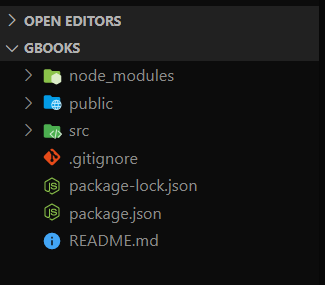
**Gbooks search engine app notes**

This app uses the google books API to produce a book search engine, where you can type in your search and have all the related books displayed for you.

**App creation:**

To create the app, I used the command **npx create-react-app gbooks.** This command sets up the development environment so that you can use the latest JavaScript features, provides a nice developer experience, and optimizes the app for production. In order to use this command the node version has to be >= 8.10 and the npm >= 5.5. The versions I used for this project are node: v14.15.0 and npm: 6.14.8.

**Link to Github:**

Once the app was created and I verified that it ran properly by typing the command **npm run** in my terminal, I proceeded to initiated as a local git repository by typing the command **git init**, this allowed me to then bridge the local repository to the one I previously created on github.

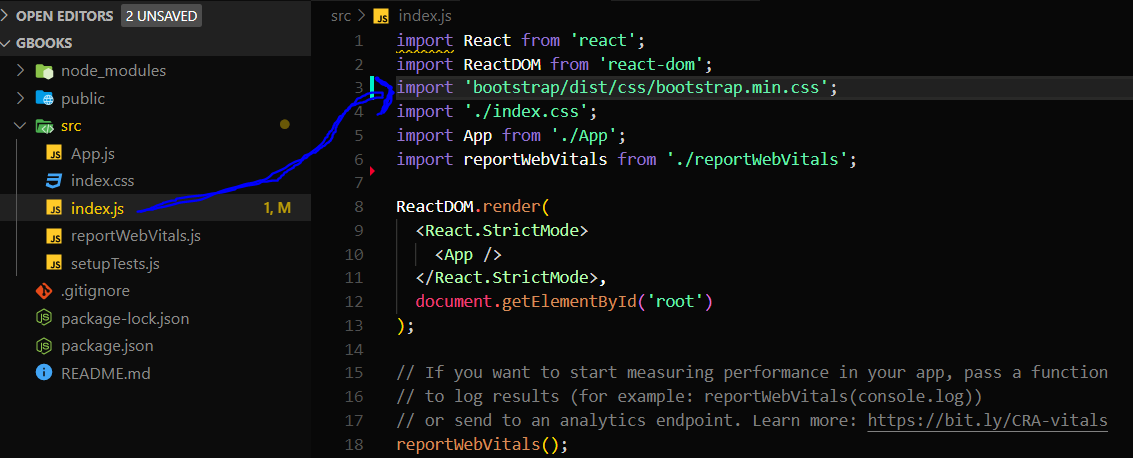
**Removing unnecessary files:**

There are some default files that were added when I created the project that I decided to remove because I ddin’t need them anymore. Files like logo.svg, App.css and test.js.

**Bootstrap:**

Bootstrap is a front-end framework for faster and easier web development. It includes HTML, CSS and JS based design templates for creating common user interface components like forms, buttons, navigations, dropdowns, alerts, modals, tabs, accordions, carousels, tooltips, and so on.

I am using bootstrap in this project, and the best way to do that is by adding a third party library called **reactstrap** [**https://reactstrap.github.io/**](https://reactstrap.github.io/)**.**

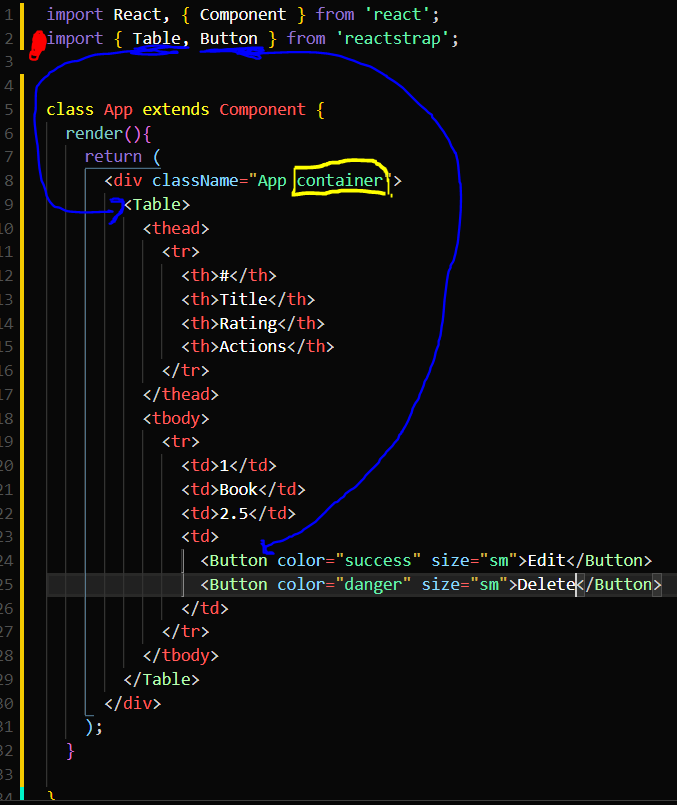
Once the installation process was done, I imported bootstrap inside the index.js file. Important to notice that I put the import statement before the index.css import statement. This is because I may want to override some bootstrap.

**App.js file**

The first file I worked on was the App.js file. The next picture shows the file with no modifications

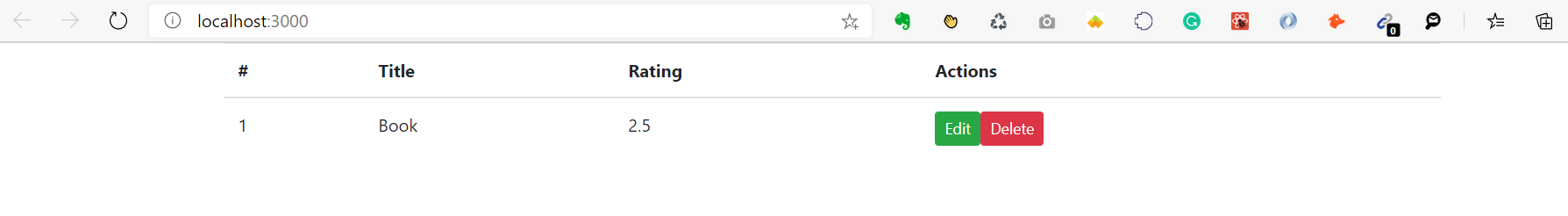
**Text

Description automatically generated**

In order to use the reactstrap components, I need to first import the library into the App.js file.

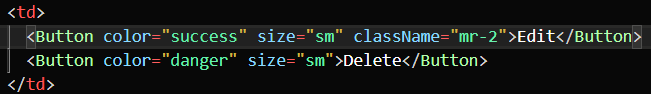
This picture illustrates the few modifications we did within this file. First, in red, we have the import statement for a couple of components that belong to reactstrap, Table and Button, which following the blue arrows we can see where they are being used. The third thing added is a “container” class outlined in yellow.

Every time we make changes to the app, it is good practice to run it and see the progress made or solve any errors that might throw at us. This is how our app looks at this stage, after we run the command **npm start** in the terminal.

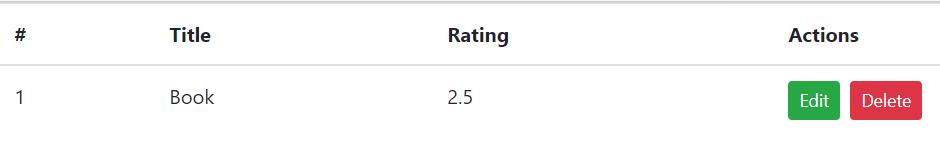


Everything is running well, but the buttons are too close to each other. Bootstrap has these utility classes that allow you to give things margin on the fly, so I used “mr-2” to define a margin room of 2px between the buttons, and this is the code and the outcome in the browser:

**Code:**



**Browser:**



**Populating the table**

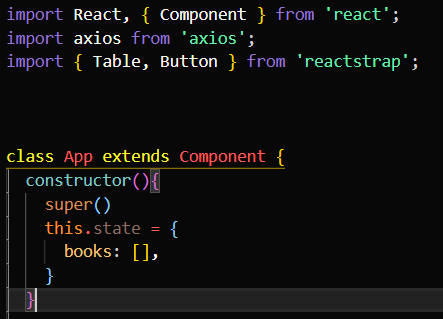
Before moving on, I decided to populate the table.

**Adding the State**

The state object is where we store property values that belongs to the component. When the state object changes, the component re-renders.

So far, our state only has an array variable called books, which is where we will store the data returned to us from the restful API.

Let’s look at our state:



In order to populate the “books” variable, we have to make a request to an external source, and then use that response to update “books”. Before we make the call, I need to define during what stage of the app I want to make the call. In React, there are methods called **lifecycle methods**  and they are methods that define different stages of the app, and we can use them to produce different results within them.

Every component in React goes through a lifecycle of events: Mounting, Updating and Unmounting.

Lifecycle methods:

**componentDidMount()** is called as soon as the component is mounted and ready. This is a good place to initiate API calls if you need to load data from a remote endpoint.

**componentDidUpdate()** is invoked as soon as the updating happens. The most common use case for this method is updating the DOM in response to prop or state changes.

**componentWillUnmount()** is called right before the component is mounted and destroyed. If there are any cleanup actions that you would need to do, this would be the right spot.

Ok this are the main three methods and for now we are using the **componetnDidMount()** to make our API call.

To make the API call we are using **axios** <https://www.npmjs.com/package/axios>. **Axios** is a library that helps us make http requests to external resources. In our case, we want to retrieve data from an API so that we can display it in our app. There are several ways to do this, but the reason we are using **axios** is because it is designed to handle http requests and responses. It’s used more often than the JavaScript API fetch, because it has a larger set of features and it supports older browsers.

