Table of Contents

PART 01 – REACT SETUP

Skip this part if you are using Vite to create your React application. Move directly on to part 02. For a vite-based install check out Appendix E.

1. Choose a folder that will become the parent of your project folder eg: /Documents

2. Open a terminal window pointing to the Documents folder.

3. Navigate into that folder (from #1) and run the command **npx create-react-app react-bc** from a terminal window pointing to your chosen folder.
   If you get a dependency conflict try adding --legacy-peer-deps at the end of the command

4. You may be asked to install packages, enter Y to proceed

5. After you CD into the react-bc folder run the command >**`npm start`**, your browser should open and you should see the default React UI. Close the running process using CTRL-C.

6. Open the **`react-bc`** folder in VSCode, view the code of app.js, this is the file that feeds the default page that shows up on the browser at port 3000. Change the text between the anchor tags, so change *Learn React* to *Learn React Now* or something similar, hit **`save`**. Note, if you closed the app fro #5, just re-open it in VS Code, using the embedded terminal window interface.

PART 02 – ADDING COMPONENTS

1. Create a folder called **`components`** inside the **`src`** folder

2. Inside of the **`components`** folder, create a new .js file called header.js
   Once the header.js file opens, import *react* and begin writing a function:

```
import React from "react";//note it is not necessary to add this line
anymore, but I left it in as it will appear in some legacy code

function Header(){

};
```

3. Finish the function by returning the component

```
function Header(){
  return (
    <header>
      This is the header
    </header>
  )
};
```

4. The component is almost complete, we just need to export this function, so that it can be used by other files, App.js in our case

```
import React from "react"

function Header(){
  return (
    <header>
      This is the header
    </header>
  )
};

export default Header;
```

5. Now we can test our component in App.js. Open App.js in VSCode and where all the imports are, include a new line:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';
import Header from './components/header';

function App() {
  return (
```

Note, since Header was exported by default, do not use de-structuring, so no { }

6. With the **`Header`** component imported, we can now use it in our app, so add this line to the top of the pair of **`div`** tags, just under the original **`<header>`** tag:

```
  return (
    <div className="App">
      <header className="App-header">
        <Header />
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
```

If you app is not running, remember npm start will get it up on the browser. We will clean up this default view soon. Open the terminal window in VS Code and run your app from there, it is a lot easier.

7. If you are using the Vite process, you may not see the words of the Header component as it will be hidden by the header bar:

```
import './styles.css'
import Header from './components/header';
function App() {
  return (
    <>
      <Header />
    </>
  )
```

Note, I have removed all the code from the return() block of code, leaving just the Header component, see below.

8. Now that we have a good understanding of how components work, remove all the code from between the **`div`** tags in App.js and just leave our custom header. Also remove the CSS classes as well as anything ending in .test.js:

```
function App() {
  return (
    <div>
      <Header />
    </div>
  );
}
```

Note: you could remove the entire index.css file, but add a comment where that file is being imported in the index.js file.

PART 03 – DEVELOPING THE HEADER COMPONENT

We can further customize our component to look like the original web page that I had built for the other bootcamp.

1. In the case of our original app, we had an image and an **`h1`** tag for our header. Find the original html files and copy just those two tags and paste them into the custom Header function we just created, so header.js.

```
function Header(){
  return (
    <header>
      <img src="images/chart.gif" id="logo" />
      <h1><a href="index.html">Skillsoft Weight Tracker</a></h1>
    </header>
  )
}
```

2. When the browser refreshes, it does not look nice, so lets add some css. Copy the styles.css file provided and paste it into the **`src`** folder. You may have to do this using the File System of the OS that you are on.

3. In App.js, instead of importing App.css, import styles.css

```
import React from 'react';
import logo from './logo.svg';
import './styles.css';
import Header from './components/header';
```

4. Back to the header, copy and paste the image, so *chart.gif*. Then, import chart.gif into header.js.

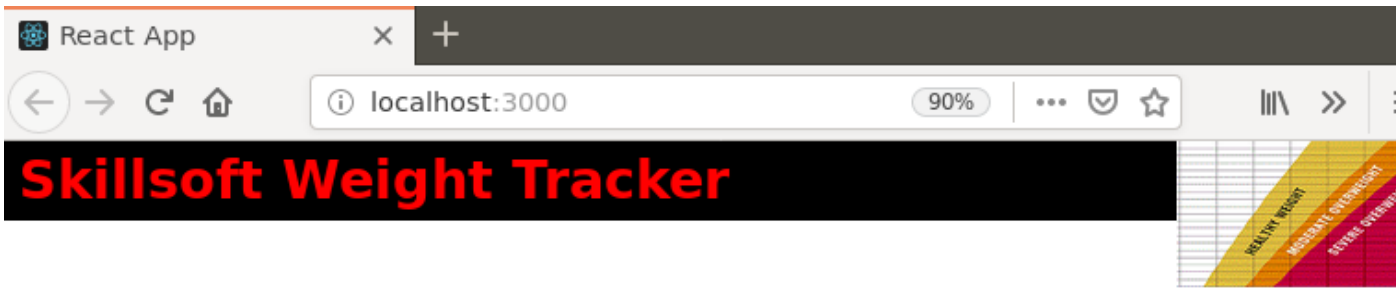
```
import React from "react";
import logo from "./../assets/chart.gif";

function Header(){
```

You can create a new folder called **assets** in the src folder.

5. Now we can use logo as our **`src`** for our img tag, around line 7 of header.js

```
function Header(){
  return (
    <header>
      <img src={logo} id="logo" />
      <h1><a href="index.html">Skillsoft Weight Tracker</a></h1>
    </header>
```

6. At this point, our default or home page should look like the image below:

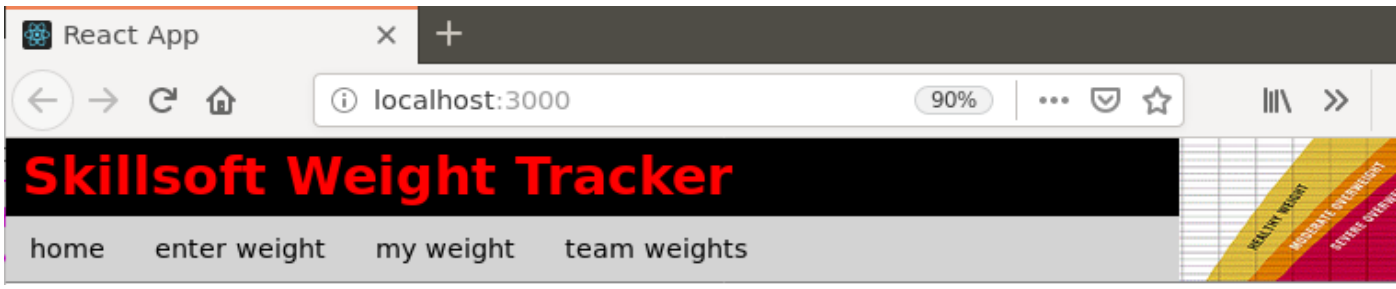


7. We could make a decision to include the navigation bar as part of the header or leave it as a separate component. I think it may work as part of the header so let's include it. If using Vite, remove the < tag and content.

8. Paste the following code into the **`header`** component.

```
        return (
          <header>
            <img src={logo} id="logo" />
            <h1><a href="index.html">Skillsoft Weight Tracker</a></h1>
            <nav>
              <ul>
                <li><a href="index.html">home</a></li>
                <li><a href="enterweight.html">enter weight</a></li>
                <li><a href="myweights.html">my weight</a></li>
                <li><a href="teamweights.html">team weights</a></li>
              </ul>
            </nav>
          </header>
        )
```

9. The page should now look like the image below:

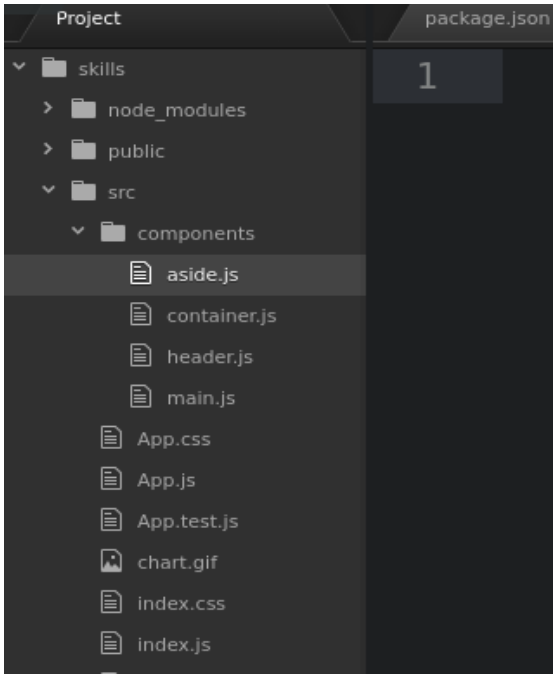


PART 04 – THE CONTAINER COMPONENT

We would need to complete the app, with the rest of our components. Again we could make decisions about which parts of the original HTML website should go into which components. In this case here, we could stay as close to the original as possible. Add this module to the imports section:

1. Create a new component by right-clicking in the Components folder and choosing new file, call it container.js

2. Repeat those same steps to create a component called **`main`** and one called **`aside`**. So create the corresponding .js files.



3. We will do component in component, so first complete the **`main`**. As the components are similar, you can just copy the content from header.js into main.js. Then copy the necessary HTML elements from the provided HTML file. Rename accordingly. You could setup a template also.

```
import React from "react";

function Main(){
  return (
    <main>
      <h2>How to Participate in the Program</h2>
      <p> …
    </main>
  )
}

export default Main
```

Note that this is the same structure as the **`Header`** component file.

4. Do the same for the **`Aside`** component, this is just the shell.

```
import React from "react";

function Aside(){
  return (
  )
}

export default Aside
```

5. Complete the **`return()`** method with the HTML from our original web site

```
return (
    <aside>
      <section>
        <h4>Health News</h4>
        <p>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
        </p>
      </section>
      <section>
        <h4>Healthy Recipes</h4>
        <a href="">grilled chicken</a>
        <a href="">minced beef patties</a>
        <a href="">potato pancakes</a>
        <a href="">fish stew</a>
      </section>
    </aside>
  )
```

6. We will import both the **`Main`** and **`Aside`** components into the **`Container`** component, then place the **`Container`** component into the App.js file

7. First in container.js, import both the **`Main`** and **`Aside`** components:

```
import React from "react";
//
function Container(){
  return (

  )
}
//
export default Container
```

This is the shell

8. Import the other two components that go inside of **`Container`**, so **`main`** and aside:

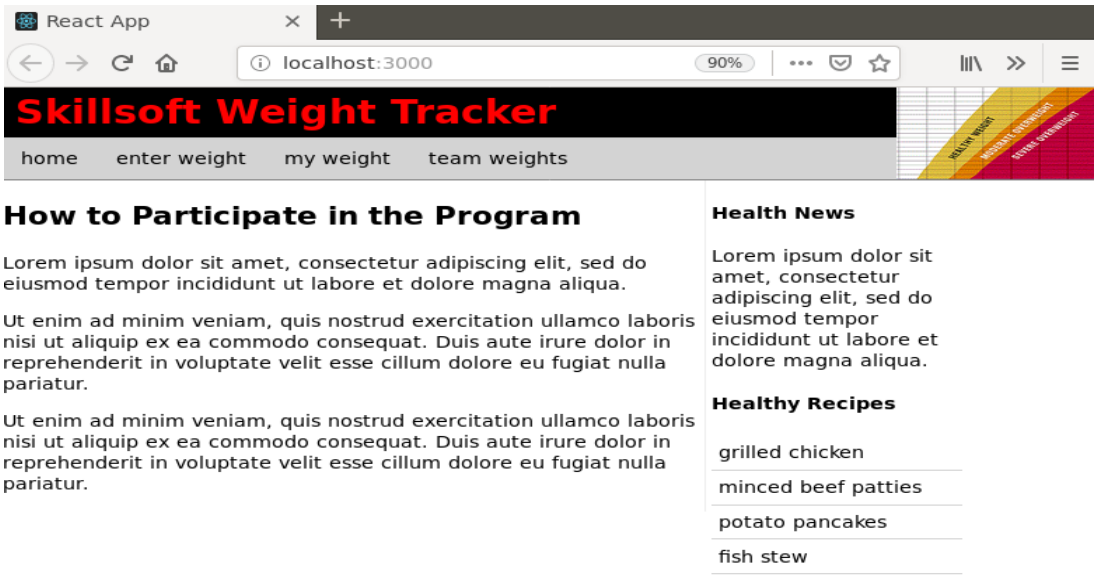
```
import React from "react";
import Main from "./main";
import Aside from "./aside";
//
function Container(){
  return (
    <div>

    </div>
  )
}
```

9. Now we can complete the **`Container`** component, remember to wrap both Components into one pair of **`<div>`** tags

```
function Container(){
  return (
    <div>
      <Main />
      <Aside />
    </div>
  )
}
```

10. Finally add the **`Container`** component to the App.js file. Remember to import container first.

```
function App() {
  return (
    <div>
      <Header />
      <Container />
    </div>
  );
}
```



Note, if you get a series of errors in the terminal window running the React app, just add a place holder into the href="" links, so
`<a href="home">grilled chicken</a>`

PART 05 – THE FOOTER

We will create the *Footer* component in the same manner as the other components. The Footer component will go directly into the app.js file for now.

1. Just copy any of the previous components and rename accordingly. For example if we copy/duplicate the **`Container`** component, we just rename the file to footer.js

2. Once inside footer.js rename **`Container`** to **`Footer`**, and export Footer. Also remove any extra imports like **`Main`** and **`Aside`**, here is the shell:

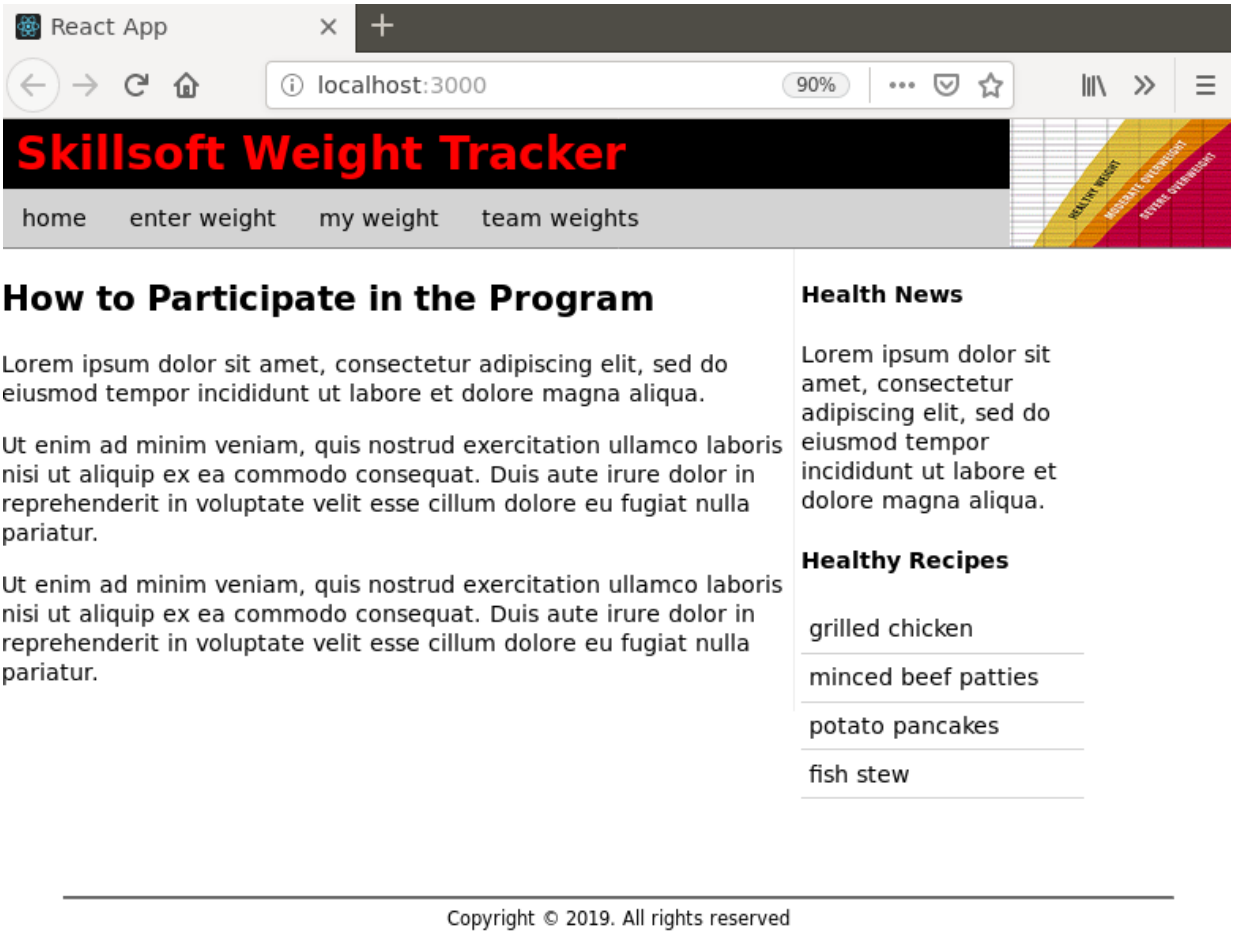
```
import React from "react";

function Footer(){
  return (
    <footer>
    </footer>
  )
}

export default Footer
```

3. Add the footer content by copying from the *html* file provided.

```
    <footer>
      <hr />
      Copyright &copy; 2025. All rights reserved
    </footer>
```

4. In App.js, first import the **`Footer`** component, then include it as part of the pair of **`<div>`** tags

```
import Container from './components/container';
import Footer from './components/footer';

function App() {
  return (
    <div>
      <Header />
      <Container />
      <Footer />
    </div>
  );
}
```



PART 06 – IMPLEMENTING ROUTER

`react-router-dom` provides browser specific components for routing in web apps

1. Install a package to your skills folder to handle routing, its called **`react-router-dom`** so do this: **`>npm install react-router-dom`**
   Of course make sure you stop the app using CTRL-C or use a different terminal window/tab.

2. Once installed go to App.js and import these 2 classes:

```
import Container from './components/container';
import Footer from './components/footer';
import {BrowserRouter, Route, Routes} from 'react-router-dom';


function App() {
```

Note, this is the first time that the package.json file will change.

3. Wrap the app's main content with <Router>, and define routes using <Routes> and <Route>

```
function App() {
  return (
    <Router>
      <Header />
      <Container />
      <Footer />
    </Router>
  );
};
```

4. Next we add the Routes pair of tags:

```
function App() {
  return (
    <BrowserRouter>
      <Routes>
      </Routes>
    </BrowserRouter>
  );
```

5. Next we add the component(s), we have three imported at the moment:

```
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<Header />} />
        <Route path='/container' element={<Container />} />
      </Routes>
```

Try navigating to the root route first then to the /container route.

PART 07 – CONSTRUCTING THE ROOT/HOME PATH

With **`router`** installed, we can now 'construct' what the user will see when they hit our site. It's the same process as before, simply put components together. We will try to match each of the menu items on our site to just one component, so **`home`**, **`enter weight`**, **`my weight`** etc.

In the components folder, copy any of the other components (eg container.js), call it home.js then rename the parts accordingly:

```
function Home(){
  return (
    <home>
    </home>
  )
}
//
export default Home
```

1. Just import the Header, Container and Footer components and add it between the <home> tags:

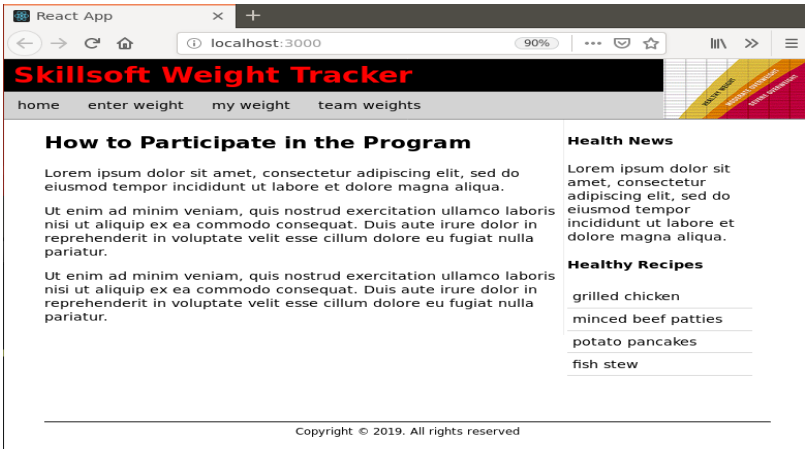
```
import Container from "./container";
import Header from "./header";
import Footer from "./footer";
//
function Home(){
  return (
    <home>
      <Header />
      <Container />
      <Footer />
    </home>
```

2. So now in App.js, import just the **`Home`** component instead of **`Header`**, **`Container`** and **`Footer`**

```
import {BrowserRouter, Route, Routes} from 'react-router-dom';
import './styles.css';
import Home from './components/home';
//
function App() {
```

3. Now we can create a path for /home to serve just the **`Home`** component via the Routes pair of tags:

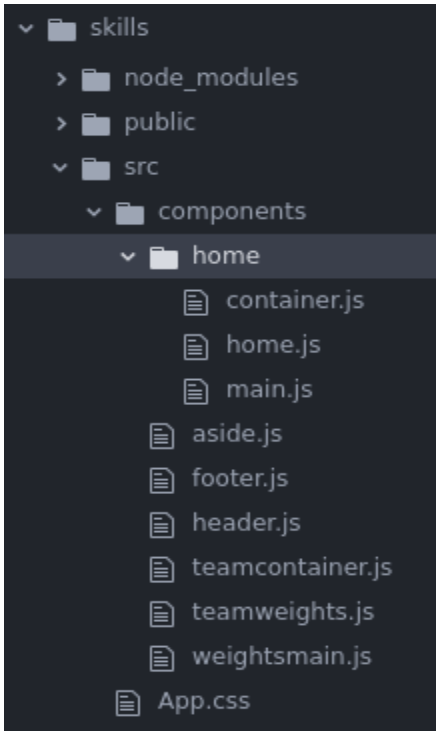
```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<Home />} />
      </Routes>
    </BrowserRouter>
```



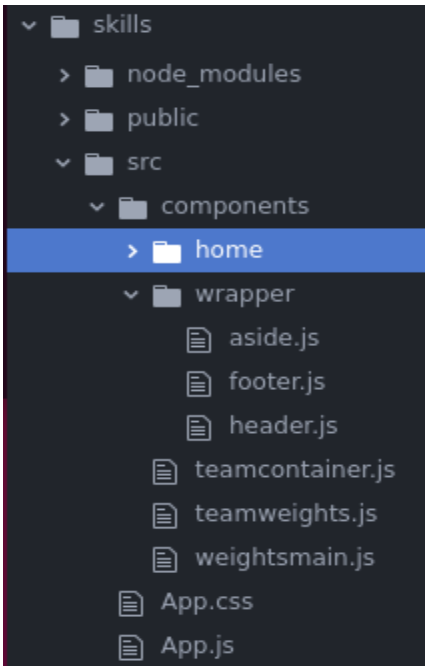
PART 08 – COMPONENT ORGANIZATION

1. If you look at the menu on the original web site, you will see *home*, *enter weight*, *my weight* and *team weights*. We will now arrange our folder structure inside of components to reflect this menu. We will now create two folders to help organize our files. All static files will go into a *wrapper* folder. Each menu item will then have its own folder, starting with the *home* folder.

2. Starting with home, create a folder inside of components called <u>home</u> and put all the .js files that you think belong to the home 'view'.



3. Put all the wrapper files like **`aside footer`** and **`header`** into a folder called **`wrapper`**



4. Of course all the links will break, so lets fix those one by one. Work on the **`home`** folder first then replicate this success to the other folders. Take a look at <u>home.js</u> inside of the <u>home</u> folder, change the paths to reflect our changes.

```
import React from "react";
import Container from "./container";
import Footer from "./../wrapper/footer";
import Header from "./../wrapper/header";
//
function Home(){
```

5. In the container.js file inside of the home folder, change the aside's **path**

```
import React from "react";
import Main from "./main";
import Aside from "./../wrapper/aside";

function Container(){
```

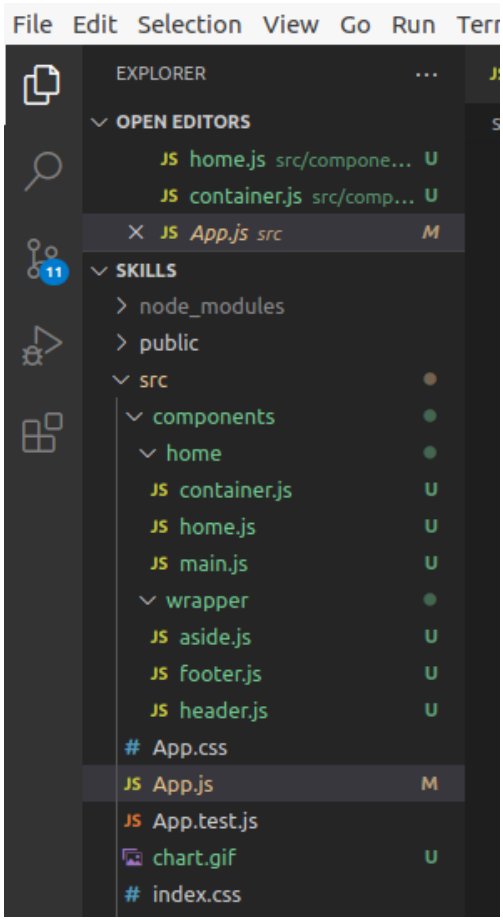
6. Now of course our App.js has to change to reflect the new location of **home**, so in App.js change line 3 or wherever **home** is being imported.

```
import './styles.css';
import Home from './components/home/home';
import { BrowserRouter, Route } from 'react-router-dom';

function App() {
```

7. In header.js fix our logo path, header.js should now be inside of the wrapper folder

```
import React from "react";
import logo from './../../chart.gif';
//
function Header(){
```

This is the view from VSCode. So far we have the components folder and the two child folders home and wrapper.



It would be a good idea here to stop and restart the server so that it builds again.

Appendix A – Pre-requisite Courses

1. Asynchronous Programming in JavaScript - Ron Sumida
   This course references asynchronous code in JS

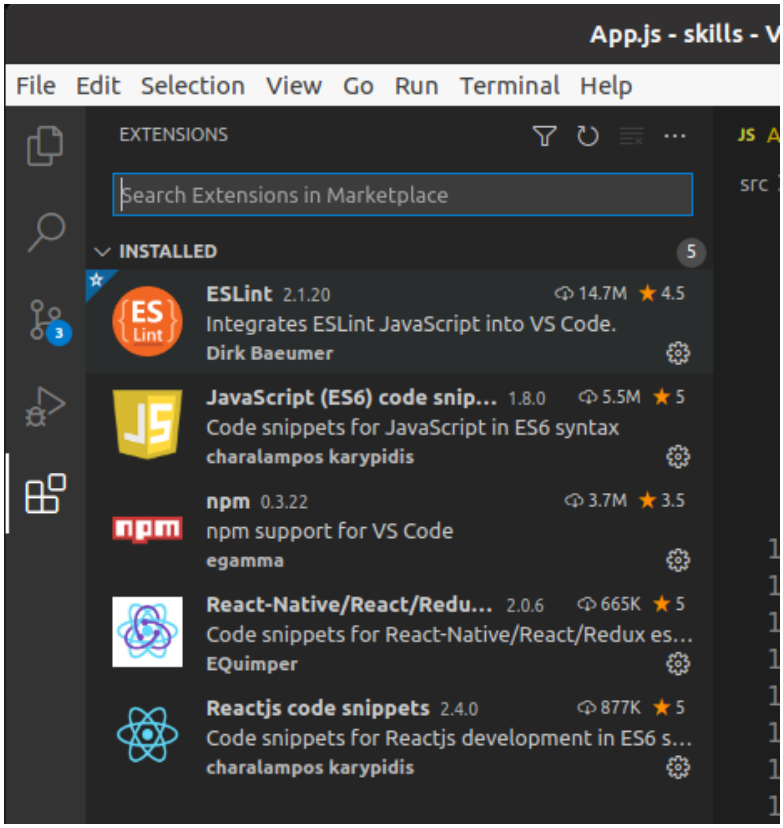
2. JavaScript: Objects - Kishan Iyer
   This course references the ***this*** keyword in JS

3. JavaScript Front-end Development: Functions & Objects - Axle Barr
   References arrow functions (lambda expressions)

Appendix B – ES6 Code Snippets for VS Code

1. Ctrl + Shift + P

2. Install extensions -> search for the following and install them via VS Code



APPENDIX C – USING TRADITIONAL FUNCTION()

1. In the following code, this is the **`componentDidMount()`** method of the main.js file in the employees folder. If you use this method, you will need to **`bind()`** the functions to the **`this`** keyword, otherwise the **`this`** keyword will refer to some other object, like the Window object.

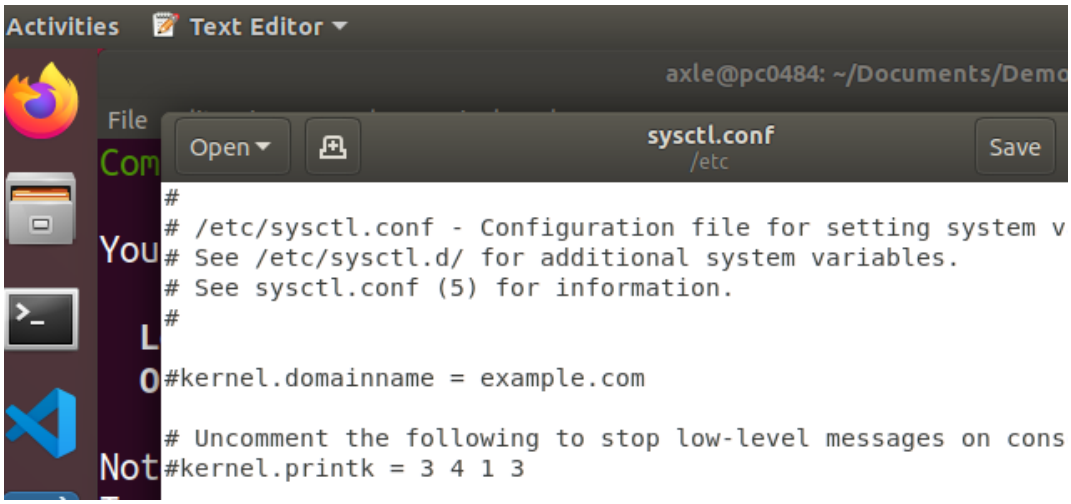
```
componentDidMount(){
  fetch("http://localhost:3030/employees")
  .then(function(data) {
    return(data.json());
  }).then(function(resolvedData){
    this.setState({
      allEmployees:resolvedData
    });
  }.bind(this));
}
```

APPENDIX D – FILE WATCHERS LIMIT REACHED (ENOSPC)

1. If you are on Linux you may get an error about file watchers limit, this is due to the number of times that React recompiles and hotloads. Hidden from view is a package called inotify that Linux systems use to track this sort of activity. You would need to adjust this value using the steps below:

Run the following command at any terminal prompt: **`sudo gedit /etc/sysctl.conf`**

This would bring up the sysctl.conf file for editing



Add a new line with this value: **`fs.inotify.max_user_watches=524288`**
Make sure that there is no # sign in front of the line.
Save the file and try restarting the app with **`npm start`**

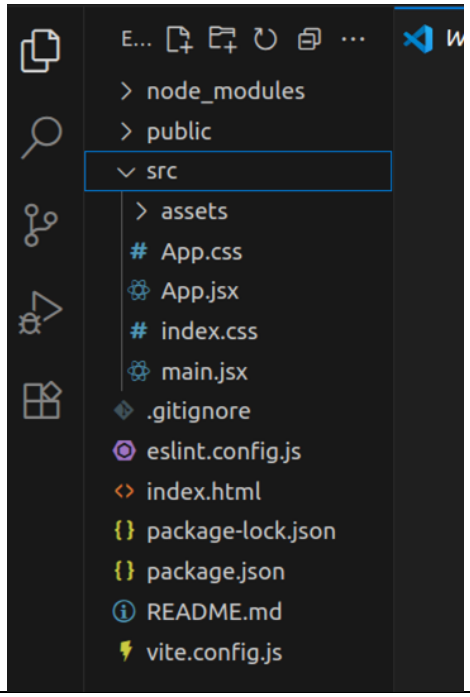
APPENDIX E – INSTALL REACT WITH VITE

1. npm create vite@latest react-bc -- --template react
   you may be asked to install packages, just type "y"

2. change directory into the new folder just created:
   cd react-bc

3. install the necessary dependencies
   npm install

4. You may now open the react-bc folder as a project in VS Code



Your application should look very close to this image.