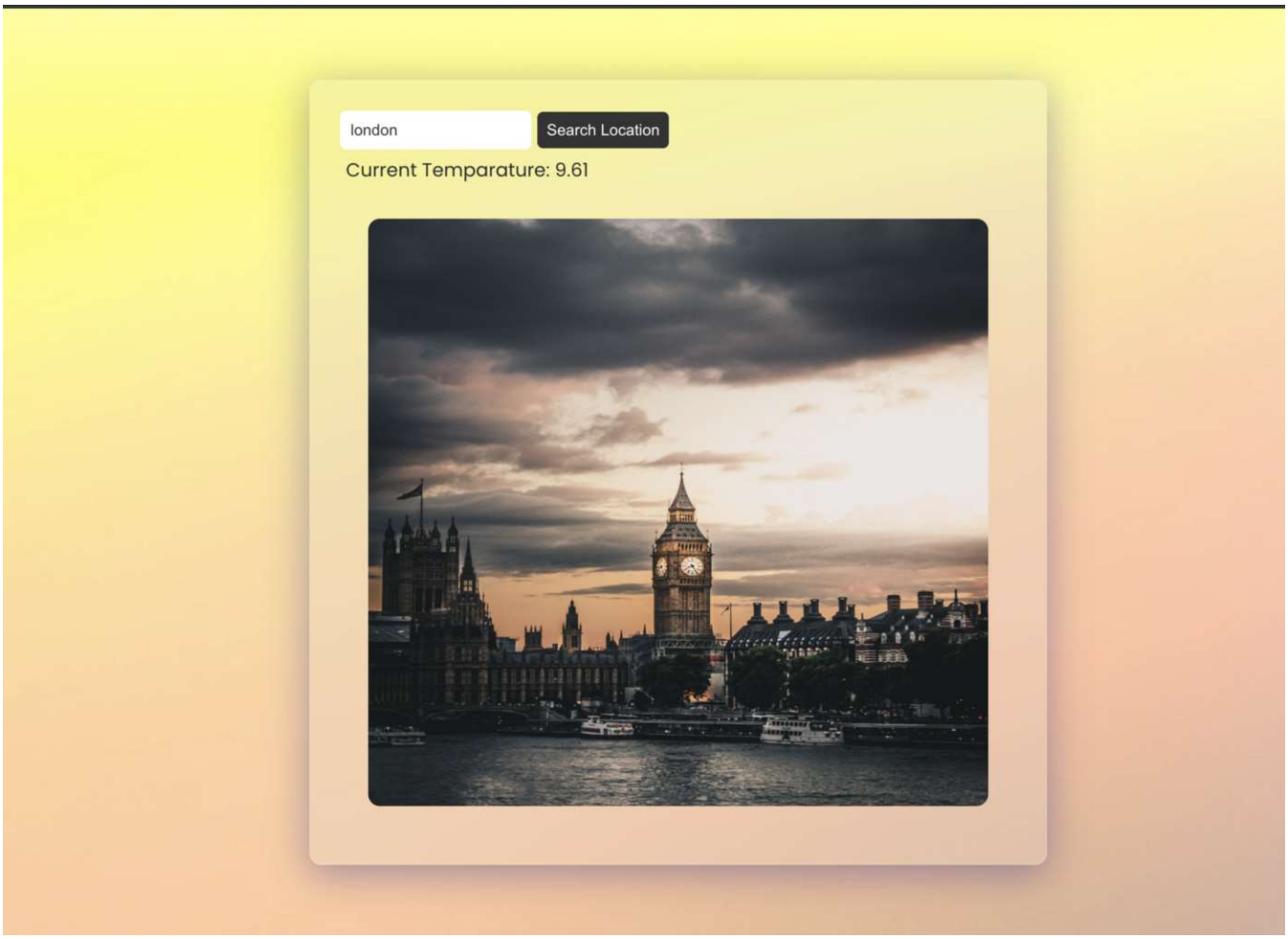
Published in Sonny Sangha

Priyanshu Saraf Follow
Apr 1, 2021 · 5 min read · Listen



# How To Make A Weather App With React JS

As we all know, React is a really powerful front-end library. To get a good idea about react, it's best to create applications and see how different hooks interact with each other. Let me give you a sneak peek related to what we will be building in this tutorial:



Before anything else, you need to have node js installed. To install node, please go to nodejs.org and install the latest stable version on your PC.

With that done, head off to your terminal and enter the following command in any folder that you want the weather app to be:

```
npx create-react-app weather-app-react
```

I'm going to create the application on my desktop, and I will call it weather-app-react.

The above command will simply create a react app for us that we can start working with immediately. After that is done, open up vs code in the terminal by using the command:

```
code .
```

For those of you who are using vs code insiders, use:

```
code-insiders .
```

This opens up vs code in the current directory.

Now, remove all the code in App.js, remove the setupTest.js, App.test.js, logo.svg, and then enter the following code in App.js:

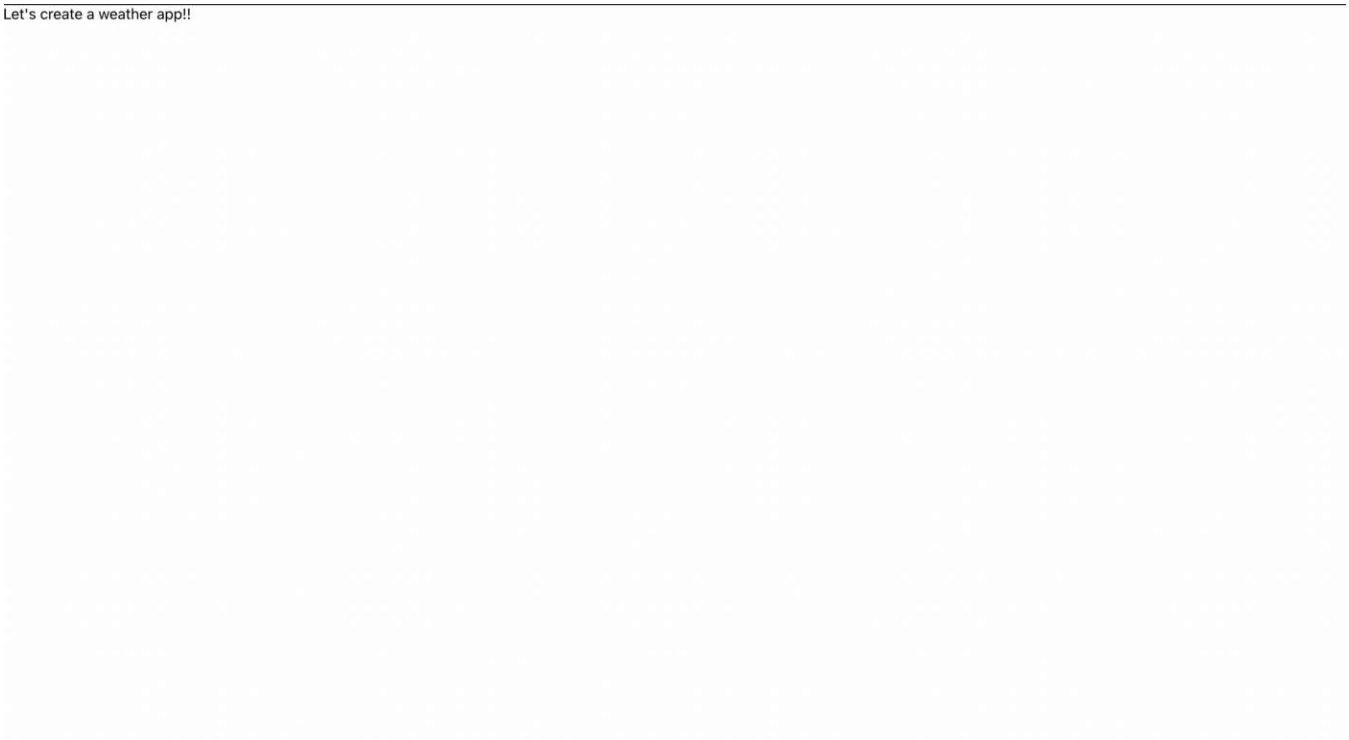
```
import React from "react";
import "./App.css";
function App() {
return <div>Let's create a weather app!!</div>;
```


And then go to your terminal and enter the following command:

```
npm start
```

This spins up the react application for us, which is exactly what we want.

You should see something like this:




If you got till here, perfect. Let's keep moving.

Proceed to OpenWeatherMap and create a free account for the current weather data API. Then, you will get an API key.

Also, go to the API of Unsplash and sign up for a free developer account. You will be prompted to create a project, name it however you wish to, and then you will again get an API key.

Unsplash will provide us with the images that we need and open weather map will provide us with the weather data.

```
import React, { useState, useEffect } from "react";
```

If you do not know much about useState and useEffect, I highly recommend checking out our hooks tutorial here.

Now, we want to create three pieces of state for our application like so:

```
import React, { useState, useEffect } from "react";
import "./App.css";
function App() {
const [weather, setWeather] = useState({});
const [locations, setLocations] = useState("london");
const [photos, setPhotos] = useState([]);
return <div>Let's create a weather app!!</div>;
}
export default App;
```

The weather state will store the response that we get from the weather API, the locations state will store the response that we get from the user's input, and the photos state will store the response that we get from the unsplash API.

You might wonder why we've given an initial state of "london" to the locations API. This is because we don't want the user to see a blank screen upon the site's load, as it results in a bad user experience.

The next thing that we want to do is setup a function that fetches data from the apis like so:

```
1  import React, { useState, useEffect } from "react";
```


```
  function ifClicked() {
    fetch(
      `http://api.openweathermap.org/data/2.5/weather?q=${locations}&APPID={API_KEY_FOR_WEATHER
    )
      .then((res) => {
        if (res.ok) {
          console.log(res.status);
          return res.json();
        } else {
          if (res.status === 404) {
            return alert("Oops, there seems to be an error!(wrong location)");
          }
          alert("Oops, there seems to be an error!");
          throw new Error("You have an error");
        }
      })
      .then((object) => {
        setWeather(object);
        console.log(weather);
      })
      .catch((error) => console.log(error));
    fetch(
      `https://api.unsplash.com/search/photos?query=${locations}&client_id={API_KEY_FOR_UNSPLAS
    )
      .then((res) => {
        if (res.ok) {
          return res.json();
        } else {
          throw new Error("You made a mistake");
        }
      })
      .then((data) => {
        console.log(data);
        setPhotos(data?.results[0]?.urls?.raw);
      })
      .catch((error) => console.log(error));
  }
  return <div>Let's create a weather app!!</div>;
}

export default App;
```

In the first API call, we are giving it the parameter of units=metric which might confuse some of you. Traditionally, we get all the data back in scientific terms (kelvin for ex-) which no one really likes to see. With this parameter in the Fetched URL, we get back the response in a format that is easy to view and understand. Then, we simply set the object response that we get back from the API to the piece of state that we discussed earlier.

While fetching data from the Unsplash API, what we really want is just the link of the first image that comes up. We enter the parameter of the query to be the location that the user enters, and we assume that the first result (hence the 0) will be optimized. If you want to, you can make this a lot more complex, but that's not what I am aiming for here. When you log the output of both the responses, you will find a bunch of data that you can play around with. For the sake of simplicity, I have chosen the link that we get from the Unsplash API although you can take it to the next level.

Now it's time for us to actually develop the UI of this application. This is the final piece of code that we need for the App.js file:

```
1  import React, { useState, useEffect } from "react";
2
3  import "./App.css";
4
5  function App() {
6    const [weather, setWeather] = useState({});
7    const [locations, setLocations] = useState("london");
8    const [photos, setPhotos] = useState([]);
9    useEffect(() => {
10     ifClicked();
11   }, []);
12
13   function ifClicked() {
14     fetch(
15       `http://api.openweathermap.org/data/2.5/weather?q=${locations}&APPID={APP_ID}&units=metri
16     )
17       .then((res) => {
18         if (res.ok) {
19           console.log(res.status);
20           return res.json();
```

```
26            throw new Error("You have an error");
27          }
28        })
29        .then((object) => {
30          setWeather(object);
31          console.log(weather);
32        })
33        .catch((error) => console.log(error));
34      fetch(
35        `https://api.unsplash.com/search/photos?query=${locations}&client_id={APP_ID}`
36      )
37        .then((res) => {
38          if (res.ok) {
39            return res.json();
40          } else {
41            throw new Error("You made a mistake");
42          }
43        })
44        .then((data) => {
45          console.log(data);
46          setPhotos(data?.results[0]?.urls?.raw);
47        })
48        .catch((error) => console.log(error));
49    }
50    return (
51      <div className="app">
52        <div className="wrapper">
53          <div className="search">
54            <input
55              type="text"
56              value={locations}
57              onChange={(e) => setLocations(e.target.value)}
58              placeholder="Enter location"
59              className="location_input"
60            />
61            <button className="location_searcher" onClick={ifClicked}>
62              Search Location
63            </button>
64          </div>
65          <div className="app__data">
66            <p className="temp">Current Temparature: {weather?.main?.temp}</p>
67          </div>
68          <img className="app__image" src={photos} alt="" />
```

```
74  export default App;
```

then displayed it on the front-end. All that's left is for us to now develop the CSS for this. We are going to be using a little bit of glassmorphism as well, which is really going to level up the application design.

Here is the CSS code for the app:

```
1   @import url("https://fonts.googleapis.com/css2?family=Poppins:wght@400&display=swap");
2   .app {
3     display: flex;
4     flex-direction: column;
5     justify-content: center;
6     align-items: center;
7     background: url(./background.jpg);
8     height: 100vh;
9     width: 100vw;
10  }
11
12  .wrapper {
13    padding: 25px;
14    border: 0.1px solid rgb(96, 96, 96);
15    display: flex;
16    flex-direction: column;
17    width: 40%;
18    height: auto;
19    background: rgba(219, 216, 216, 0.25);
20    box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
21    backdrop-filter: blur(10px);
22    -webkit-backdrop-filter: blur(4px);
23    border-radius: 10px;
24    border: 1px solid rgba(255, 255, 255, 0.18);
25  }
26
27  .app__image {
28    margin: 24px;
29    max-height: 500px;
30    width: auto;
31    border-radius: 10px;
```

```
    border: none;
    border-radius: 5px;
    margin-left: 5px;
  }
  .location_input {
    padding: 8px;
    border: 1px solid rgba(255, 255, 255, 0.6);
    color: #333;
    border-radius: 5px;
  }

  .temp {
    margin: 5px;
    font-family: Poppins;
    color: #2e2e2e;
  }
```

I'm using the Poppins font from Google fonts but feel free to use any font that you desire. In glassmorphism, we give the background element a blur filter so that we get a beautiful component design. That's why we use the backdrop-filter: blur(10px) CSS code here.

With that said, we finally have the application developed. It was pretty simple, wasn't it?

I hope that all of you enjoyed developing this application along with me. I left this application open-ended, and I want you to provide your own code to it and customize it however you can.

For more complex apps be sure to check out Sonny's YouTube channel. Thank you for reading the article and don't hesitate to buy me a coffee if you like the content!


Thank You!

Priyanshu Saraf
(PAPA Team Writer)