

React Day 2

Table of Contents

PART 01 – NEW APP SKILLS	2
PART 02 – DEVELOPING THE HEADER COMPONENT	3
PART 03 – THE CONTAINER COMPONENT	5
PART 04 – THE FOOTER	8
PART 05 – IMPLEMENTING ROUTER	10
PART 06 – CONSTRUCTING THE ROOT/HOME PATH	11
PART 07 – COMPONENT ORGANIZATION	13
PART 08 – JSON-SERVER	15
PART 09 – DATABASE DUMP	17
PART 10 – GET REQUEST	18
PART 11 – SHOWING THE RESULTS	21
APPENDIX A – USING TRADITIONAL FUNCTION()	23

PART 01 – NEW APP SKILLS

1. Choose a name for your app and run the command to create a new app, in my case I will be running the command `create-react-app skills`. Run this command in your parent folder.
So *skills*, is the name of my React application.
2. Using VSCode or the file system, create a folder called `components` inside the `src` folder
3. 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:

```
function Header() {  
  }  
}
```

4. Finish the function by returning the component

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

5. 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

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

6. 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 logo from './logo.svg';  
import './App.css';  
import Header from './components/header';  
  
function App() {  
  return (  
    <div>  
      <img alt="Logo" data-bbox="152 809 449 824" />  
      <h1>Skills</h1>  
      <Header />  
    </div>  
  )  
}
```

7. Remove all the code from between the `div` tags in `App.js` and just leave our custom header. Also remove the CSS classes.

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

PART 02 – 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>  
        
      <h1><a href="index.html">Skillsoft Weight Tracker</a></h1>  
    </header>  
  )  
}
```

2. When the browser refreshes, it does not look nice, so let's 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 logo from './logo.svg';  
import './styles.css';  
import Header from './components/header';
```

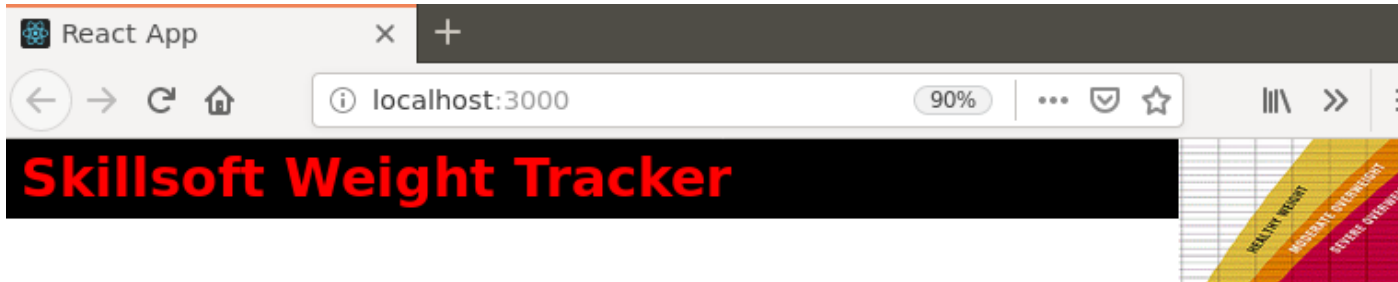
4. Repeat the steps from step 3, but this time copy and paste the image, so `chart.gif`. Then, import `chart.gif` into `header.js`.

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

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:

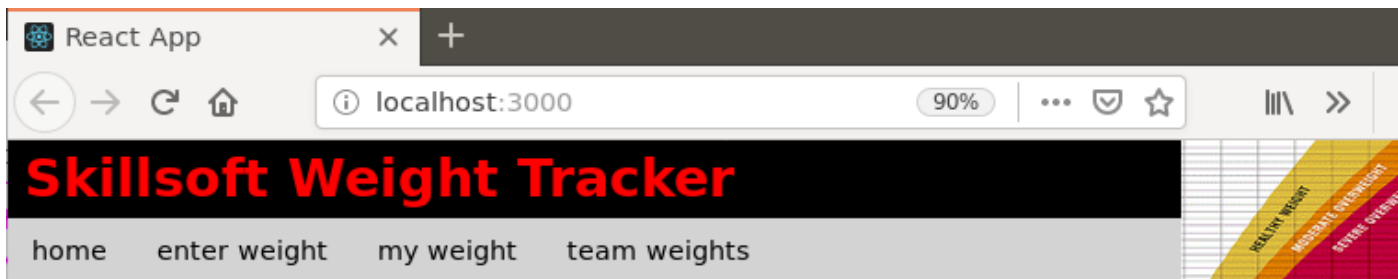


7. We could make a decision to include the navigation bar as part of the header or leave it as a separate component. I think it may work as part of the header so let's include it.

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>  
)
```

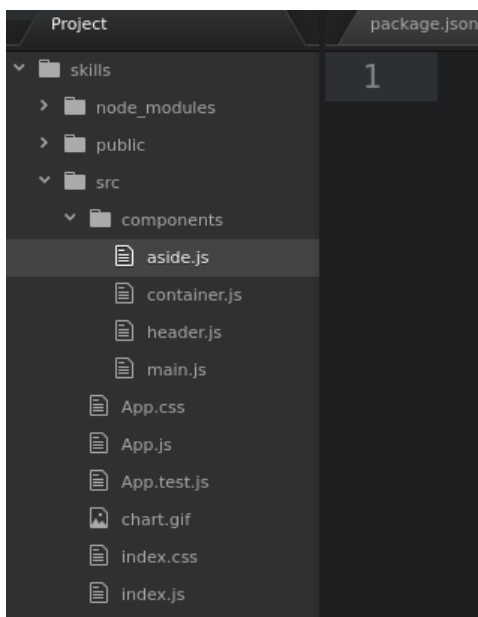
9. The page should now look like the image below:



PART 03 – 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.

```
function Main() {  
  return (  
    <main>  
      <h2>How to Participate in the Program</h2>  
      <p> ...  
    </main>  
  )  
}  
  
export default Main
```

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, create a shell to 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>

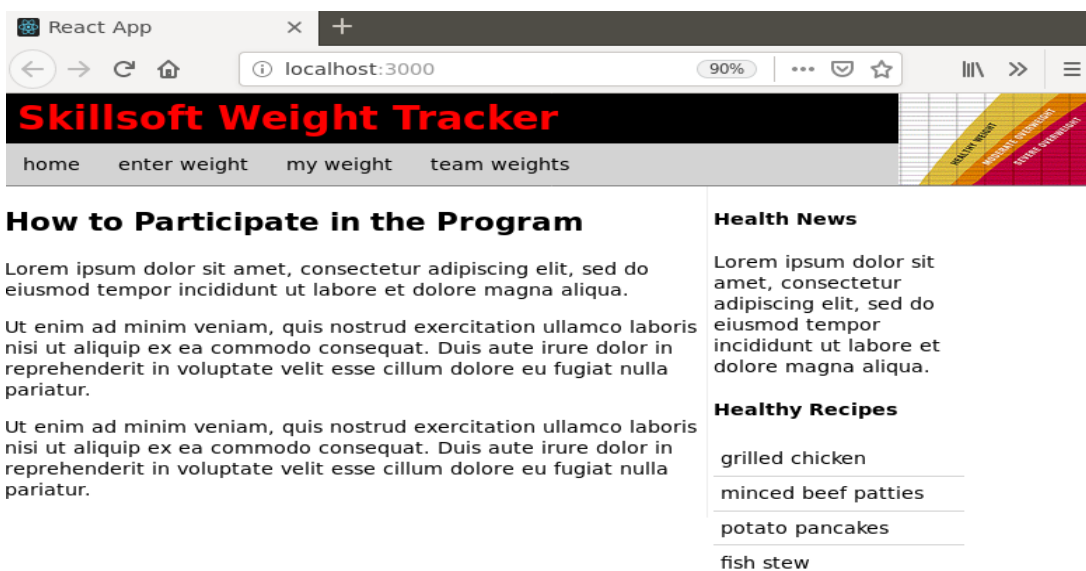
    </div>
  )
}
```

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
`grilled chicken`

PART 04 – 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

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
```

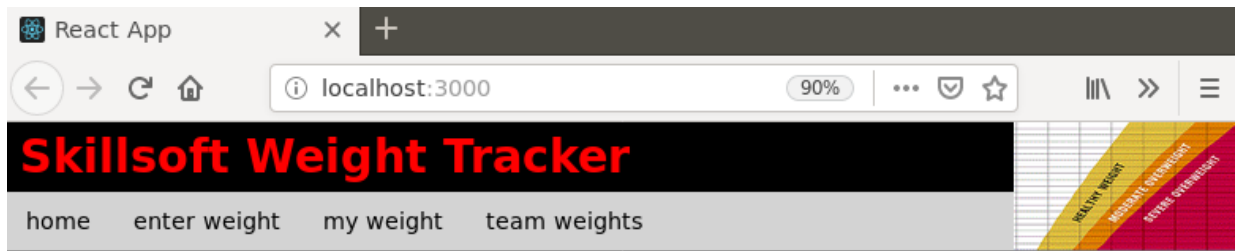
3. Add the footer content by copying from one the *html* file provided.

```
<footer>
  <hr />
  Copyright &copy; 2022. 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>
  );
}
```

How to Participate in the Program

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Health News

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Healthy Recipes

grilled chicken

minced beef patties

potato pancakes

fish stew

PART 05 – 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@5.3.0`

Of course make sure you stop the app using CTRL-C or use a different terminal window/tab.

Notice that this is NOT the latest version of the package, but due to a conflict we need this particular version for this bootcamp.

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} from 'react-router-dom';

function App() {
```

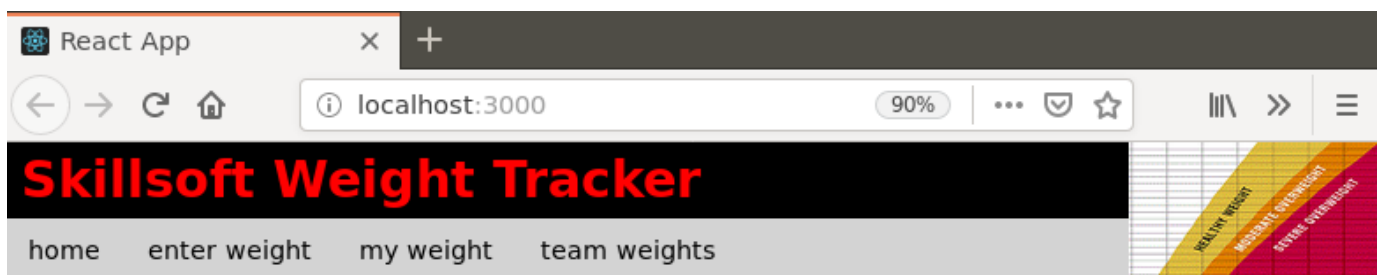
3. Remove all the components from between the `<div>` tags and replace it with one component, the `BrowserRouter`

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

4. Next we add the `Routes` component and within that we add our routes in this manner.

```
return (
  <div>
    <BrowserRouter>
      <Route path="/" component={Header}/>
    </BrowserRouter>
  </div>
);
```

In this case, the path is the *root* and the component, now element, being served there is the `Header` component/element. We will construct an entire page or “view” shortly.



PART 06 – 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 jut 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:

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

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

export default Home
```

1. Just import all the other components necessary to create the home "view", also remove the ones we don't need

```
import React from "react";

function Home(){
  return (
```

Remember that `Aside` and `Main` are contained inside of `Container`, so we only need `Container`.

Remember to change the return part of `Home`

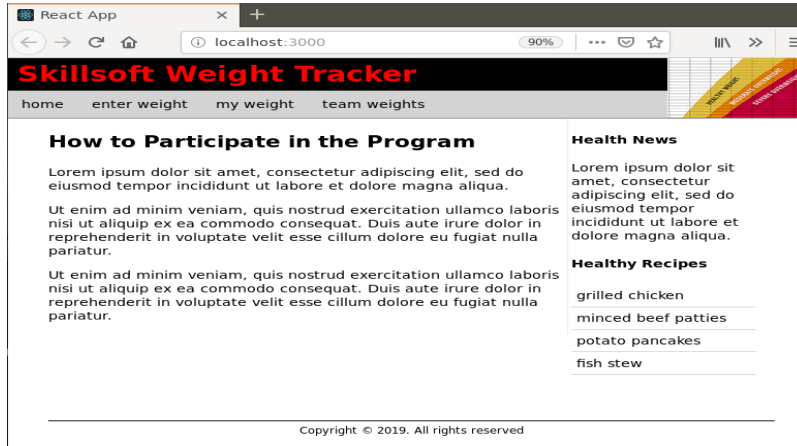
```
function Home(){
  return(
    <div>
      <Header />
      <Container />
      <Footer />
    </div>
  )
}
```

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

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

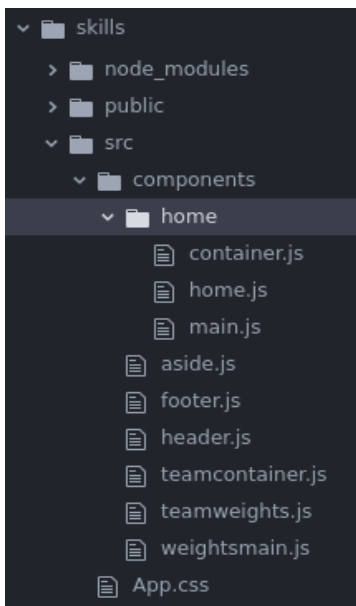
3. Now we can serve just the `Home` component via the path attribute

```
return (  
  <div>  
    <BrowserRouter>  
      <Route path="/home" component={Home}/>  
    </BrowserRouter>  
  </div>  
)
```



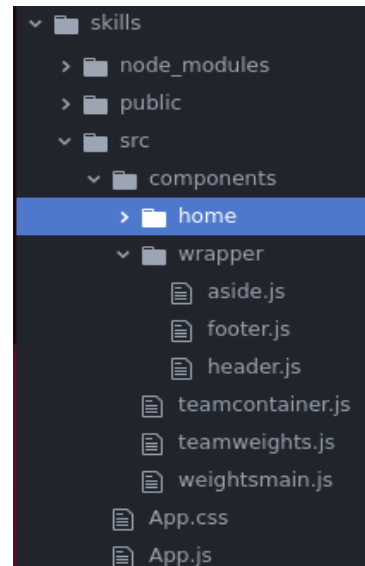
PART 07 – COMPONENT ORGANIZATION

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 home 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 let's fix those one by one. Work on the **home** folder first then replicate this success to the other folders. Take a look at home.js inside of the home 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() {
```

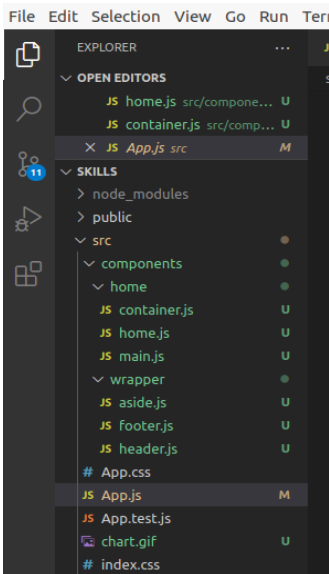
6. Now of course our App.js has to change to reflect the new location of home, so in App.js change line 3 or wherever home is being imported.

```
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 aside.

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

PART 08 – JSON-SERVER

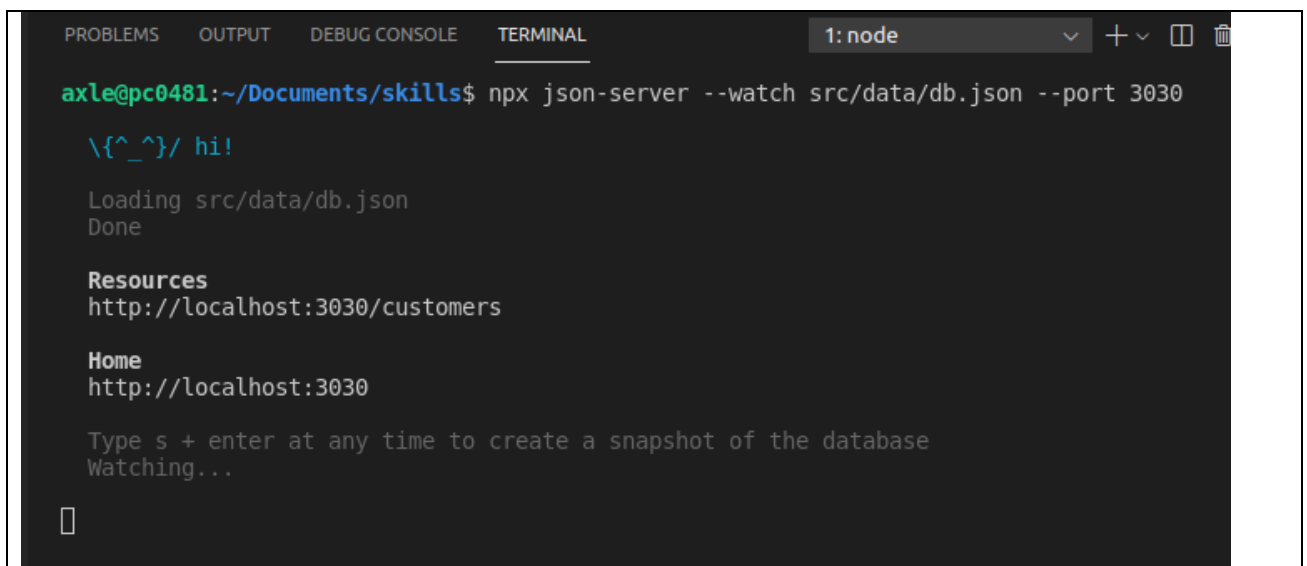
We will now introduce a fake back-end server so that we can simulate a real API call using **http**.

1. Install the JSON server from any terminal using this command: `npm install -g json-server`
2. Create a new folder in the current src folder called data. Then inside of that create a db.json file using the following code:

```
{
  "customers":[
    {
      "username":"Axle",
      "password":"1234"
    }
  ]
}
```

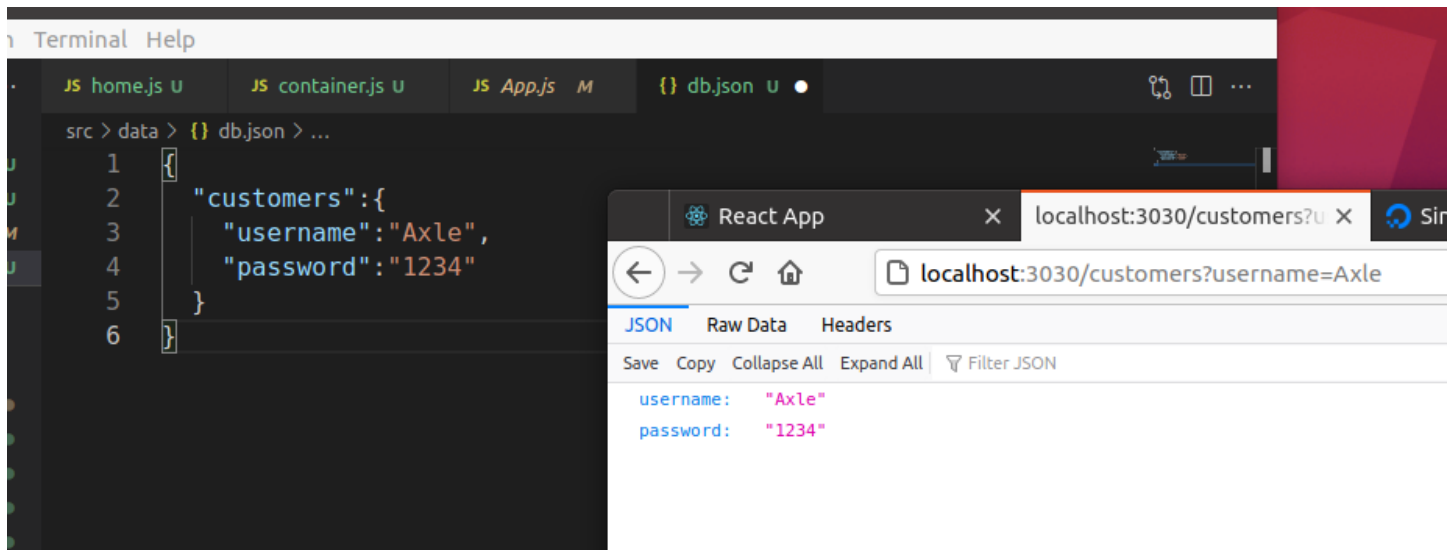
Note: *customers* represent an array of *customer* objects

3. Use a terminal window and point to the **same folder** where your application lives, and type in `npx json-server --watch src/data/db.json --port 3030` (be careful with this command, there are **two dashes** before watch and port)
4. If all goes well, you should see something like the image below:

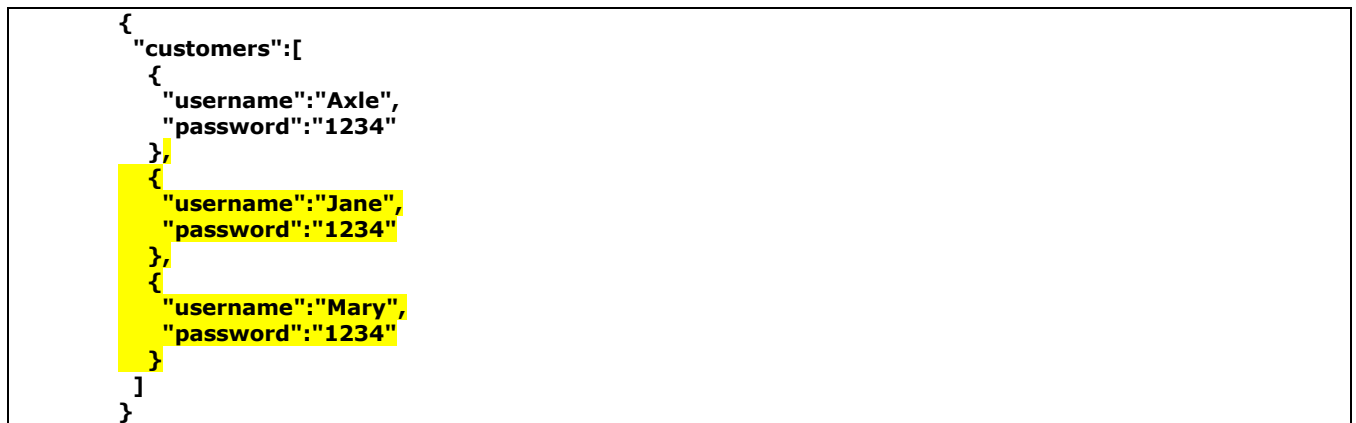
A screenshot of a VS Code terminal window. The terminal title bar shows '1: node'. The command prompt is 'axle@pc0481:~/Documents/skills\$'. The command entered is 'npx json-server --watch src/data/db.json --port 3030'. The output shows a green prompt character, a blue prompt character, and the text 'hi!'. Below that, it says 'Loading src/data/db.json' and 'Done'. Then it shows 'Resources' and 'http://localhost:3030/customers'. Below that, it shows 'Home' and 'http://localhost:3030'. At the bottom, it says 'Type s + enter at any time to create a snapshot of the database' and 'Watching...'. There is a small square icon at the bottom left of the terminal window.

Notice I used the terminal accessible from VS Code and this window is now running a process, so we cant close it. Also I am using port 3030 since ReactJS runs on 3000 which is the default port of the Json-server

5. Open a new browser tab and navigate to <http://localhost:3030/customers>, you should see the database data there



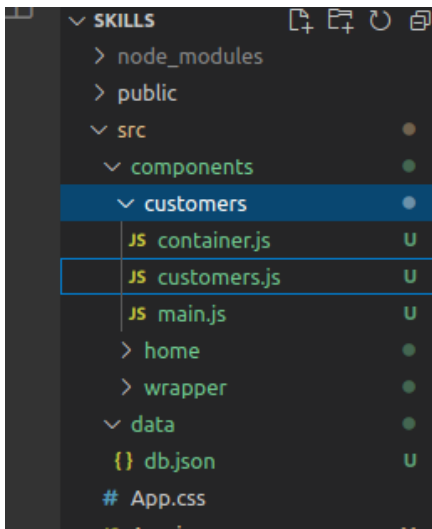
6. To make the app more interesting, add two more customers using the same structure as the one for Axle



PART 09 – DATABASE DUMP

in this part, we will create a new component to simply read all the data in our database by making an API call into the db.json file.

1. Duplicate (or copy/paste/rename) the original home folder to just customers
2. Rename the home.js file inside of the new customers folder to customers.js



3. In customers.js, change the name of the function to customers and also the last line

```
import React from "react";
import Container from "../container";
import Footer from "../wrapper/footer";
import Header from "../wrapper/header";

function Customers(){
  return(
    <div>
      <Header />
      <Container />
      <Footer />
    </div>
  )
}

export default Customers
```

4. In main.js change the content being displayed to something like this:

```
function Main(){
  return(
    <main>
      <h2>Customers List</h2>
    </main>
  );
}>
```

5. Adding the new component to the route is easy, go to App.js and import the customers component.

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

function App() {
```

6. Then simply add a new route like this:

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/customers" element={<Customers />} />
  </Routes>
```

7. Now if you navigate to **localhost:3000/customers** you should see the new component, but something looks not right. The first path should have the **exact** keyword added to it:

```
function App() {
  return (
    <div>
      <BrowserRouter>
        <Route exact path="/" component={ Home } />
        <Route path="/customers" component={ Customers } />
      </BrowserRouter>
    </div>
```

PART 10 – GET REQUEST

In this section we will make an API call to the /customers end point of our json server. The result from that call will have to be handled asynchronously. Most of the changes will be in the main.js file which is in the customers folder.

1. In main.js import the **Component** module from React by using de-structuring syntax. Also turn **Main** into a class by *extending Component*

```
import React, { Component } from "react"
//
class Main extends Component{
  return (
    <main>
```

2. Add a render() method, then wrap the entire return statement into the render method:

```
class Main extends Component{
  render(){
    return (
      <main>
        <h2>Customer List</h2>
      </main>
    )
  }
}
```

3. and add a constructor, also export **Main** as before.

```
import React, { Component } from "react"
class Main extends Component{
  constructor(){}
}
export default Main
```

4. The page should refresh and show a reference error, lets fix this by referencing the super constructor and also passing props as a parameter

```
class Weights extends Component{
  constructor(props){
    super(props);
  }
}
```

In a class, calling the constructor and passing props is how the class can access the *this* parameter and also be able to pass properties from parent to child etc. Also at this point, since we activated ES Lint, it will complain about a no-useless-constructor, just follow VS Code quick fix solution.

5. This is the entire main.js file at the moment

```
import React, { Component } from "react"
//
class Main extends Component{
  // eslint-disable-next-line no-useless-constructor
  constructor(props){
    super(props);
  }
  //
  render(){
    return (
      <main>
        <h2>Customer List</h2>
      </main>
    )
  }
}
//
export default Main
```

6. Methods are executed in a specific order with React, so the suggestion is to use the `componentDidMount()` method (of any class component) as a wrapper for accessing external data. Insert the method and also within it, insert the `fetch()` method to access our API (json-server)

```
constructor(props){  
  super(props);  
}  
  
componentDidMount(){  
  fetch()  
}  
render(){
```

7. Before we execute the `fetch()` method, let's prepare a variable to hold the contents of that 'hit'. Within the constructor let us initialize an empty array for this purpose

```
constructor(props){  
  super(props);  
  this.state = {  
    allCustomers: []  
  }  
}  
  
componentDidMount(){
```

We have to use `this.state` and point it to a complex object, containing our variable.

8. Now we can work on the `fetch()` method. First pass into the `fetch()` method a parameter that represents our API's URL

```
componentDidMount(){  
  fetch("http://localhost:3030/customers")  
}
```

9. The `fetch()` method returns a *promise*, so let's chain on a `then()` method and log the response

```
componentDidMount(){  
  fetch("http://localhost:3030/customers")  
    .then(data => {  
      console.log(data.json());  
    });  
}
```

Note: if you want to see this function using the traditional *function* keyword, check the appendix.

10. Since the return is also a promise object we can return that object so that we can handle it with another `then()` method:

```
componentDidMount(){  
  fetch("http://localhost:3030/customers")  
    .then(data => {  
      return data.json();  
    })  
}
```

11. We can log the second return just to see what is inside

```
componentDidMount(){
  fetch("http://localhost:3030/customers")
  .then(data => {
    return data.json();
  })
  .then(resolvedData=>console.log(resolvedData));
}
```

12. Once we have the data we can store it in our class property

```
fetch("http://localhost:3030/customers")
.then(data => {
  return data.json();
})
.then(resolvedData=>{
  this.setState({
    allCustomers:resolvedData
  });
});
```

Note we have to use a function to achieve this, `setState()`

PART 11 – SHOWING THE RESULTS

In this section we will display the data returned by our `fetch()` method. Most of the changes will be in the `render()` method since this is what gets passed to the browser. Remember we now have the `allCustomers` array which contains all of our customer objects

1. You may have inserted `<main>` tags from the previous section, so keep this and insert curly braces so that React can interpret our `allCustomers` array

```
render(){
  return (
    <main>
      <h2>Customer List</h2>
      {}
    </main>
  )
}
```

2. Insert the `allCustomers` array and since it is an array add the `map()` method to the end.

```
render(){
  return(
    <main>
      {
        this.state.allCustomers.map( )
      }
    </main>
  );
}
```

3. The `map()` method gives us access to the array members but we need a variable so that each member can be stored somewhere, here we will use `customer` (singular) and we pass that value into a pair of parenthesis with a pair of `<div>` tags inside

```
<main>
{
  this.state.allCustomers.map(customer =>
    <div>
    </div>
  )
}
```

4. Now we can get access to the value of each of the items inside the array.

```
{
  this.state.allCustomers.map( customer => (
    <div>
      {customer.username}&nbsp;
      {customer.password}
    </div>
  ))
}
```

5. The error showing up in the console window is related to how React works with the `map()` method. This method needs a key and a value when it displays data so we can use the implicit count of `map()` itself to give the code a key and value:

```
render(){
  return(
    <main>
      <h2>Customers List</h2>
      {
        this.state.allCustomers.map((customer, i) =>
          (<div key={i}>
            {customer.username}&nbsp;
            {customer.password}
          </div>)
        )
      }
    </main>
  );
}
```

Skillsoft Weight Tracker

home enter weight my weight team weights

Customers List

Axle 1234
Jane 1234
Mary 1234

6. (Optional) this is an alternative to writing constructing what is inside of the `map()` method:

```
render(){
  return (
    <main>
      <h2>Customer List</h2>
      {
        this.state.allCustomers.map( (customer, i) => {
          return <div key={i}>
            {customer.username}&nbsp;
            {customer.password}
          </div>
        })
      }
    </main>
  )
}
```

APPENDIX A – USING TRADITIONAL FUNCTION()

1. In the following code, this is the `componentDidMount()` method of the `main.js` file in the `customers` 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/customers")
    .then(function(data) {
      return(data.json());
    }).then(function(resolvedData){
      this.setState({
        allCustomers:resolvedData
      });
    }).bind(this);
}
```