Table of Contents

[**part 01 – ReactJS Setup** 2](#_Toc40359583)

[**part 02 – Adding Components** 3](#_Toc40359584)

[**part 03 – Developing the Header Component** 5](#_Toc40359585)

[**part 04 – The Container Component** 7](#_Toc40359586)

[**part 05 – The Footer** 10](#_Toc40359587)

[**Part 06 – Implementing Router** 11](#_Toc40359588)

[**part 07 – Constructing the Root/Home Path** 12](#_Toc40359589)

[**part 08 – Teamweights View** 14](#_Toc40359590)

[**part 09 – Configuring Teamweights** 16](#_Toc40359591)

Day03 UI with ReactJS

Previously when we build the ReactJS app, we worked with a basic and fundamental structure. For today however we introduce a helper application developed by the Facebook group called **Create React App**. With this app, we build a complete ReactJS application with lots of boilerplate code and even testing packages are included. As we already learned, ReactJS works with components that are delivered to some HTML document. What we build therefore are *views* that are streamed via the HTML container. Along the way we introduce routing in ReactJS applications.

For a different Bootcamp I had created a simple website again based on employees entering weights on a daily basis. We will be using the HTML files from this site to build our ReactJS application. You have access to this site via a zipped file on GitHub.

# part 01 – ReactJS Setup

1. From a terminal window, pointing to your parent folder, run the command   
   >**sudo npm install -g create-react-app** (remember to do this globally)
2. 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**  
   So *skills*, is the name of my React application. In the end, you will have to folders, App which contains our NodeJS code and skills which will hold our front end React application.
3. (optional for Linux based systems) If you notice that there is a lock on the app folder, run the **chown** command against the Day03 folder like this:   
   **sudo chown -R $USER ./**
4. CD into the **skills** folder and open the application with Atom (or another editor) and most of our work will be in the **app** folder, which acts like the parent folder.
5. Follow the prompts that appear in the terminal window, so first CD into the skills folder then run the command >**npm start**, your browser should open and you should see the default React UI  
   (to comment out the logo change line 9 to this**:   
   {/\*<img src={logo} className="App-logo" alt="logo" />\*/})**
6. In Atom, 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**.

# part 02 – Adding Components

1. Using Atom or the file system, 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";**  **function Header(){**    **}** |

1. Finish the function by returning the component

|  |
| --- |
| **function Header(){**  **return (**  **<header>**  **This is the header**  **</header>**  **)**  **}** |

1. 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

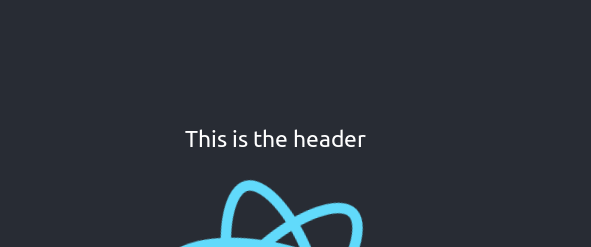
|  |
| --- |
| **import React from "react"**  **function Header(){**  **return (**  **<header>**  **This is the header**  **</header>**  **)**  **}**  **export default Header** |

1. Now we can test our component in App.js. Open App.js in Atom 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 (** |

1. 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.** |

1. You should now be able to see this custom header just above the spinning logo on the default page:  
   
2. Now that we have a good understanding of how components work, remove all the code from between the **div** tags and just leave our custom header. Also remove the CSS classes.

|  |
| --- |
| **function App() {**  **return (**  **<div>**  **<Header />**  **</div>**  **);**  **}** |

# 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>**  **)**  **}** |

1. 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.
2. In App.js, instead of importing App.css, import styles.css

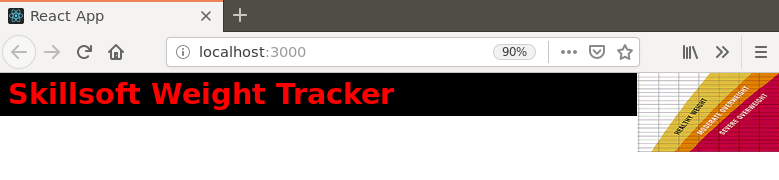
|  |
| --- |
| **import React from 'react';**  **import logo from './logo.svg';**  **import './styles.css';**  **import Header from './components/header';** |

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

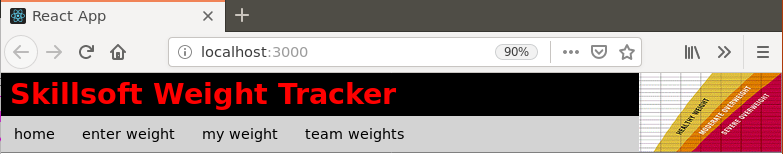
|  |
| --- |
| **import React from "react";**  **import logo from '../chart.gif';**  **function Header(){** |

1. 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>** |

1. At this point, our default or home page should look like the image below:  
     
   
2. 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.
3. Paste the following code into the **header** component.

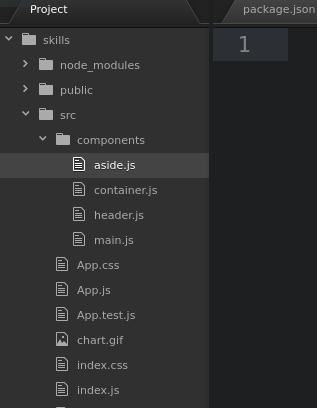
|  |
| --- |
| **return (**  **<header>**  **<img src={logo} id="logo" />**  **<h1><a href="index.html">Skillsoft Weight Tracker</a></h1>**  **<nav>**  **<ul>**  **<li><a href="index.html">home</a></li>**  **<li><a href="enterweight.html">enter weight</a></li>**  **<li><a href="myweights.html">my weight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

1. 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.



1. 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.

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

|  |
| --- |
| **import React from "react";**  **function Aside(){**  **return (**  **)**  **}**  **export default Aside** |

1. 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>**  **)** |

1. We will import both the **Main** and **Aside** components into the **Container** component, then place the **Container** component into the App.js file
2. First in container.js, import both the **Main** and **Aside** components:

|  |
| --- |
|  |

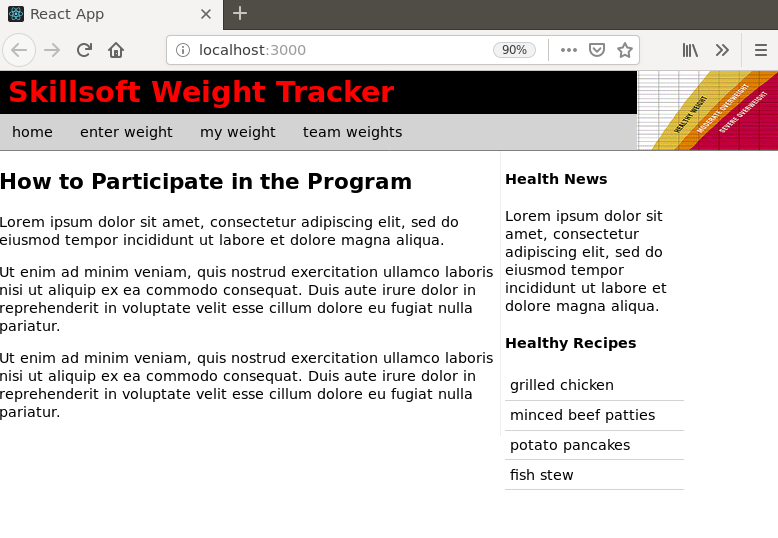
This is the shell

1. 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>**  **)**  **}** |

1. 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

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:

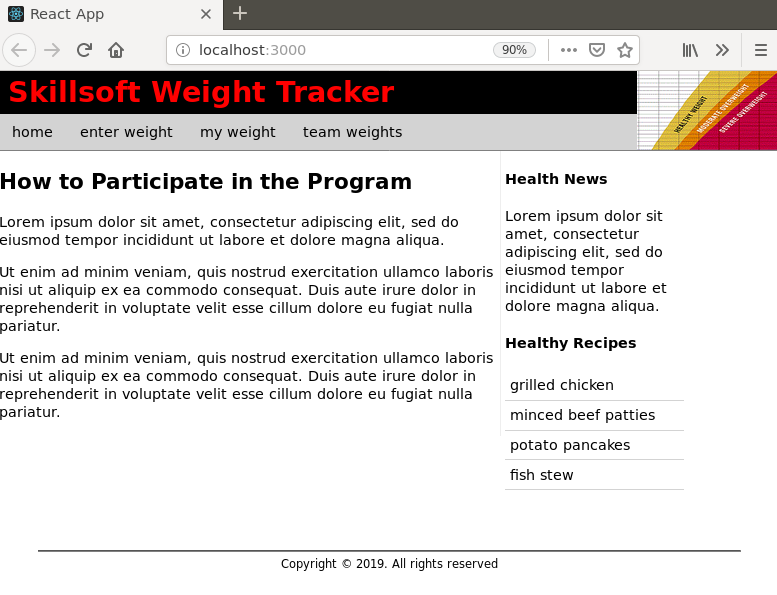
|  |
| --- |
| **import React from "react";**  **function Footer(){**  **return (**  **<footer>**  **</footer>**  **)**  **}**  **export default Footer** |

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

|  |
| --- |
| **<footer>**  **<hr />**  **Copyright &copy; 2019. All rights reserved**  **</footer>** |

1. 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 web apps

1. Install a package to your skills folder to handle routing, its called **react-router-dom** so do this: **>npm install --save react-router-dom**
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() {** |

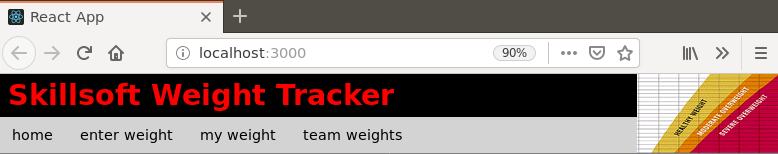
1. Remove all the components from between the **<div>** tags and replace it with one component, the **Router**

|  |
| --- |
| **function App() {**  **return (**  **<div>**  **<BrowserRouter>**  **</BrowserRouter>**  **</div>**  **);**  **}** |

1. For each route, we use the **Route** component and pass the component we want to route to. The component is defined by a path.

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

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



# 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. 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";**  **import Container from "./container";**  **import Footer from "./footer";**  **import Header from "./header";**  **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**

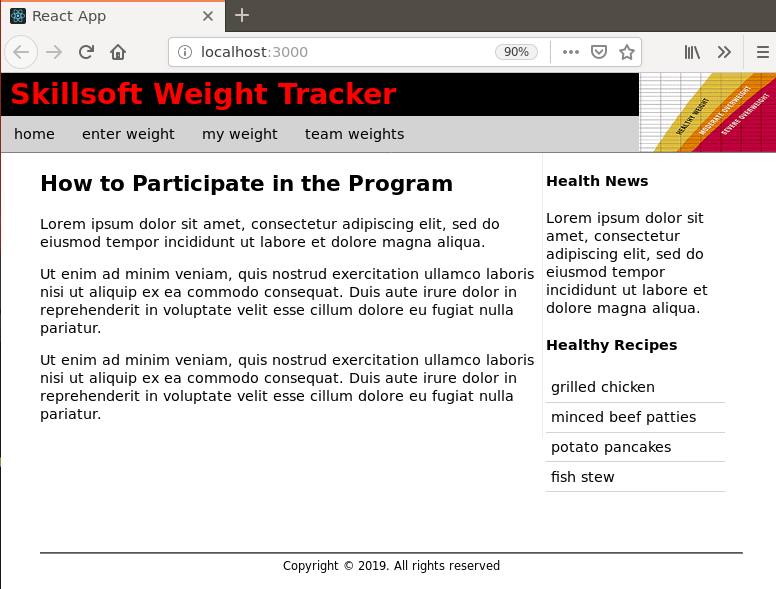
|  |
| --- |
| **function Home(){**  **return(**  **<div>**  **<Header />**  **<Container />**  **<Footer />**  **</div>**  **)**  **}** |

1. 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';** |

1. Now we can serve just the **Home** component via the path attribute

|  |
| --- |
| **return (**  **<div>**  **<BrowserRouter>**  **<Route path="/" component={Home}/>**  **</BrowserRouter>**  **</div>** |



# part 08 – Teamweights View

Now that we have a **Home** component, we can create a **Teamweights** component but this component has to interact with our API. Remember our API supplies data.

1. Duplicate the home.js file and call it teamweights.js. Change the function name and the export default values to **Teamweights**.
2. The **Teamweights** ‘view’ cannot use the original container.js as it contains a different **Main**. So duplicate container.js also but call it twcontainer.js
3. Change the names in twcontainer.js accordingly

|  |
| --- |
| **import React from "react";**  **import Main from "./main";**  **import Aside from "./aside";**  **//**  **function TwContainer(){**  **return (**  **<div>**  **<Main />**  **<Aside />**  **</div>**  **)**  **}**  **//**  **export default TwContainer** |

Notice everything else is the same as in the original **Container** component

1. Back in teamweights.js, change line 10 from **Container** to **TwContainer**. Remember to import it at the top.

|  |
| --- |
| **import React from "react";**  **import TwContainer from "./twcontainer";**  **import Footer from "./footer";**  **import Header from "./header";**  **//**  **function Teamweights(){**  **return (**  **<div>**  **<Header />**  **<TwContainer />**  **<Footer />**  **</div>**  **)**  **}**  **//**  **export default Teamweights** |

1. In App.js import the **TwContainer** and create a path to it

Here is the entire App.js file:

|  |
| --- |
| **import React from 'react';**  **import './styles.css';**  **import Home from './components/home';**  **import Teamweights from './components/teamweights';**  **import {BrowserRouter, Route} from 'react-router-dom';**  **function App() {**  **return (**  **<div>**  **<BrowserRouter>**  **<Route path="/" component={Home}/>**  **<Route path="/teamweights" component={Teamweights}/>**  **</BrowserRouter>**  **</div>**  **);**  **}**  **export default App;** |

1. Now if you go to <http://localhost:3000/teamweights> you should see exactly what you see at <http://localhost:3000/>
2. When the page is displayed, it may display 2 home pages, we can solve that problem by inserting the *exact* keyword just after the Route on line 11.

|  |
| --- |
| **<div>**  **<BrowserRouter>**  **<Route exact path="/" component={Home}/>**  **<Route path="/teamweights" component={Teamweights}/>**  **</BrowserRouter>**  **</div>** |

# part 09 – Configuring Teamweights

Right now **teamweights** and **home** both show the same content, but we can change **teamweights** by changing what appears in **Main.** For this part, you must have your Node server running. Remember to start your Node app, you would run **npm run dev** in a terminal window pointing to your **App** folder.

1. Duplicate the original main.js file to just teamweightsmain.js
2. Change the weightsmain.js file accordingly

|  |
| --- |
| **import React from "react"**  **function TwMain (){**  **return (**  **<main>**  **<h2>Teamweights</h2>**  **</main>**  **)**  **}**  **export default TwMain** |

The **<main>** tags can remain for now

1. In twcontainer.js, change lines 2 and 8 to reflect T**eamweights** instead of **main**.

|  |
| --- |
| **import React from "react";**  **import TwMain from "./twmain";**  **import Aside from "./aside";**  **//**  **function TwContainer(){**  **return (**  **<div>**  **<TwMain />**  **<Aside />**  **</div>**  **)**  **}**  **//**  **export default TwContainer** |

1. We will configure the twmain.js file into a *Class* instead of a *function*.
2. In twmain.js import the **Component** module from React by using de-structuring syntax. Also turn **TwMain** into a class by *extending* Component

|  |
| --- |
| **import React, { Component } from "react"**  **//**  **class TwMain extends Component{**  **return (**  **<main> >** |

1. Remove the **return** statement and add a constructor, also export **TwMain** as before. So this should be all the code inside twmain.js

|  |
| --- |
| **import React, { Component } from "react"**  **class TwMain extends Component{**  **constructor(){}**  **}**  **export default TwMain** |

1. 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);**  **}**  **}** |

1. Now we get a render error message. When using a class, we must use the **render()** method just like *return* for a function. Just add a **render()** method.

|  |
| --- |
| **class Weights extends Component {**  **constructor(props){**  **super(props);**  **}**  **render(){**  **return();**  **}**  **}** |

Note that we still get an error, this is because the render is not returning anything at the moment. To prevent this error just place a pair of **<main>** tags inside of the *return* parameters.

1. 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 (the one we build on day01)

|  |
| --- |
| **constructor(props){**  **super(props);**  **}**  **componentDidMount(){**  **fetch()**  **}**  **render(){** |

1. Before we execute the **fetch()** method, lets 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 = {**  **allWeights: []**  **}**  **}**  **componentDidMount(){** |

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

1. 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:800/getweights")**  **}** |

1. The **fetch()** method returns a *promise*, so lets chain on a **then()** method to pass the returned data through a **json()** method

|  |
| --- |
| **componentDidMount(){**  **fetch("http://localhost:800/getweights")**  **.then(response => response.json())**  **}** |

1. The next **then()** method will take the response from the first **then()** method and store it in the array we prepared in step 9

|  |
| --- |
| **componentDidMount(){**  **fetch("http://localhost:8000/getweights")**  **.then(response => response.json())**  **.then(response => {**  **this.setState({**  **allWeights:response**  **})**  **})**  **}** |

1. The **allWeights** variable is now available to any part of this class, so in the render method we can apply it there. First lets prepare the return statement with a pair of **<main>** tags and curly braces:

|  |
| --- |
| **render(){**  **return(**  **<main>**  **{}**  **</main>**  **);** |

1. We now use **allWeights** and interpolation to present the data to the browser. As the browser refreshes you get a nasty error in red.

|  |
| --- |
| **render(){**  **return (**  **<div>**  **{ this.state.allWeights }**  **</div>**  **)**  **}** |

1. We chain a **map()** method onto the **allWeights** variable since **allWeights** represents an array

|  |
| --- |
| **render(){**  **return(**  **<div>**  **{** **this.state.allWeights.map() }**  **</div>**  **);** |

1. Pass a function to the **map()** method to handle the data within **allWeights** but return each document in a separate pair of **<div>** tags.

|  |
| --- |
| **return(**  **<main>**  **{**  **this.state.allWeights.map(emp =>**  **(<div></div>)**  **)**  **}**  **</main>**  **);** |

1. Then, we can access the data using interpolation, using the space between the pair of **<div>** tags. Use the React app to see results of this code, so **http://localhost:3000/teamweights**

|  |
| --- |
| **render(){**  **return(**  **<main>**  **{**  **this.state.allWeights.map(emp =>**  **(<div>**  **{emp.empName}**  **</div>)**  **)**  **}**  **</main>**  **);**  **}**  **}>** |

The problem now is that we still have the repeating data locked up inside the **employeeWeights** which was defined in **weightSchema**(see Day02). If we try to display emp.employeeWeights like this: {emp.empName} {emp.employeeWeights} we get a nasty error complaining about children object keys etc.

|  |
| --- |
| **this.state.allWeights.map(emp =>**  **(**  **<div>**  **{emp.empName}**  **{emp.employeeWeights}**  **</div>**  **)**  **)** |

1. But the error is able to access the inner field names (**weighedDate** and **empWeight**). We would need to create a inner loop to loop through this array and display **weighedDate** and **empWeight**. We will use the **map()** method a second time.

|  |
| --- |
| **this.state.allWeights.map(emp => (**  **<div>**  **{emp.empName}**  **{emp.employeeWeights.map()}**  **</div>**  **))** |

1. Lets pass to **map()** a function to extract the data stored in **employeeWeights**.

|  |
| --- |
| **this.state.allWeights.map(emp => (**  **<div>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {})}**  **</div>**  **))** |

Note, **weight** will represent the **weightSchema** schema, which has basically 2 fields that we are interested in.

1. From within that anonymous function we can extract the weight:

|  |
| --- |
| **this.state.allWeights.map(emp =>**  **(**  **<div>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {weight.empWeight})}**  **</div>**  **)**  **)** |

Although we have one of the values we need, we still need to push it to the browser, for that we use the **return()** method again, along with a JSX element.

1. Wrap the **weight.empWeight** value inside of **<div>** tags and return the entire structure to the browser

|  |
| --- |
| **<div>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {**  **return <div>{weight. empWeight }</div>**  **})}**  **</div>** |

1. Lets add the date and construct a meaningful sentence

|  |
| --- |
| **this.state.allWeights.map(emp => (**  **<div>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {**  **return <div>**  **Date: {weight.weighedOn}**  **Weight: {weight.empWeight}**  **</div>**  **})}**  **</div>**  **))** |

1. Of course it would be better to shorten the date and insert a space betweeen these two values

|  |
| --- |
| **this.state.allWeights.map(emp => (**  **<div>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {**  **return <div>**  **Date: {new Date(weight.weighedOn).toLocaleDateString()}**  **{' '}**  **Weight: {weight.empWeight}**  **</div>**  **})}**  **</div>**  **))** |

1. If you check the console window, there should be an error about a unique key should be present. Lets add a key to satisfy the way **map()** works

|  |
| --- |
| **this.state.allWeights.map((emp,i) => (**  **<div key={i}>**  **{emp.empName}**  **{emp.employeeWeights.map(weight => {** |

1. Do the same for the next map() method, but use a different variable

|  |
| --- |
| **{emp.empName}**  **{emp.employeeWeights.map((weight,j) => {**  **return <div key={j}>**  **Date: {new Date(weight.weighedDate).toLocaleDateString()}** |

Here is the entire weightsmain.js file

|  |
| --- |
| **import React, { Component } from "react";**  **class TwMain extends Component{**  **constructor(props){**  **super(props);**  **this.state = {**  **allWeights: []**  **}**  **}**  **render(){**  **return(**  **<main>**  **{**  **this.state.allWeights.map((emp,i) =>**  **(**  **<div key={i}>**  **{emp.empName}**  **{emp.employeeWeights.map((weight,j) => {**  **return <div key={j}>**  **Date: {new Date(weight.weighedOn).toLocaleDateString()}**  **{' '}**  **Weight: {weight.empWeight}**  **</div>**  **})}**  **</div>**  **)**  **)**  **}**  **</main>**  **);**  **}**  **componentDidMount(){**  **fetch("http://localhost:8000/getweights")**  **.then(response => response.json())**  **.then(response => {**  **this.setState({**  **allWeights:response**  **})**  **})**  **}**  **}**  **export default TwMain** |

1. At the moment some of the CSS is not taking effect, so change the **div** within all of your container files to <div id="container">
2. You could give a page title to teamweights also