Exercise 2- Weather App

Due: March 22nd @ 8pm

In this exercise, you Will be creating a 3-part Weather app with three distinct parts to it. You will have to decide how to structure your project in order to pass data in a way that works. Take a look at the starter code to see how the components will be rendered on the same page. You may add any other relevant files to the project. **Parts 1 and 2 must use part 0 in its implementation. Lastly, individual styling (colors, images, fonts etc.) do not matter but columns must match with my example pictures.**

Part 0: WeatherComponent

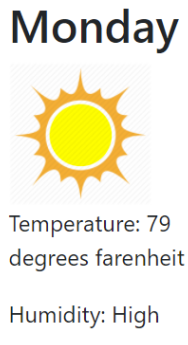
In a file called WeatherComponent.js, you should be able to render a component with the following data:

- condition (with these possibilities ***only, you choose the image that is displayed***):
  - cloudy
  - sunny
  - rainy
  - snowy
  - sleety
  - thunderstormy
  - Anything else: An error image
- temperature ***(only valid numbers will be tested)***:
  - Display the temperature using the h2 tag
  - Please ensure the data in temperature is a number, or should display ERROR (includes if it is empty)
- dayOfWeek (***Only valid, fully spelled days will be used, capitalization should not matter)***:
  - Display at the top of the component using h1 tag
- pollenCount:
  - High
  - Moderate
  - Low
  - Anything else, display ERROR

For example, if I would like to render this component with the following props:

```
<WeatherComponent dayOfWeek = "Monday" temperature = "79" condition = "sunny" pollenCount = “High”/>
```

It should render something like this:



Part 1: WeekForecast

Please create a second component called WeekForecast which will use your WeatherComponent to create a weeklong forecast with the array of Objects passed in through props.

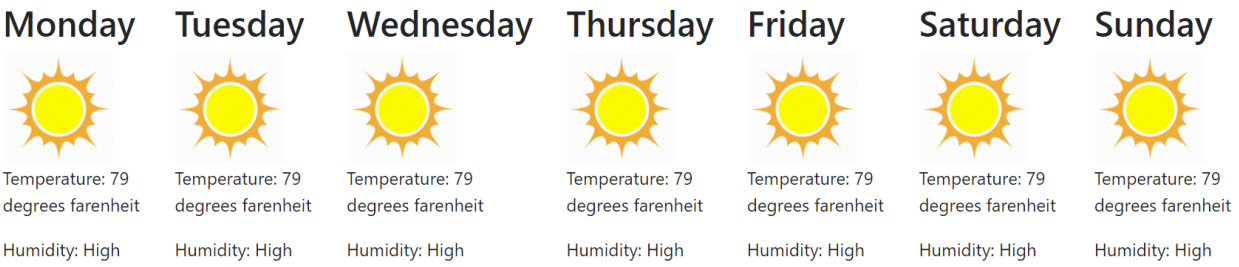
Assuming forecastInfo has been created like so:

```
let forecastInfo = [
{dayOfWeek: "Monday", temperature: 79, condition: "cloudy", pollenCount:"High"},
{dayOfWeek: "Sunday", temperature: 79, condition: "sunny", pollenCount:"High"},
{dayOfWeek: "Tuesday", temperature: 79, condition: "sunny", pollenCount:"High"},
{dayOfWeek: "Wednesday", temperature: 79, condition: "sunny", pollenCount:"High"}
,
{dayOfWeek: "Friday", temperature: 79, condition: "sunny", pollenCount:"High"},
{dayOfWeek: "Saturday", temperature: 79, condition: "sunny", pollenCount:"High"},
{dayOfWeek: "Thursday", temperature: 79, condition: "sunny", pollenCount:"High"}]
;
```

In index.js, you would render the following component:

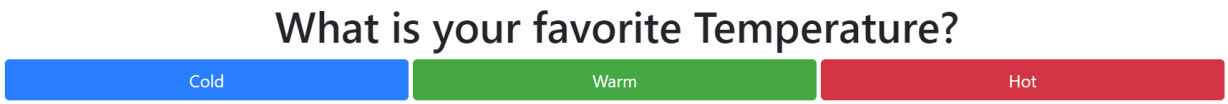
```
<WeekForecast data = {forecastInfo}/>
```

It should now render each of the days and images after each other on the page. For example, the above example should render the following:



Part 2: BestDay

In a component called BestDay.js, you will ask the user to click which type of temperature they enjoy the most like so:



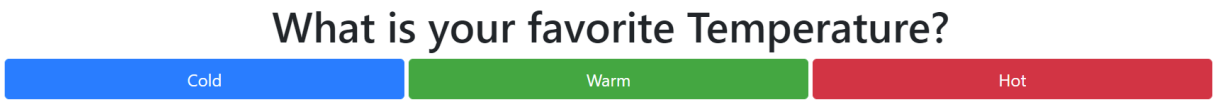
It should render using the following tag (may use the above array in part 1 as a test case):

```
<BestDay data = {forecastInfo}/>
```

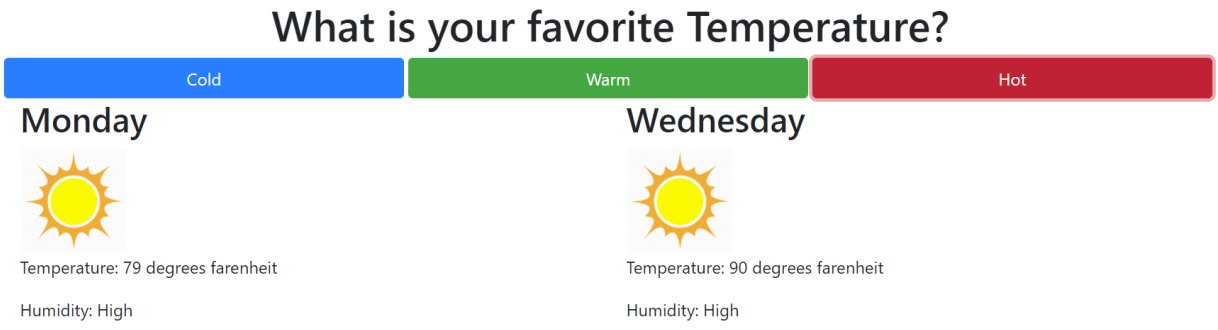
The conditions are:

- If the user clicks “Cold” it should display any WeatherComponent that has a temperature less than or equal to 32 degrees
- If the user clicks “Warm” it should display any WeatherComponent that has a temperature between 33 and 70 degrees (inclusive)
- If the user clicks “Hot” it should display any WeatherComponent with a temperature of over 70

For example, your part two should start like so:



Then, when I click “Hot” it should automatically display (assuming only Monday and Wednesday are above 70 degrees farenheit):



Then, if I click a different button, it should replace the existing days (Monday and Wednesday in this case) with the applicable days. If no days are applicable, display something that lets the user know that no days this week are applicable for that temperature preference.