The Weather App

VIMOSAN NANTHAKUMARAN

Table of Contents

INTRODUCTION

This is a simple React weather application that displays a 5-day forecast in 3 hour intervals for a user inputted city. Weekly averages statistics such as pressure, temperature, and wind are also displayed for the city. A graph of the maximum and minimum temperatures for that 5-day forecast and a graph of the maximum and minimum wind speeds for that 5-day forecast are also displayed.

INSTALL & LAUNCH

Please first ensure you have NodeJS installed.

To install all the required modules for this application, navigate to the root folder of this application from command line and run **npm install.**

After the required modules have been installed, run **npm start** to launch the application on port 3000.

FILE STRUCTURE

- Modules can be found in \node_modules
- Main index.html file can be found at \public\index.html
- Font required for ErikFlowers' weather icons can be found at \public\font
- CSS file required for ErikFlowers' weather icons can be found at \public\weather-icon.css
- Main index.js (React DOM render) can be found at \src\index.js
- OpenWeatherMap base API instance can be found in \src\api
- All other components can be found in \src\components
- JS exported functions can be found in \src\components\functions
- CSS files can be found in \src\components\css
- Image files can be found in \src\components\images

GETTING STARTED

This Weather App consists of 9 separate components, excluding index.js which renders the main component (App.js) to the DOM. Comments outlining details are also included in the source code to supplement this documentation. This section will give you a brief overview of how this project is put together.

OpenWeatherMap

This React Weather App uses OpenWeatherMap's 5 day / 3 hour forecast API (https://openweathermap.org/forecast5) to display the weather based on city inputted by the user.

ErikFlowers Weather Icons

The icons used in this application are not the icons provided by the OpenWeatherMap API. A third party set of icons which are mapped to weather codes from OpenWeatherMap are used. The fonts required for these icons can be found in the fonts folder (\public\font), and the CSS file required can be found in the public folder (\public\weather-icons.css). This CSS file (internal) is linked in the **index.html** file (\public\index.html). Documentation for these icons can be found at https://erikflowers.github.io/weather-icons/.

Google Places Autocomplete API

The city input by the user is determined with the help of Google's Places Autocomplete API (https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete-addressform) and the user's current geolocation. The script for this integration can be found in the **index.html** file (\public\index.html).

React-Spring

The animations for this project are created using React Spring (https://react-spring.surge.sh/). Animations are applied to each component of this project, and all the components have the same animation. React Spring is used to fade in all the components (opacity 0 to opacity 1). The React Spring tags wrap the JSX to be returned from each component. For more details on React-Spring please see the documentation found in the link provided.

Semantic-UI

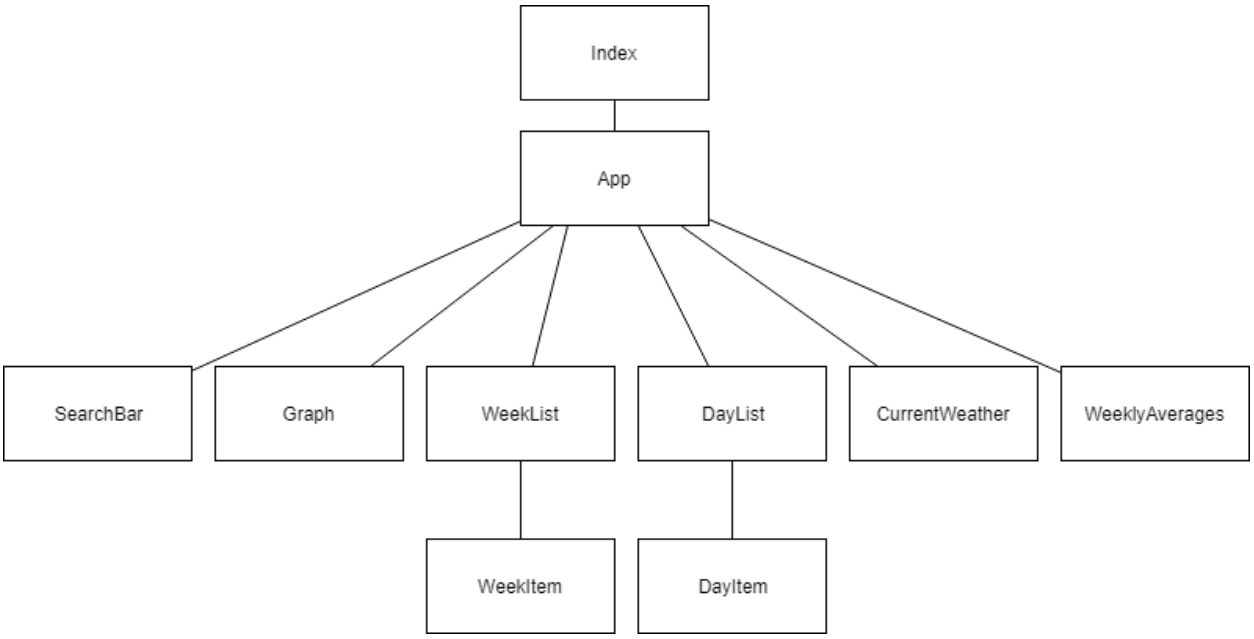
Semantic UI is used to help structure the components on the page, and provide for a responsive design. The CSS file (external) is linked in the **index.html** file (\public\index.html). The documentation for Semantic UI can be found at https://semantic-ui.com/.

Components Overview

The components and subcomponents used to create this app are:

- App
- SearchBar
- Graph
- WeekList
    - WeekItem
- DayList
    - DayItem
- CurrentWeather
- WeeklyAverages

Component Hierarchy



Views & Styling

There are 2 different views which can be rendered from the parent App component. When there is no user inputted city (when the site is first loaded), a landing page is displayed (landingPage.css). Which displays only one component, the SearchBar. When there is a user inputted city another view is rendered which contains all the components listed above. Different styling is displayed depending on the current temperature. If it is warm (above 5 degrees) classNames with warm colored styling are applied (warm.css), if temperature is 5 degrees or below, classNames with cold colored styling are applied (cold.css). These components, along with important functions will all be discussed in this documentation.

COMPONENTS

App

INTRODUCTION

This component houses the state for the entire application, which will be passed as props to the required components. This component is also where the API function call (onTermSubmit) is placed, however it is not invoked from the App component. It is passed as a prop to the SearchBar component where it is invoked. This component is responsible for handling the rendering of all the other components. If there is currently no city selected, a landing page is displayed with styling from **landingPage.css.** If a city is selected and the temperature is above 5 degrees, all the components are rendered and classNames with warm styling are applied (**warm.css**). If a city is selected and the temperature is 5 degrees or below, all the components are rendered as well, except the classNames with cold styling are applied (**cold.css**). The temperature at which the styling changes can easily be adjusted if necessary.

STATE

The state of this App consists of 4 properties. These 4 properties are:

- weather
- weatherSplit
- selectedDay
- graphKey

The state of these are set when the API call (discussed in further detail below) is made.

FUNCTIONS

onTermSubmit(term)

Please note other parameters not discussed in this documentation can be specified during the API call to get responses of different formats. Please see OpenWeatherMap documentation for more details (https://openweathermap.org/forecast5).

Function Call/Parameters

This function is what handles the API call when called from the SearchBar component. The API calls are handled using the **Axios** package (https://www.npmjs.com/package/axios). The parameters we will specify for this API call are **APPID**, **units**, and **q**.

The API instance can be found in '\src\api' folder. The API instance specifies the base URL and some parameters. The parameters specified in the instance are the API key (**APPID**), obtained from OpenWeatherMap, and the units (**units**) which is set to metric.

The onTermSubmit function takes a parameter called '**term**'. This term should be of the form CityName, CountryCode (Example: Toronto,CA). When onTermSubmit is called it makes a GET request to the API with the parameter **q** set to the value of 'term'. This term will be determined from the user input and will be discussed in further detail in the SearchBar component documentation.

Response

The response from this API call is an array of JSON objects (by default, but can be changed if necessary) consisting of weather data for 5 days in 3 hour intervals.

Updating State

After the response is received, the states of this component are updated.

- The **weather** state is set to the entire array received from the response of the API call. This is only updated again once another API call is made by the user inputting another city using the SearchBar component.
- The **weatherSplit** state is set to the response as well but split into days of the week. This array may be referred to as **days.** It is an array in which each element consists of another array (referred to as **day**) containing the weather data for that day (in 3 hour intervals). Each element of this day array may be referred to as **interval**. This is only updated again once another API call is made by the user inputting another city using the SearchBar component.
- The **selectedDay** is set to default as the first day of the weatherSplit array. The selectedDay state indicates which day is currently selected, it will be a single element from the weatherSplit array. In other words, the selectedDay is a **day** (see above) containing the weather data in 3 hour intervals. Again, the entire array will be referred to as a **day** and each element will be referred to as an **interval**. This state can be updated using the **onDaySelect** function (discussed below). This function is passed as a prop to the WeekItem component where it will be invoked.
- The **graphKey** state is set to a random number. This random number updates each time a new API call is made. The purpose of this is to update the graphs when a new city is entered. This is only updated again once another API call is made by the user inputting another city using the SearchBar component.

onDaySelect(day)

Function Call/Parameters

This function is simply used to update the state of selectedDay in the App component. It takes only one parameter, **day**. This day parameter is the same one discussed above (updating state), and will be an array consisting of elements that contain the weather data in 3 hour intervals. Each element is the weather data for a 3-hour span, a new element is given for each 3-hour interval. This function can be passed to child components where it can be invoked in order to change the selectedDay state. In this project, it will be passed to the WeekItem component. This will be discussed in further detail in the WeekItem component documentation.

getDayOfWeekArray(days) – *imported function*

Function Call/Parameters

This function is used to get an array consisting of the days of the week. It takes an array of **days** (see **onTermSubmit**) and returns an array where each element represents the day of the week for each element of the **days** array that was used as the parameter.

For instance, if one passed in a **days** array which had data for Sunday, Monday and Tuesday (ie. days array would have 3 elements) this function would return ['Sunday','Monday','Tuesday']. This function was made to be reusable when necessary and is used multiple times in this app.

getMinMaxValue(days,property1,property2,maxmin='min') - *imported function*

Function Call/Parameters

This function is used to create an array of maximum or minimum values for a specified property in the **days** (see **onTermSubmit**) array, which as expected, should be passed as the **days** parameter to this function. This function was created to be used in the Graph component (which will be discussed in further detail in the Graph component documentation) and is used multiple times in this project. The **property1** parameter is the parent property and **property2** is the child property (see OpenWeatherMap documentation for property names). **Maxmin** property should be set to either 'max' or 'min' depending on whether you want to obtain the maximum or minimum values of the property. This property is set to 'min' by default.

SearchBar

INTRODUCTION

This component is a simple form with one input field. This input field has Google Places API (autocomplete) integration which allows for autocomplete as the user types a city. The integration has been modified to suggest only cities. Geolocation is also used to enhance this autocomplete. If geolocation is enabled the autocomplete will suggest cities closer to the user's location circle. The Google Places API (including geolocation) documentation link can be found in the Getting Started section of this documentation.

PROPS

This component takes the onTermSubmit from the parent App component as a prop, the prop name is kept the same, onTermSubmit. As a reminder, this function is what handles the API call. This function will be invoked in this SearchBar component.

STATE

The state of this SearchBar component consists of only one property, **term**.

The value of this state property is set when the user selects a city from the dropdown (autocomplete integration). It is of the form CityName, CountryCode for example Toronto,CA. This state property is what will be used to invoke the onTermSubmit function (Ex: onTermSubmit(Toronto,CA)).

This state property is only used as a parameter for the API call (onTermSubmit) in this project, however it can be used elsewhere if needed in future modifications of this project.

FUNCTIONS

OnFormSubmit()

Function Call/Parameters

This function does not take any parameters. This function simply prevents the default form submission action. When enter is pressed from the form (input field), a page reload occurs (default action). This function prevents that page reload. This function is passed as the onSubmit callback to the form field which is rendered in this component.

handlePlaceChanged()

Function Call/Parameters

When the user begins typing into the SearchBar component's input field Google Places API autocomplete creates a dropdown list of suggested cities This function is the callback function that is invoked when a city from the dropdown is clicked. This function sets the SearchBar component's state property '**term**' to the CityName,CountryCode of the city chosen from the dropdown list. It then invokes the onTermSubmit function (see App component for more details) which was passed as a prop to this

SearchBar component. This onTermSubmit function is called with its parameter set to the state property '**term**', which was set when the user selected an item from the dropdown. This parameter, **'term'**, will be of format CityName,CountryCode which is what is required by the API call that will be made in the onTermSubmit function. This function does not take any parameters.

geolocate()

Function Call/Parameters

This function is used to get user's location. This location will be used with the Google Places API to provide more accurate suggestions (cities close to user location). Uses the browser's 'navigator.geolocation' object, and passes location data to required objects of the Google Places API. See Google Places API documentation for more details. This function is invoked when the SearchBar component mounts.

Graph

INTRODUCTION

This component is used to render a line graph from data that is passed as a prop to this component. Some of the data (arrays) that will be passed as props to this component are created using the functions getDayOfWeekArray and getMinMaxValue, which are described in detail in the App component section of this documentation. This component uses the **ChartJS** package to render the graph. More information on ChartJS can be found at https://www.npmjs.com/package/react-chartjs-2.

The config object found in this component is what is used to define the configuration for the line graph. This config object can be used to define things like titles, scales, legend, colors, etc. Please see the ChartJS documentation for more information on what can be configured.

PROPS

This component uses the data passed to it as props from the parent (App) component to render the line graph. This component takes a total of 7 props:

- key
- label
- data1label
- data1
- data2label
- data2
- title

The **key** prop is a randomly generated number from the parent component's state (App.js graphKey). A new key is only generated when a user selects a new city, this is to allow ChartJS to rerender when a new city is selected.

The **label** prop should be an array containing the labels of the graph, which will be the **x-axis values**. In our project we use the getDayOfWeekArray function (see App component documentation) to generate an array containing the days of the week from the API call. So, the x-axis of the graph will be days of the week (ie. Sunday Monday Tuesday Wednesday).

In this project each line graph component will have 2 lines on it, thus requiring two datasets. One dataset will be for maximum values and one will be for minimum values. The props **data1** and **data2** are the datasets that will be used for the graph. These data sets, similar to the label prop will be an array containing the data, and will be the **y-values** for the graph. The arrays for **data1** and **data2** are created using the getMinMaxValue function (see App component documentation). This function is invoked with different parameters (ie max and min) to generate the required datasets. The getMinMaxValue function allows one to quickly and easily generate a dataset array based on any property desired so that it can be passed as a prop to the graph component which can then easily render a graph. Please see the App component documentation for more details on this function. The props **data1label** and **data2label** are just strings that are labels for the respective datasets, for instance one could be labelled 'Max Wind Speed', and the other 'Min Wind Speed'.

The **title** prop is a string that specifies the title of the graph to be rendered.

STATE

The state of this Graph component has only one property, '**chartData**'. This property is what stores the data required to render the graph. It includes both datasets, their labels, and some configuration for the datasets (datapoint colors, etc). These are the Y values for the graph. The chartData state also includes the data for the X values, which is called "labels". These chartData values are set using the data passed as props to this component. More information about the chartData state can be found in the ChartJS documentation (see Intro section of Graph component documentation).

WeekList/WeekItem

INTRODUCTION

WeekList and WeekItem are related functional components. WeekList is the parent component and WeekItem is the child component. The WeekList component maps through an array of **days** (prop passed to Weeklist component) and creates a new array containing a WeekItem component for each day of the days array. It then renders this new array (list). The Weekitem component displays the data for each day, the data displayed will be the first returned element (**interval**) for each **day** from the OpenWeatherMap API (see App component documentation)**,** however this can easily be modified if necessary. Each WeekItem is also selectable, and will cause a state change (**selectedDay**) in the parent component to the WeekList component. When a WeekItem (day) is selected, a visual change will indicate that the item has been selected.

PROPS

The WeekList component is passed 4 props:

- days
- DayOfWeekArray
- onDaySelect
- selectedDay

The **days** prop is the weatherSplit array from the parent component (see App component documentation). As a reminder this array has an element for each day returned from the OpenWeatherMap API, and each element consists of another array containing the **intervals** for each day, where each interval contains the weather data for a 3-hour span. This **days** array is what the WeekList component will map through. Each element, referred to as **day**, of the days array will be passed as a prop to the child component WeekItem, the prop name will be set to **day.** The WeekItem component will use this prop and display the first element's, or first interval's weather data of that day. Again, instead of the first element, we can easily choose second or third element to get weather data of an interval later on in the day (Please note – this may cause issues as OpenWeatherMap sometimes only returns a single interval if the day is almost over). When mapping the days array, a **key** is also passed to the WeekItem as a prop as a unique identifier. The key value is set to the dt_txt (datetime) property of the first interval of that day, this allows for each WeekItem to have a unique key.

The **DayOfWeekArray** is the return of the function getDayOfWeekArray called with the parameter weatherSplit. Essentially, DayOfWeekArray will be an array containing the days of the week that are returned from the OpenWeatherMap API. Elements from this array will be passed down to WeekList's child component, WeekItem. This is in order to display the day of the week in the WeekItem component. Since this prop is passed to the child component while mapping the days array (see above), we can index the DayOfWeekArray accordingly so that the correct day of week (element) will be passed to the child component.

The **onDaySelect** prop that is passed to the WeekList component is passed down further to the child component WeekItem, the prop name is kept consistent. The onDaySelect prop is a function that takes a

parameter, **day,** and sets the state of **selectedDay** in the parent component (App) to that day. This function, as mentioned above, is invoked when a WeekItem is selected (onClick).

The **selectedDay** prop is the state property from the App component. Its value is set to a single **day**. By default, it is selected to first day of the weatherSplit array (see App component documentation for more details). This selectedDay prop is passed down further to the child component, WeekItem, the name of the prop is maintained. In the WeekItem component, the selectedDay prop is used to compare the day object of the Weekitem, which was passed from WeekList when mapping the weatherSplit array, to selectedDay object. Depending on the result of the comparison, a specific class is applied to the rendered element to indicate whether it is selected or not.

DayList/DayItem

INTRODUCTION

DayList and DayItem are related functional components. DayList is the parent component and DayItem is the child component. The DayList component maps through an array of a single day, **selectedDay**, which is passed as a prop to this DayList component with the name **day**. It creates a new array containing a DayItem component for each **interval** (see App component documentation) of the selectedDay array. It then renders this new array (list).

The DayItem component takes this interval as a prop displays the weather data for that interval.

DayList is only rendered if the prop day (selectedDay) has two or more elements. If the selectedDay only has one element (one interval) it is not displayed, since this data will already be displayed on WeekList/WeekItem component and on the CurrentWeather component. It would be redundant if included again in this component.

PROPS

This component has only one prop called **day**. This day prop is the selectedDay array from the App component's state (see App component documentation). This day array has an element containing data for each 3-hour **interval** of that day. This day array is what the DayList component will map through. Each element, referred to as an **interval**, will be passed as a prop to the child component DayItem. The prop name will be kept consistent. The DayItem will use this prop and display the data for that interval. When mapping through the day array, a **key** is also passed to the DayItem as a prop as a unique identifier. The key value is set to the dt (datetime) of each interval, this allows for each DayItem to have a unique key.

WeeklyAverages

INTRODUCTION

This component is also very simple. It calculates weekly average values for some weather properties, such as temperature, wind speeds, etc.

PROPS

This component has only one prop, **weather**. The weather prop is just the weather array from the App component's state (see App component documentation). This prop is used for the calculation of weekly averages.

FUNCTIONS

getAverage (weather,property1,property2)

Function Call/Parameters

This function, as the name implies, loops through the array that is passed to the function, in this case the weather array (all the weather data for the city), and calculates the average of a specified property. Property1 is parent property and Property2 is the child property of each array element (see OpenWeatherMap documentation for property names). This function was created to easily calculate weekly averages for specified properties that will then be rendered to the DOM.

CurrentWeather

INTRODUCTION

This component is very simple; it only displays the current (most recent) weather data. It shows an icon representing the current weather, the temperature and description.

PROPS

This component has only one prop, **today**. The today prop is just the first element of the weather array from the App component's state (see App component documentation). The first element (or the first interval) represents the earliest weather data available from the OpenWeatherMap API.