Project Documentation

1.1 Project overview

1.2 Implementation

2.1 Introduction to react hooks

2.2 Snippets of code

3.1 CSS tricks

3.2 Advanced Styling

**1.1 Project Overview**

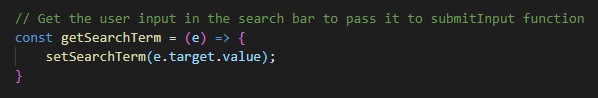
In this project, I have implemented a simple react app to fetch data from open weather API and display it on the screen. Primarily, my main focus was to create something reusable that can benefit any sort of data fetching no matter what the data is. Luckily, this worked pretty well and the outcome was enough to serve the overall trajectory which is, fetching the weather data when a user inputs a city’s name and displaying the data accordingly. Not to mention, the decent work that paid off in showing the app in the desired style and look. Hopefully it meets the expectations and even exceed them.

**1.2 Implementation**

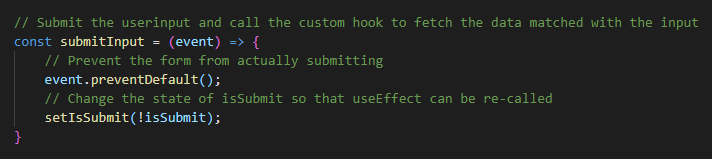
Speaking of the technical side, the code is implemented using functional components with not even a single class. The files are listed as below respectively:

* Main.js
* useHttp.js
* styles.scss

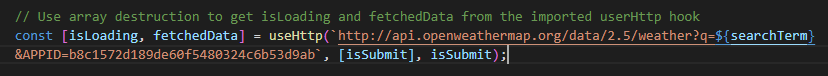
The first file (“main.js”) contains the functional component in which we render our data with html tags and scss for styling. We first declared some state using “useState” hook in order to get the input from the user through the function “onChange”.



Then, we send the input to the submission function “submitInput” in which we change the value of “isSubmit” to true as it was declared false beforehand. The main reason behind this will be explicitly explained later on.



After we submit, we permit the custom hook useHttp to set to work using the input claimed from the user. Thereby, we pass our endpoint and also the API key and then the name of the city that the user inputs (Which is defined as a searchTerm in the app). Finally, the results are shown on the screen with all corresponding data.



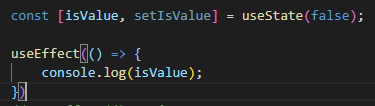
The first argument is simply the url that we are fetching the data from. Next argument is the dependency. It typically re-renders the component when one of its elements changes only. It helps avoid redundancy. The last argument is simply “isSubmit” value that allows the customized hook “useHttp” to fetch the data only when it is true. The main purpose is that we only want to use “useHttp” hook when the user submits a value not every time the app is rendered.

The idea is very simple and straightforward. It can be easily customized to fit the needs of any other projects and that is all the more reason why it comes first. Another key point to remember, using scss was not a coincidence. Yet, it was the first choice as it contains several features that helps maintain the projects readability and reusability. For example, nesting and declaring variables which are not applicable in CSS.

**2.1 2.2 Introduction to React Hooks**

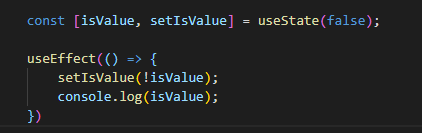
React hooks have been released recently and they gained a lot of traction due to some issues related directly to the code’s readability and simplicity. First things first, they turn your react components into functional components. No classes anymore, Seems interesting? Of course. Being stuck in a class component and passing a lot of states and props throughout the project especially when you have large sets of components and they are all attached together. Functional components are a lot neater and much less ugly unlike class components. The first react hook is “useState” which simple implies “this.state” according to the old way of declaring the state in classes. There is also “useEffect” and we can say it the replacement of componentDidMount lifecycle method that also runs when the component is rendered. There can also be dozen of ways to incorporate “useEffect” hook within your app but this is what we used it for in this project.

To get things much clearer, here is a simple example about how to use these two hooks.



So as we can see above, we declared the state of “isValue” to false and then we used “useEffect” hook to print the value when the app renders.

But, what is “setIsValue” used for? It is simply used exactly the same way “this.setState” works though which you can change the state dynamically in the app. And here is a simple example on how to use it as well.

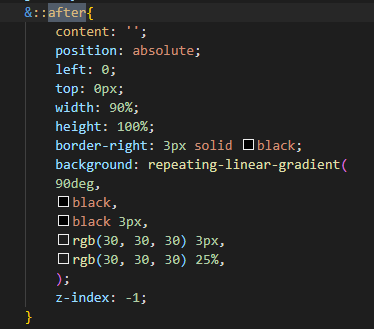


So what do you think the output could be. Yes, it will be displayed as “true” because we have changed the value of the state using “setIsValue”

**3.1 CSS tricks**

The styling work was a bit burdensome. Yet, it is tailored to perfectly match the design. There are some pseudo selectors in CSS that helped on a great scale to get the styling done this way. We are not to mention all of them but some are worthy such as (first-of-type, last-of-type, nth-of-type) and they help you to select between several tags of the same kind.

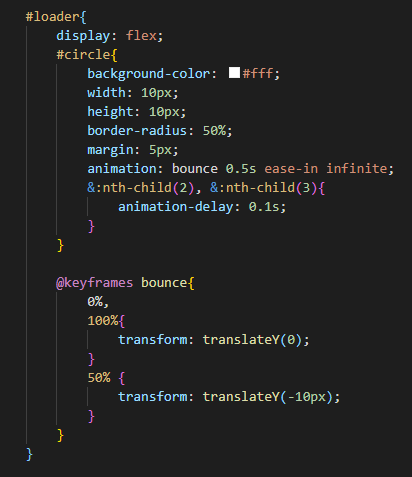
Also the (::before and ::after) that help add some elements before and after each other. They have to get a value in “content” and we can -by the way- append icons also by adding their Unicode with a forward slash before it. For example (content: “\f102”).



Here us how I divided the background into 4 divisions and added the design header using these pseudo selectors shown above in the figure.

**3.2 Advanced Styling**

Here comes the last part of our simple documentation which is the few advancements that I added to the styling using CSS animation as shown below.



I styled the loader as three consecutive circles that animate with a delay between each of them and then translate in Y axis in order to give the look of moving circles from left to right. This works infinitely as long as the data is not fetched yet. Otherwise, it disappears.

**Outro**

In the end, after going through every line in the code and explaining it as much as possible. I hope that things are obvious and clear now. Feel free to visit my github repo if necessary, as any pull requests are most welcomed. Here is the link:

<https://github.com/Saifsamirk/weatherApp>