CS 295R
Lab 4 – Weather

The objective of this lab is to allow you to continue to develop your understanding of data flow between components in React applications. You will also have the opportunity to work with controlled (form) input and to use axios to make an AJAX get request. The ES6 version of the weather app from CS233JS should be used as the starting point for this portion of the lab.

Overall Process

1. Familiarize yourself with the geolocation and weather apis from openweathermap.org.
2. Use create-react-app to create a template for the application.
3. Add axios and bootstrap to the application.
4. Remove all of the unnecessary parts of the generated code.
5. Add a utilities folder to your project. Move dates.js and weatherParsing.js from the ES6 version into that folder.
6. Add api.js to the utilities folder. Create and export async functions getLocation and getWeather.
7. Add a components folder to your project.
8. Create the ZipForm component. In should receive onSubmit as a prop. It should also save the zipcode in state using a controlled (form) component and pass the zipcode as a parameter to onSubmit to make the zipcode available to the App component. Use it render an instance of the ZipForm on the screen in the App component.
9. Add handleSubmit to the App component. Call getLocation and then getWeather and save forecast, city and selectedDay in state. Test.
10. Create the CurrentDay component. It should receive city and forecastDay as props and render JSX. Add an instance of CurrentDay to the JSX for the App component. Pass it the city as well as the first element in the forecast array as props. Test.
11. Add handleDayClick to the App component. It should take an index as its parameter. It should set the selectedDay in state.
12. Create the WeatherListItem component. It should receive onDayClick, index and forecastDay as props. It should declare a nested function expression handleClick that calls the function passed as the onDayClick prop with index as its argument. Add an instance of WeatherListItem to the JSX for the App component. Pass it the first element in forecast as the forecastDay prop. Test.
13. Create the WeatherList component. It should receive forecast and onDayClick as props. It should call map on forecast to render and instance of a WeatherListItem for each element in forecast. Remove the single WeatherListItem from App and replace it with an instance of WeatherList. Pass it the forecast and handleDayClick as forecast and onDayClick props. Test.

14. Change the JSX in the App component to conditionally render the CurrentDay component only when the selectedDay is not null. Test.
15. Create a production version of the app. Deploy the production version to citstudent.

The Details

1. Familiarize yourself with the geolocation and weather apis from openweathermap.org.
    a. Register as a developer and get a FREE api key from openweathermap.org.
    b. Comments on line 6 – 9 in weather.js in the ES6 version provide a sample api call for both the geolocation and weather api at openweathermap.org.
    c. Test the geolocation api in a browser. Familiarize yourself with the url pattern as well as the returned data.
    d. Test the weather api in a browser. Familiarize yourself with the url pattern as well as the returned data. I have provided you with a function called parseForecast (in weatherParsing.js) that will transform the array of 40 forecast elements (5 days at 3 hour intervals) into an array containing a 4 day simplified forecast.
2. Use create-react-app to create a template for the application.
3. Add axios and bootstrap to the application.
4. Remove all the unnecessary parts of the generated code.
5. Add a utilities folder to your project. Move dates.js and weatherParsing.js from the ES6 version into that folder.
6. Add api.js to the utilities folder. Create and export async functions getLocation and getWeather.
    a. Import axios.
    b. Import parseForecast from weatherParsing.js
    c. Declare a global constant apikey and set it equal to appid= followed by your apikey for openweathermap. My apikey can be found in weather.js in the ES6 version of the app.
    d. Export a function expression called getLocation. Mark it as async because it makes an AJAX call. It should take zipcode as its parameter.
        i. The body of the arrow function should contain a local constant, geoUrl. Its value can be copied from weather.js in the ES6 version.
        ii. Declare a local constant called response. Set it equal to the return value from calling axios.get with the geolocation api url. An ES6 template literal with the api url can be found in onFormSubmit of weather.js in the ES6 version. The call should be marked with await.
        iii. Return a json object with name, lat and lng as properties. The property values should come from the response.
    e. Export a function expression called getWeather. Mark it as async because it makes an AJAX call. It should take lat and lng as its parameters.

i. The body of the arrow function should contain a local constant, weatherUrl. Its value can be copied from weather.js in the ES6 version.

ii. Declare a local constant called response. Set it equal to the return value from calling axios.get with the weather api url. An ES6 template literal with the api url can be found in onFormSubmit of weather.js in the ES6 version. The call should be marked with await.

iii. Return the return value from calling parseForecast passing the data.list property from the response as its argument.

7. Add a components folder to your project.
8. Create the ZipForm component.
    a. It should receive onSubmit as a prop.
    b. Create the state variable zipcode and its mutator using the useState hook. Initialize zipcode to an empty string.
    c. Declare a nested function expression called handleChange. It should take event as its parameter. Use the mutator function for zipcode to set the zipcode to event.target.value.
    d. Declare a nested function expression called handleSubmit. It should take event as its parameter. It should ensure that the form is not submitted to the server for processing and should then call the function passed to the component as the onSubmit prop using zipcode as its argument.
    e. Return JSX to render the form. Html for the form can be found in the index.html file in the ES6 version. The value prop for the input control should be set to the state variable zipcode. The onChange event should be associated with the handleChange function expression.
    f. Use it render an instance of the ZipForm on the screen in the App component. Pass handleSubmit as the onSubmit prop to the component.
9. Add handleSubmit to the App component. It should be an async function expression that takes zipcode as its parameter. In the body of the arrow function
    a. Create try / catch blocks.
    b. In the try block
        i. Call getLocation passing zipcode as its argument. Mark the call with await and set the return value to a local constant city.
        ii. Call getWeather passing city.lat and city.lng as arguments. Mark the call with await and set the return value to forecast.
        iii. Use the mutator functions to set the city and forecast state variables to the return values from the function calls above.

iv. Use the mutator function for selectedDay to set the selectedDay state variable to null.

v. Console.log the city and the forecast for testing purposes.

c. Use the following error handling code in the catch block

```
if (error.response) {
  // 5xx or 4xx error
  console.log(error.response.data);
  console.log(error.response.status);
  console.log(error.response.headers);
}
else if (error.request) {
  // request never left
  console.log(error.request);
}
else {
  // anything else
  console.log(error.message);
}
```

d. Test. You should be able to enter a zipcode and send off both AJAX requests. The city and the forecast array should be logged in the console. You should also be able to introduce an error in either api url and see an error in the console as well.

10. Create the CurrentDay component.

    a. It should receive city and forecastDay as props.

    b. It should return JSX for the detailed weather forecast for one day. Sample html for the component can be found in renderCurrentDay in weather.js in the ES6 version.

    c. Add an instance of CurrentDay to the JSX for the App component. Pass it the city as well as the first element in the forecast array as props.

    d. Test. You'll need to add a conditional expression that only renders the CurrentDay when the forecast is not an empty array for testing at this point in the process.

11. Add handleDayClick to the App component. It should take an index as its parameter. It should use index to set the selectedDay in state.

12. Create the WeatherListItem component.

    a. It should receive onDayClick, index and forecastDay as props.

    b. It should declare a nested function expression handleClick that calls the function passed as the onDayClick prop with index as its argument.

    c. Add an instance of WeatherListItem to the JSX for the App component. Pass it the first element in forecast as the forecastDay prop.

d. Test. You'll need to add a conditional expression that only renders the WeatherListItem when the forecast is not an empty array for testing at this point in the process.

13. Create the WeatherList component.
    a. It should receive forecast and onDayClick as props.
    b. It should call map on forecast to render and instance of a WeatherListItem for each element in forecast. Each WeatherListItem needs a key prop set to the dt property of the forecastDay, a forecastDay prop, an index prop and an onDayClick prop.
    c. Remove the single WeatherListItem from App and replace it with an instance of WeatherList. Pass it the forecast and handleDayClick as forecast and onDayClick props.
    d. Test.
14. Change the JSX in the App component to conditionally render the CurrentDay component only when the selectedDay is not null. Test.
15. Create a production version of the app. Deploy the production version to citstudent.
    a. Create a file called .env in the main folder for your application.
    b. Add one line in the file that assigns the predefined environment variable PUBLIC_URL to ./ This value will be used rather than / in the index.html file in script and link elements for your application.
    c. npm run build to create a production build of the application.
    d. Test the production build by launching index.html from the build folder on your machine.