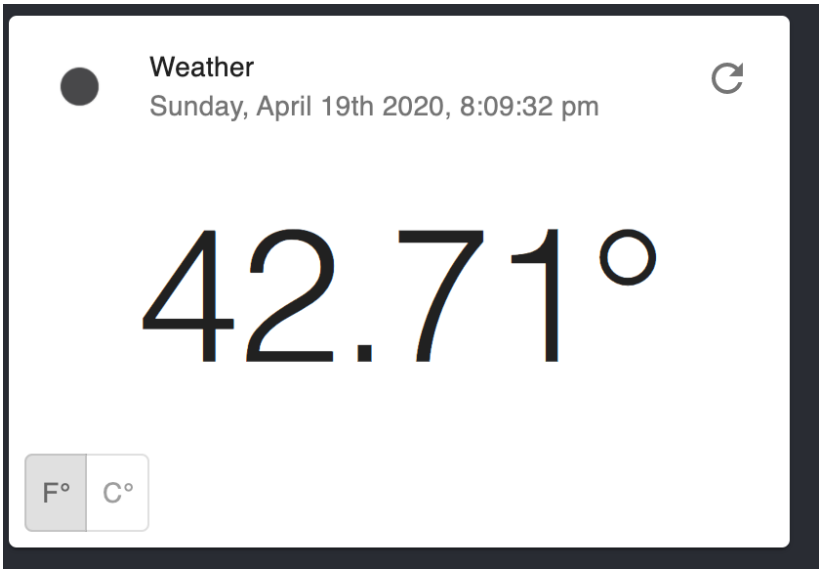
Weather Widget

We are going to be creating a React component that displays the weather for the zip code 60654. While seemingly simple, this widget will encompass multiple aspects of production grade web-applications including:

- AJAX: to get data from the server-side
- Material Design: Modern & thoughtful UX
- Accessibility: Leveraging semantic elements & ARIA

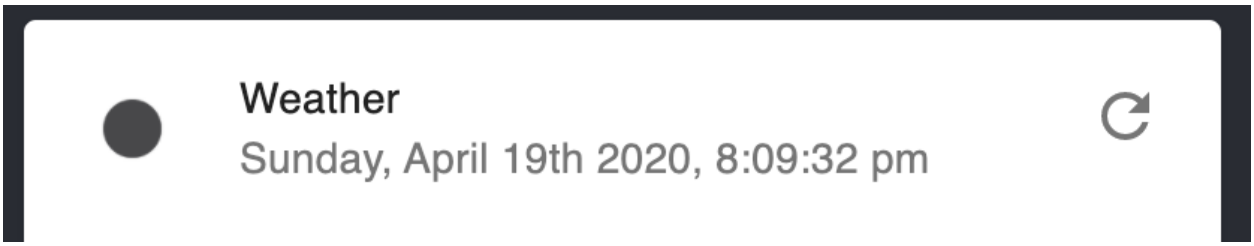
Starting ASAP is recommended.

Requirements



The application should be a material design "card" comprised of three sections – header, content, and bottom actions.

Header



The header should begin with an image representing the weather. Sunny clear weather would have an orange circle, partly cloudy would have the circle partly obscured by a little cloud, etc. In the example above, there is a clear sky at night.

The next item should be the title “Weather” and below it a description field with the time that the displayed weather was calculated/derived from. The format of the time should be `dddd, MMMM Do YYYY, h:mm:ss a` Using a date/time manipulation library like moment.js makes this easy.

Finally, right at the end, there should be a Refresh action with the appropriate Refresh icon. Clicking this icon should result in a new AJAX call being made to retrieve the most up-to-date weather information the API has available.

Content

The content of the card should be the temperature value for the zip code of 60654. It should be displayed in prominent typography and suffixed by a degree symbol.

Bottom Actions

There should only be one action on the bottom in the form of a Toggle Button that lets the user select between displaying the weather in Celsius or Fahrenheit.

Non-functional Requirements

The widget should indicate to the user when the content is loading using some kind of indicator. Two examples that can be used are a Progress Indicator (such as a linear or circulator loading indicator), or skeletons. The widget should remain on the screen in an interactive state while the loading progress is shown to the user (e.g. the entire screen shouldn't change and display one big loading indicator – it should be encapsulated within the widget.)

Accessibility

The component should pass the WAVE (web accessibility evaluation tool) assessment with 0 errors.

The component should be usable with a screen-reader. If the user selects the weather image, there should be a description available like "scattered clouds." Another example is if the toggle buttons were selected by a screen reader, the reader should indicate the following:

- That these buttons are toggle-able
- Which is currently selected
- The description of the button e.g. having the reader tell the user "F degrees" is not good accessibility. Having it read "Fahrenheit" would be more appropriate.

Testing a screen reader depends on what operating system you are using. Windows has Narrator and OS X has Voiceover.

Submission Instructions

The submission should be a ZIP file with a name scheme of "LastnameFirstname-HW3.zip"

Inside the ZIP, you should have your create-react-app generated project. **You do not need to upload the npm_modules folder (it will be massive)!** Your dependencies are already declared in the `package.json` file, so I can get them locally by just executing **`npm install`**

The way I will launch the app to grade/verify will be executing the **`npm start`** command. If your website does not launch and work correctly after executing this command within your project folder, points will be subtracted.

How to Implement

First start off with the UI design. Fill it in with mock data. Everything in this assignment can be completed by using out-of-the-box Material UI components. The *only* custom CSS I used in my entire application was to set the body background color, and set a maximum width for my Card.

Then work on obtaining the weather data. We will be using the https://openweathermap.org API. Go on their website and register for an API key – it's free. Once you have the API key, read over the API documentation https://openweathermap.org/current#zip on how to obtain weather information for a ZIP code. This is also where the weather image & description data is obtained, and it's documented on this page: https://openweathermap.org/weather-conditions

A useful tool for testing HTTP requests and viewing the responses is Postman. I recommend trying the API out first in Postman to figure out how to get your weather in Fahrenheit, Celsius, etc.

Next implement the refresh button functionality, and the temperature scale toggle. A new AJAX call should be made whenever the user conducts one of these actions:

- Toggles Celsius/Fahrenheit to a different value than the previously selected one. Tapping on a scale selection when it's already selected should not trigger another AJAX call.
- Refresh button is pressed.

Now start working on the finishing touches.

Incorporate a loading indicator or skeletons for the information on the widget that is loaded by AJAX. This would include the temperature, weather image, and the timestamp of the weather datapoint. Test whether this works as expected by simulating a slow network connection. Throttling to a profile like "Slow 3G" will make it very obvious if your loading indication is working as intended. https://developers.google.com/web/tools/chrome-devtools/network#throttle

Incorporate accessibility tags like aria-labels and image alt descriptions. Install the WAVE Chrome plugin and run it against your web page. Ensure that there are 0 errors.

Turn on your operating system's screen reader and try to navigate your website. Close your eyes - would you be able to refresh your widget, or toggle between the temperature scales? If not, then it's not accessible!

Extra Credit Opportunity

One of the things you may notice after completing the assignment is that whenever an AJAX call is made, your loading indicator or skeleton will "flash" for a very brief period of time before the new temperature is displayed. This kind of flash can be a bit distracting and unappealing to the user. Best practice is to delay showing the progress indicator. If the AJAX call is taking a bit of time, the indicator will be displayed re-assuring the user that we're working on their request. But if the AJAX call returns very quickly, the delay will never have gotten the chance to

show the indicator/skeleton and the user will be presented with the new value without an abrupt flash. This can be implemented in a fairly simple way using the approach outlined here: https://material-ui.com/components/progress/#delaying-appearance

Completing this will earn you **+2** points.