Challenge 6 / Weather App using APIs

https://stephcambria.github.io/WeatherAPIApp/

Reading through the acceptance criteria, I already felt overwhelmed, especially since I feel as though I still don't fully understand how to use APIs to begin with. While I was skimming through the Weather API documentation, I quickly realized that I need to see how this works in action before I even think about writing any code. I wanted to see how more seasoned developers not only use APIs like this, but also how they choose to structure their code to accommodate it.

Mostly every source I used had a very simple html skeleton, so I opted for the exact same simplicity, knowing that the bulk of the information on screen will be added dynamically via javascript:

```
<body>

<!-- search bar -->
<div class="card">

   <div class="search">
       <input type="text" class="search-bar" placeholder="Search">
       <button> <img src="./assets/images/search_FILL0_wght400_GRAD0_opsz48.svg"
></button>
   </div>

<!-- weather -->
<div class="weather">

   <h4 class="city">Weather in Denver</h4>
   <h1 class="temp">51°</h1>
   <img src="" alt="" class="icon" />
   <div class="description">Cloudy</div>
   <div class="humidity">Humidity: 60%</div>
   <div class="wind">Wind speed: 6.2 mph</div>

</div>

</div>


</body>
```

For the placeholder text here, I picked a random city, random temperature, and basic additional information. I would be adding more later, and just wanted a baseline to work with for now.

While digging around, I learned about an API called Unsplash, where a user can implement randomly generated, high definition images onto their application for free. I believe I read on their website that it is being deprecated, but is still usable, so I figured why not practice using APIs by using an extra API source in this project?

I decided that the design I would go with is a centered card with all of the core components laid out in the html div, and a randomly-changing background through Unsplash. The main card would hold the current weather information, and I would generate smaller cards with the five day forecast underneath. I made a point to keep my css very clean and simple, and wanted to let the API I had implemented do the bulk of the visual work.

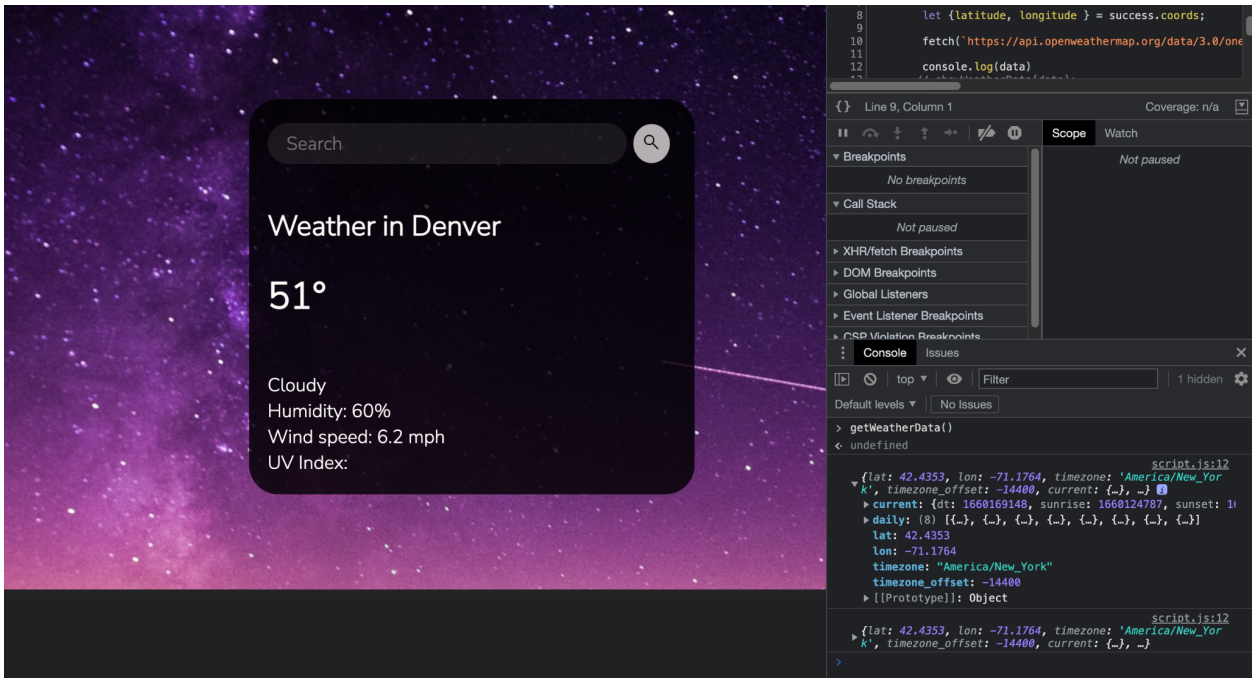
Here is where I really cracked into my research. I wanted to see exactly how other developers used this API specifically: how they implement it, how they organize the information it generates for them, how they choose to display that information on their applications, how they manipulate it, everything. My biggest goal with this project was to feel highly acquainted with APIs. Not an

expert, because that would be impossible, but *comfortable* with them, especially since the last challenge wound up being so confusing and unsatisfying for me.

I noticed that the one call data required me to use latitude and longitude rather than simply a city name. My immediate thought was “oh good! I don’t understand how to convert that”, and my immediate reaction was to deep dive into google to figure it out. Luckily, I found another helpful source, and I began to rework my api call like this:

```
getWeatherData()
function getWeatherData () {
   navigator.geolocation.getCurrentPosition((success) => {

       let {latitude, longitude } = success.coords;



fetch(`https://api.openweathermap.org/data/3.0/onecall?lat=${latitude}&lon=${longitude
}&exclude=hourly,minutely&units=metric&appid=${apiKey}`).then(res =>
res.json()).then(data => {

       console.log(data)
       })

   })
}
```

With a result of:



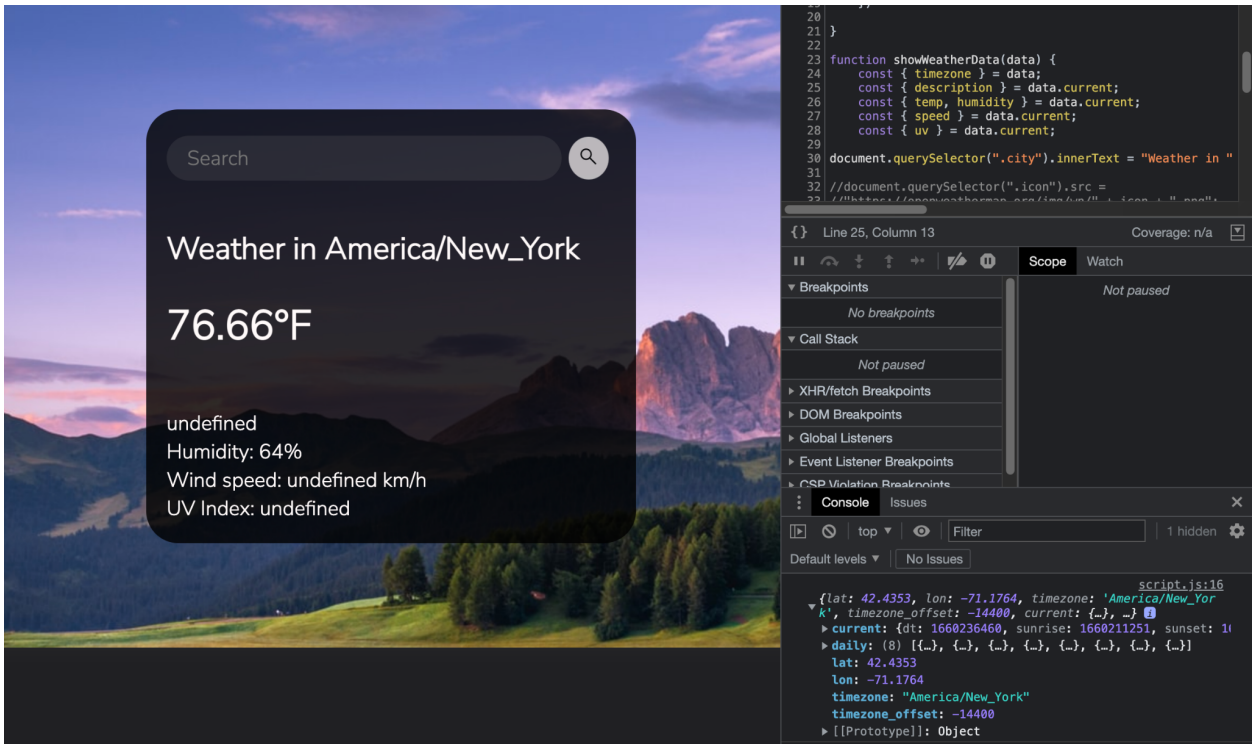
Problem seemed to be solved.

Next it was time to work on the display function I had started writing out. To do this, I had to grab each element I made a placeholder for in the html file and replace their contents dynamically through javascript. Since I was able to generate a list of timezone-specific content through the console log, I would be able to specify which information I actually want displayed on the page.

Here is the initial code I got working:

```
function showWeatherData(data) {
   const { timezone } = data;
   const { description } = data.current;
   const { temp, humidity } = data.current;
   const { speed } = data.current;
   const { uv } = data.current;

document.querySelector(".city").innerText = "Weather in " + timezone;

//document.querySelector(".icon").src =
//"https://openweathermap.org/img/wn/" + icon + ".png";

document.querySelector(".description").innerText = description;
```

```
document.querySelector(".temp").innerText = temp + "°F";

document.querySelector(".humidity").innerText =
"Humidity: " + humidity + "%";

document.querySelector(".wind").innerText =
"Wind speed: " + speed + " km/h";

document.querySelector(".uv-index").innerText = "UV Index: " + uv;

document.querySelector(".weather").classList.remove("loading");

}
```



Not quite there yet, but I was getting dynamically changing results! I was able to change the return from undefined to the correct information by adding all of the variables I declared to one string:

```
function showWeatherData(data) {
   const { timezone } = data;
  // const { description } = data.current;
```

```
   const { temp, humidity, wind_speed, uvi } = data.current;
```

And to get the correct city name rather than a timezone, I implemented a separate fetch function:

```
// ========= get city function ========= //

getCity()

function getCity() {



fetch(`https://api.openweathermap.org/data/2.5/weather?q=Boston&appid=${apiKey}`).then
(res => res.json()).then(data => {
      console.log(data)
      getWeatherData(data);
})
}
```

```
document.querySelector(".city").innerText = "Weather in " + city;
```

Up until this point, I felt as though I was really fumbling when it came to writing out the functions. I wound up writing walls of (unfinished) code that was actually on the right track, but second-guessed myself and kept deleting it. It felt as though I was trying to write too much for something so simple, and got paranoid that I was overthinking how many steps I needed to take again. When I asked my dad, though, he assured me that it's normal to make multiple API calls in multiple functions to achieve the end goal for an application.

Getting the weather icon and description was tricky at first, but I figured I would need to just find the correct array of information that my console log returns, and then parse it correctly. And that really was all it came down to:

```
function showWeatherData(city, data) {
 const { temp, humidity, wind_speed, uvi } = data.current;
const { icon, description } = data.current.weather[0];

document.querySelector(".city").innerText = "Weather in " + city;
 document.querySelector(".icon").src = "http://openweathermap.org/img/wn/" + icon +
".png";
 document.querySelector(".description").innerText = description;
```

```
 document.querySelector(".temp").innerText = temp + "°F";
 document.querySelector(".humidity").innerText =
"Humidity: " + humidity + "%";
 document.querySelector(".wind").innerText =
"Wind speed: " + wind_speed + " km/h";
 document.querySelector(".uv-index").innerText = "UV Index: " + uvi;



}
```

So now I still needed the date, a five day forecast, a working search function, local storage, and a color-coded UV index. Going in order of this list, I used the moment js function I made in the last project to get the current date:

```
const currentDate = document.querySelector("#date");
function date() {
   var now = moment();
   $("#currentDate").text(now.format("dddd, MMMM Do YYYY"));
}
setInterval(date, 1000);
date();
```

Then I got to work on the forecast:

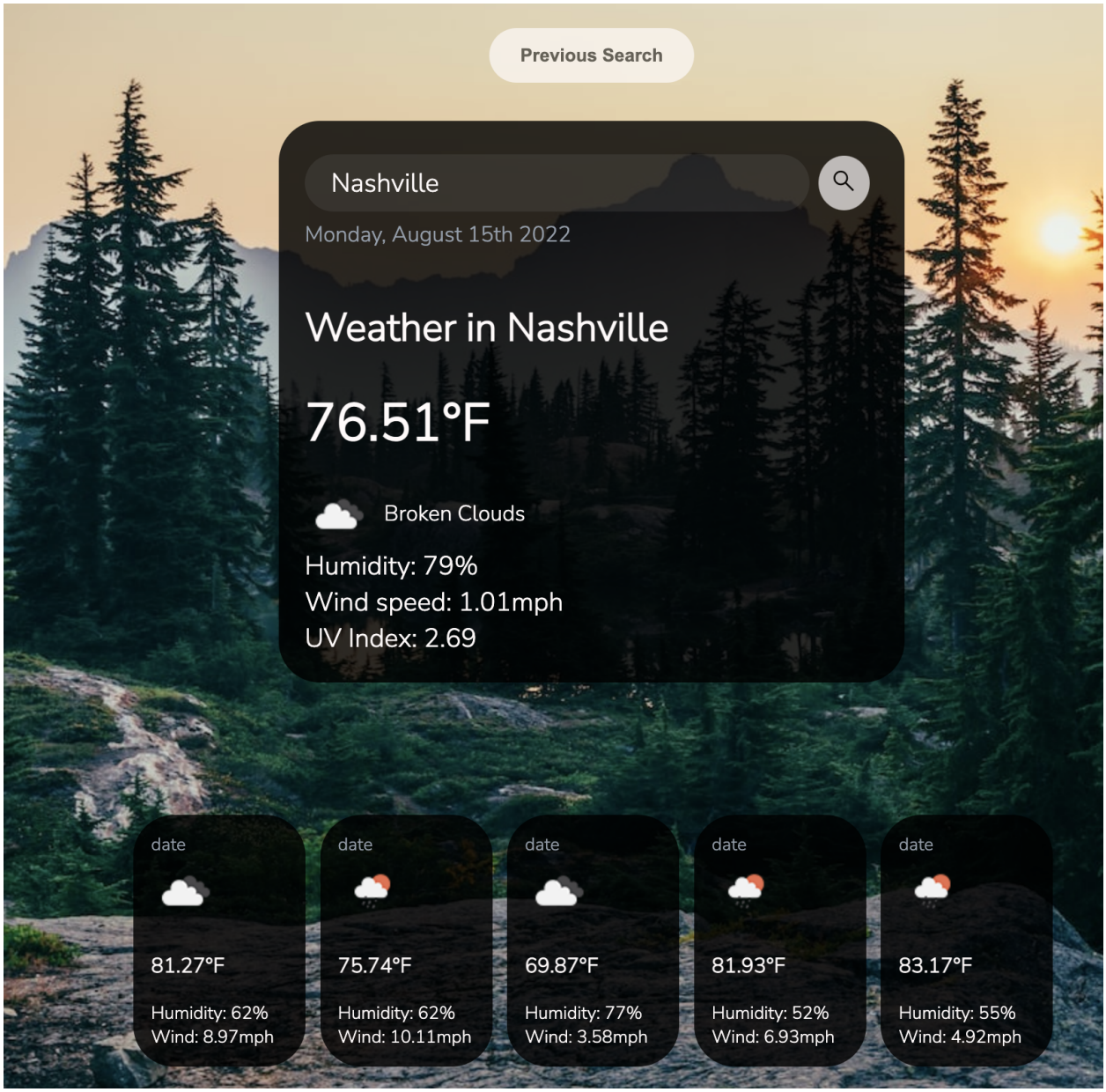
```
  function getForecastData() {



fetch(`https://api.openweathermap.org/data/2.5/forecast?q=Boston&appid=${apiKey}`).the
n(res => res.json()).then(data => {
   console.log(data);
  })
```

I also added a button that will load the previous search while I was at it. To get the forecast data, I dug into the information provided through Open Weather Map's one call API, and copied the information provided:

```
// ========= display 5 day forecast function ========= //

function showForecast(data) {


      // ========== day 1 ========== //

                // var { temp } = data.daily[0].temp.day;
                // var { humidity, wind_speed } = data.daily[0];
                 var icon = data.daily[0].weather[0].icon;
```

```
                    var temp = data.daily[0].temp.day;
                    var humidity = data.daily[0].humidity;
                    var wind_speed = data.daily[0].wind_speed;



                     document.querySelector("#forecastIcon0").src =
"http://openweathermap.org/img/wn/" + icon + ".png";

                     document.querySelector("#forecastTemp0").innerText = temp + "°F";

                     document.querySelector("#forecastHumidity0").innerText =
"Humidity: " + humidity + "%";

                     document.querySelector("#forecastWind0").innerText = "Wind: " +
wind_speed + "mph";



       // ========== day 2 ========== //

                var temp = data.daily[1].temp.day;
                var humidity = data.daily[1].humidity;
                var wind_speed = data.daily[1].wind_speed;
                var icon = data.daily[1].weather[0].icon;


                     document.querySelector("#forecastIcon1").src =
"http://openweathermap.org/img/wn/" + icon + ".png";

                     document.querySelector("#forecastTemp1").innerText = temp + "°F";

                     document.querySelector("#forecastHumidity1").innerText =
"Humidity: " + humidity + "%";

                     document.querySelector("#forecastWind1").innerText = "Wind: " +
wind_speed + "mph";
```

I divided up each day in the DOM for simplicity's sake. The output:

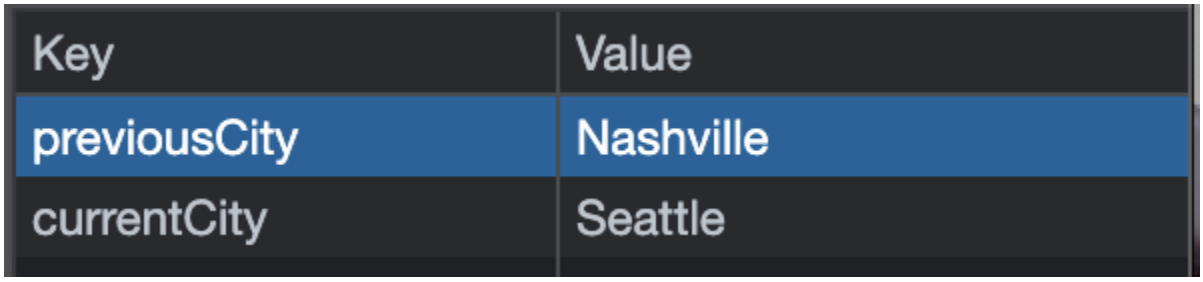


As you can see, I also got the search bar to work! I thought it would be tricky, but there are thankfully about a million resources around for this very function, and amazingly, I didn't need to use them:

```
<div class="search">
      <input type="text" class="search-bar" placeholder="Search">
      <button onclick="getCity()"> <img
src="./assets/images/search_FILL0_wght400_GRAD0_opsz48.svg" ></button>
   </div>


// ========= get city function ========= //
```

```
function getCity() {

   let city = document.querySelector("input").value;



fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`).the
n(res => res.json()).then(data => {

      localStorage.setItem('previousCity', localStorage.getItem('currentCity'));
      localStorage.setItem('currentCity', city);
      getWeatherData(data);

})
}
```

Because I had already set up both my city name function and my one call function, all I really needed to do was tell my DOM how to access them. I intentionally saved the search function for this late in the process because I assumed I would be able to set it up this way. I know this isn't the ideal way for developers to call their functions, but it works for now. In my getCity function, I also began the local storage process. Rather than have an entire index of search history cluttering up my interface, I wanted a button that would simply load the most recent search when clicked. Here is how it looks in the chrome devtools:



As usual, I made the process far more complicated for myself than I needed to, and had to grab my father to help with this function. I was trying to build an entirely new, detailed function to display the previous search, and he simplified it to just a few lines of code:

```
function previousCity() {
   var city = localStorage.getItem('previousCity');
```

```
fetch(`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`).the
n(res => res.json()).then(data => {

   document.querySelector("input").value = city;
   getWeatherData(data);
})
}
```

```
<!-- most recent search -->
<button id="previousCity" onclick="previousCity()">previous search</button>
```

It turns out all I had to do was specify the absolute bare minimum so that the function was only looking for the previous input. Maybe I shouldn't try to write code the second I wake up.

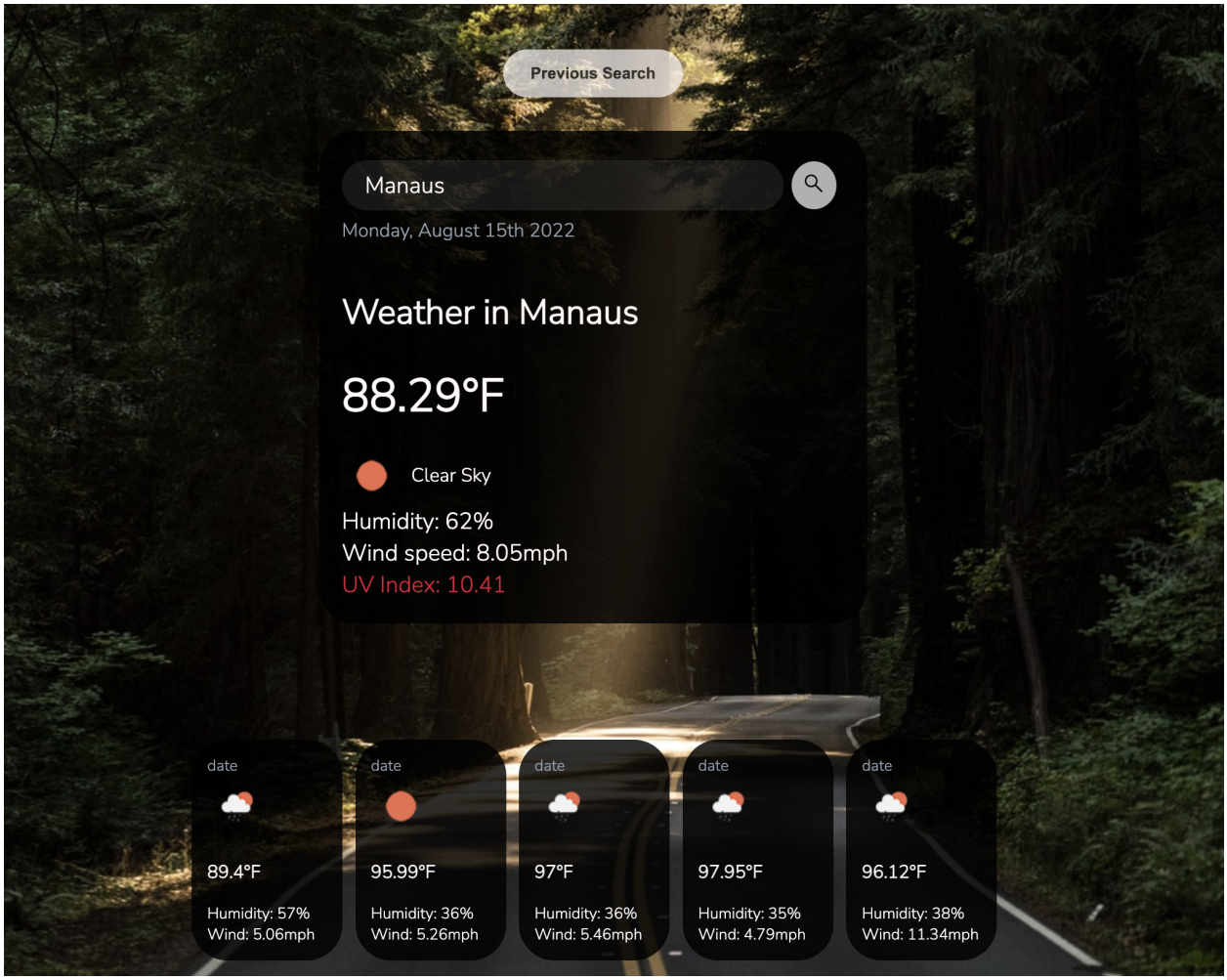
Now all I had left to do was color-code the UV index and get the forecast dates.

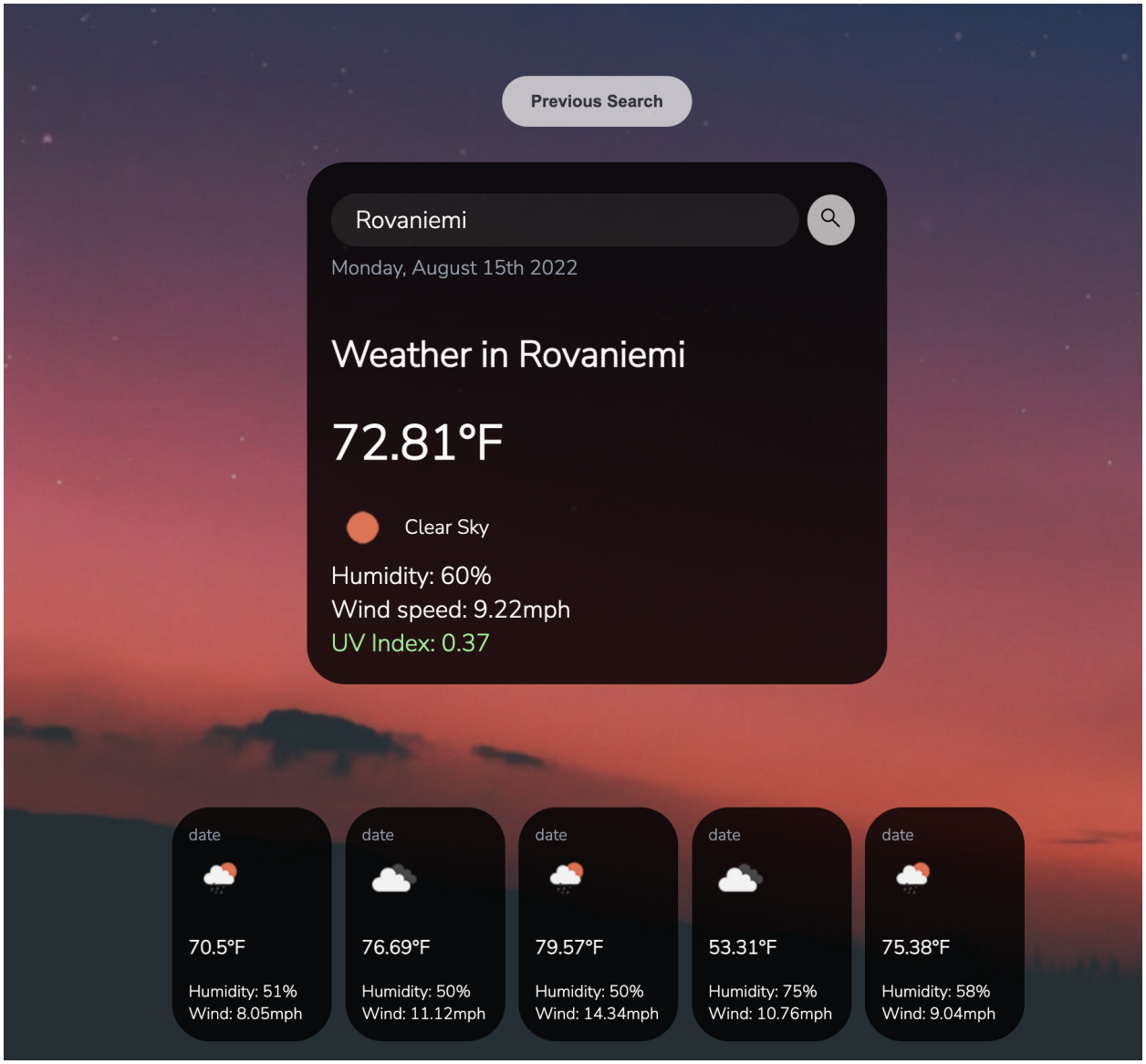
Booleans and if statements are so simple, and yet I have so much trouble with them. Before I had the chance to overthink again, I ran to youtube to brush up on my basics for if statements, so hopefully I could get one working correctly for my UV index the first time. I kept it simple at first and only used the console log before I moved on to any real output on the DOM:

```
document.querySelector(".uvindex").innerText = "UV Index: " + uvi;

if(uvi > 7) {
   console.log("high");
}
else if(uvi >= 5) {
   console.log("moderate");
}
else {
   console.log("low");
}

}
```

I kept this directly in my showWeatherData function, as this is where the color change will happen, and it is where my UV index is already declared and output. I tried multiple towns and cities across the world before I finally got a result that showed me the output was reading my numbers correctly and changing accordingly. This was all the confirmation I needed, so next I went in and actually color-coded my output text:

```
if(uvi > 7) {
   document.querySelector(".uvindex").classList.add("high");
   console.log("high");
}
else if(uvi >= 5) {
   document.querySelector(".uvindex").classList.add("moderate");
   console.log("moderate");
}
else {
   document.querySelector(".uvindex").classList.add("low");
   console.log("low");
}
```





Lastly, I had to get the dates for the five day forecast:

```
const currentDate = document.querySelector("#date");

function date() {
   var now = moment();
   $("#currentDate").text(now.format("dddd, MMMM Do YYYY"));

   var nextDay = moment(startdate = now).add(1, 'days');
   $("#forecastDate0").text(nextDay.format("MM/DD"));
   var nextDay2 = moment(startdate = nextDay).add(1, 'days');
   $("#forecastDate1").text(nextDay2.format("MM/DD"));
   var nextDay3 = moment(startdate = nextDay2).add(1, 'days');
```

```
    $("#forecastDate2").text(nextDay3.format("MM/DD"));
    var nextDay4 = moment(startdate = nextDay3).add(1, 'days');
    $("#forecastDate3").text(nextDay4.format("MM/DD"));
    var nextDay5 = moment(startdate = nextDay4).add(1, 'days');
    $("#forecastDate4").text(nextDay5.format("MM/DD"));
}
setInterval(date, 1000);
date();
```

And that’s a wrap!