

## Making a Bookstore App using React

Last update April 9, 2021

Lesson 1 Introduction to React

Lesson 2 React Components

Lesson 3 Bookstore App Components

Lesson 4 Displaying Books for Sale

Lesson 5 Ordering Books and Shopping Cart

Lesson 6 Checkout Order

Lesson 7 Thank you Page and storing orders in a Firebase data base

### Conventions used in these lessons:

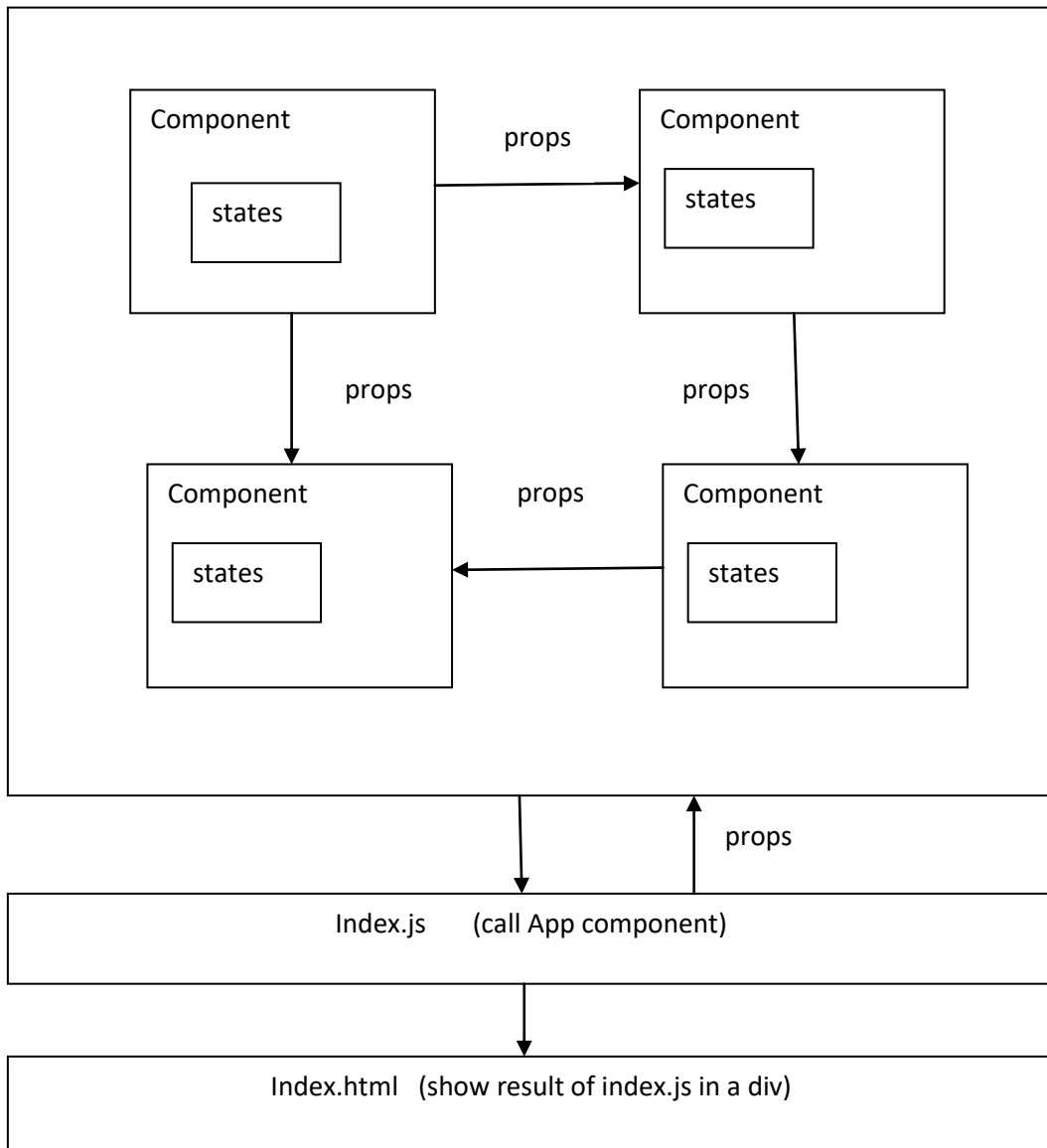
**bold** - headings, keywords, code

*italics* - code syntax

underline - important words

## Lesson1 Introduction to React

React let's you create web pages using components. Components contain data values known as states. Components also send data known as props (properties) to other components. Components display web content on the screen, using its render method. Components are like building blocks. A web page using React components can be visualized like this:



## Installing React

You need node.js and NPM to install React

### Step 1 Install node.js

<https://nodejs.org/en/download/>

if you all ready have node js installed on your computer you may want to reinstall to the latest version. React only work on node.js version s 10 or greater.

Node.js is an open source server environment.

Node.js allows you to run JavaScript on the Node.js server.

NPM is used to create and run React apps and to install additional modules.

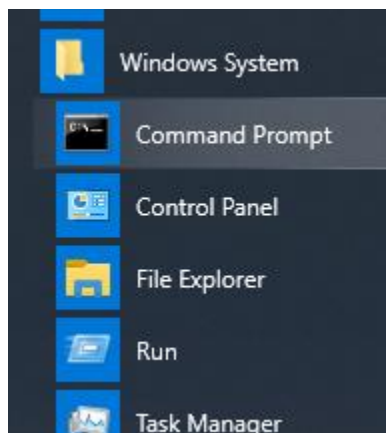
NPM is a package manager for Node.js packages and modules.

The NPM program is installed on your computer when you install Node.js

React uses node.js to create react apps and also uses modules provided by node.js

### Step 2 install create-react-app

From the start menu under Windows System open up a Command prompt



Navigate to the root of your drive

```
C:\Users\ADMIN>cd ..
```

```
C:\Users>cd ..
```

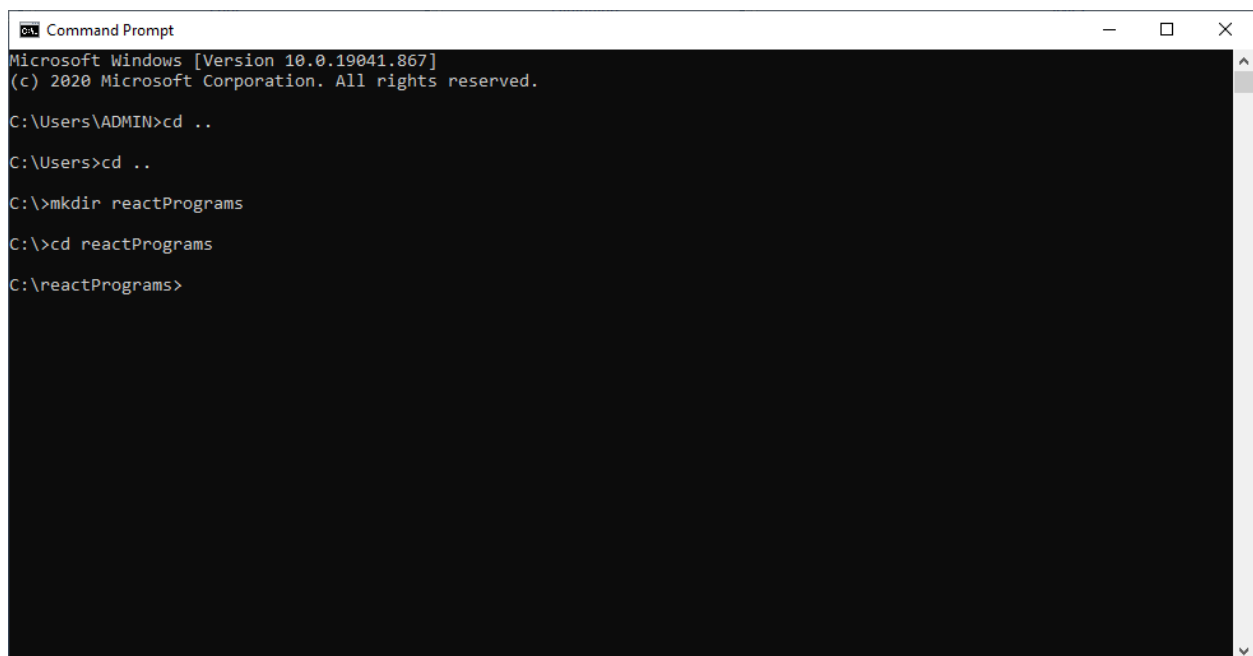
In the Command Prompt make a folder called `reactPrograms` using **mkdir** command

```
mkdir reactPrograms
```

(do not put any spaces in your folder name)

Navigate to this folder

```
cd reactPrograms
```

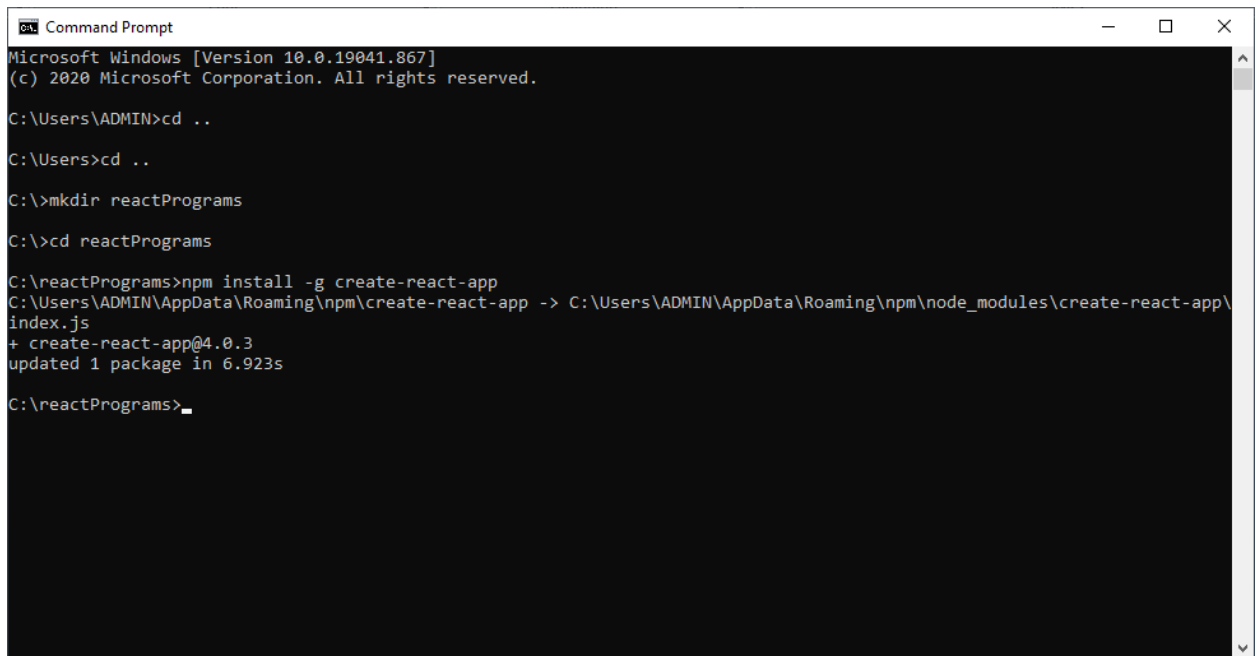


```
Command Prompt
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>cd ..
C:\Users>cd ..
C:\>mkdir reactPrograms
C:\>cd reactPrograms
C:\reactPrograms>
```

In the command prompt window type on:

**npm install -g create-react-app**



```
Command Prompt
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>cd ..
C:\Users>cd ..
C:\>mkdir reactPrograms
C:\>cd reactPrograms
C:\reactPrograms>npm install -g create-react-app
C:\Users\ADMIN\AppData\Roaming\npm\create-react-app -> C:\Users\ADMIN\AppData\Roaming\npm\node_modules\create-react-app\
index.js
+ create-react-app@4.0.3
updated 1 package in 6.923s
C:\reactPrograms>_
```

The create-react app command initialize the folder so you can create react apps.

### **Step 3 create react app**

On the windows prompt command line type

**npx create-react-app bookstore\_app**

It will start out like this:

```
Select Command Prompt
Microsoft Windows [Version 10.0.19041.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>cd ..
C:\Users>cd ..
C:\>mkdir reactPrograms
C:\>cd reactPrograms

C:\reactPrograms>npm install -g create-react-app
C:\Users\ADMIN\AppData\Roaming\npm\create-react-app -> C:\Users\ADMIN\AppData\Roaming\npm\node_modules\create-react-app\
index.js
+ create-react-app@4.0.3
updated 1 package in 6.923s

C:\reactPrograms>npx create-react-app bookstore_app

Creating a new React app in C:\reactPrograms\bookstore_app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

> core-js@2.6.12 postinstall C:\reactPrograms\bookstore_app\node_modules\babel-runtime\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

> core-js@3.9.1 postinstall C:\reactPrograms\bookstore_app\node_modules\core-js
```

If successful you will end up something like this:

```
Command Prompt
y"} (current: {"os":"win32","arch":"x64"})
removed 1 package and audited 1950 packages in 12.27s
found 0 vulnerabilities

Success! Created bookstore_app at C:\reactPrograms\bookstore_app
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

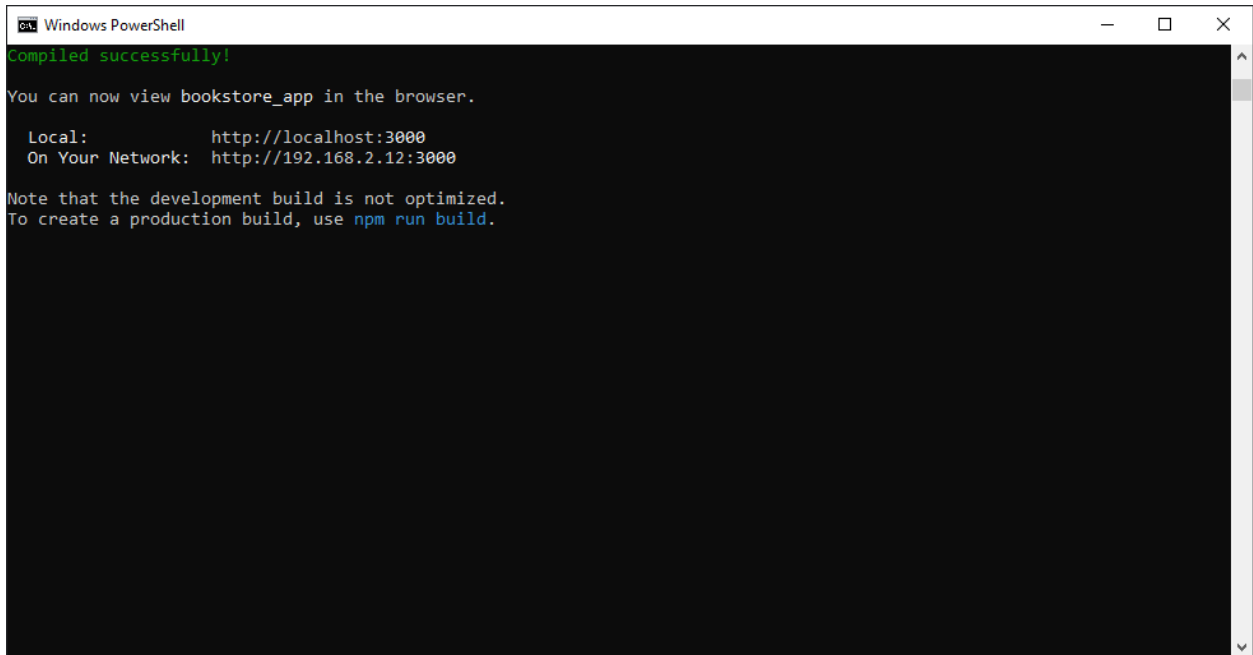
  cd bookstore_app
  npm start

Happy hacking!
C:\reactPrograms>
```

#### Step 4: run book store app

```
cd bookstore_app  
npm start
```

You will get something like this:



```
Windows PowerShell  
Compiled successfully!  
You can now view bookstore_app in the browser.  
Local:      http://localhost:3000  
On Your Network: http://192.168.2.12:3000  
Note that the development build is not optimized.  
To create a production build, use npm run build.
```

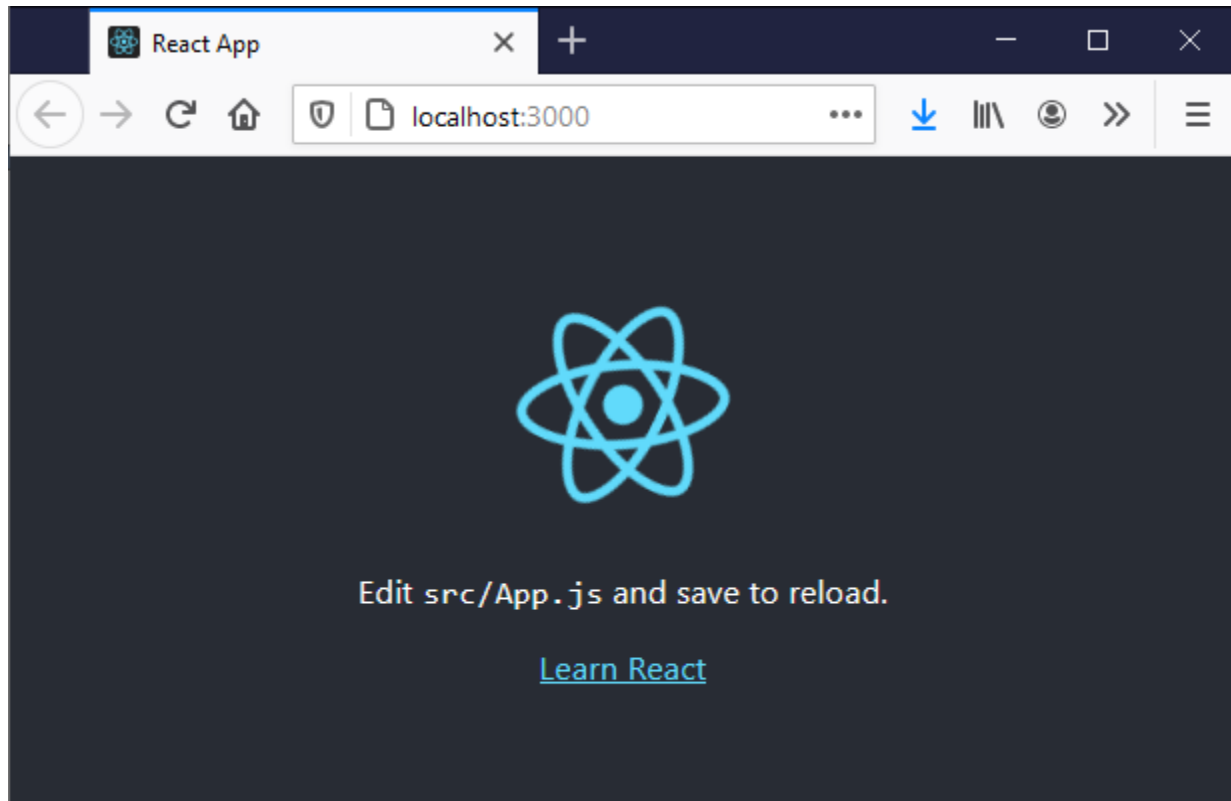
#### Step 5 view react bookstore app

Start your web browser at localhost: 3000

<http://localhost:3000/> actually is calling <http://localhost:3000/index.html>

Usually this is an automatic process, react usually automatically starts the web browser opened at localhost:3000. If it does not start automatically then go to your web browser and type localhost:3000 in the address bar.

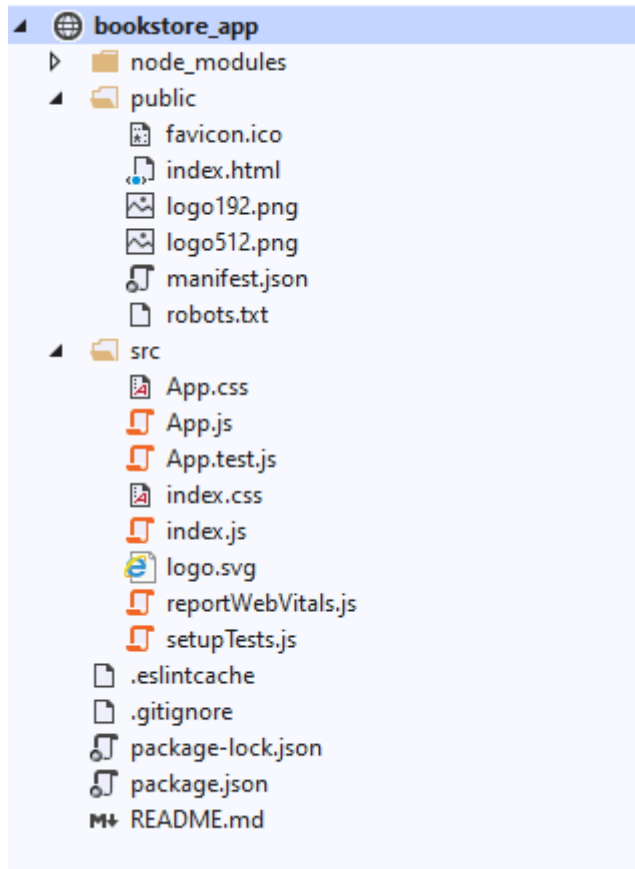
You should get something like this:



### **BookStore app file structure**

The Bookstore App has many created files. The public folder contains index.html what the web browser runs where as the src folders contains the index.js file that is used to display the web content rendered from the react components. The node\_modules contain all the modules that react will use. (too many files too show.





## Basic Folder Structure Explained

1. **package.json:** This File has the list of node dependencies which are needed.
2. **public/index.html:** When the application starts this is the first page that is loaded. This will be the only html file in the entire application since React is generally written using **JSX** which I will explain later. Also, this file has a line of code `<div id="root"></div>`. All the application components are loaded into this div.
3. **src/index.js:** This is the javascript file corresponding to index.html. This file has the following line of code which is used to call App component to display. `ReactDOM.render(<App />, document.getElementById('root'));` The above line of code is telling that **App** Component to be loaded into an html div element with id **root** located into the **index.html** file.

4. **src/index.css**: The CSS file corresponding to index.js.
5. **src/App.js** : This is the file for **App** Component. **App** Component is the main component in React which acts as a container for all other components.
6. **src/App.css** : This is the CSS file corresponding to **App** Component
7. **build**: This is the folder where the built files are stored. React Apps can be developed using either JSX, or normal JavaScript itself, but using JSX definitely makes things easier to code for the developer :). But browsers do not understand JSX. So JSX needs to be converted into javascript before deploying. These converted files are stored in the build folder after bundling and minification.

Here are the index.html, index.js and App.js files created by React. The App.js file renders the webpage contents and uses index.js to send it to the index.html file to be displayed in an div element.

// index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop.
      See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
```

```

<title>React App</title>
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
</html>

```

The index.js file basically just shows the react content by rendering the React App component image.

#### // Index.js

```

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

App.js for now just displays the React logo

#### // App.js

```

import logo from './logo.svg';
import './App.css';

```

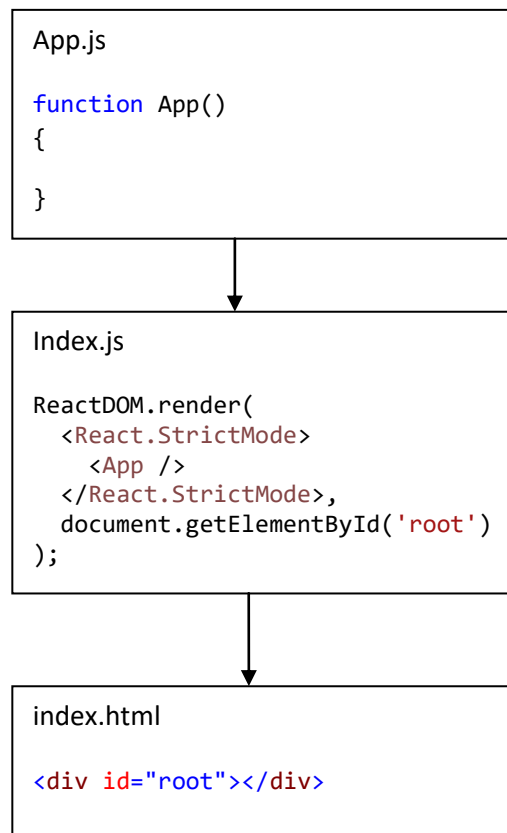
```

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

```

```
export default App;
```

Here is the program code flow



The index.js file renders the App component using the ReactDOM.render function.

```
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Using the <App /> component and a reference to the root div located in the index.html file.

React.StrictMode is used for highlighting potential problems in an application.

document.getElementById('root') is the element that is used to display the App.

## Lesson2 React Components

React components are used to build web pages. A Component display web page content. Components can be made from JavaScript classes or JavaScript functions.

React Components made from JavaScript classes are known as React Class Components.

React Components made from JavaScript functions are known as React Function Components.

A Component combines HTML tags and JavaScript to render content.

A React Class Component has a explicit **render** method where as a React Function Component does not and implicitly renders. The **render** method in the React Class Component returns HTML tags converted to JavaScript code that can be rendered. Like wise the React Function Component returns HTML tags converted to JavaScript code that can also be rendered. Rendering means displaying HTML tags on a web browser.

Here is a simple React Class Component that renders the message “I like React” using a <h1> tag.

```
import React, { Component } from 'react';

class MyApp extends Component {
  render() {
    return (

      <div>
        <h1>I Like React</h1>
      </div>

    );
  }
}
```

Here is React Component function that renders "I like React" also inside a <h1> tag.

```
function MyApp() {
  return (
    <div>
      <h1>I like React</h1>
    </div>
  );
}
```

The **return** keyword return's the HTML tags converted to JavaScript code that can render content.

Here is a functional component using the arrow function definition that renders "I like React". The arrow function is similar to the JavaScript function but omits the **function** keyword and the function code follows the => (arrow). They both do the same thing that renders "I like React" in a <h1> tag. The arrow function is just a short compact form of a JavaScript function definition.

```
const MyApp = () => {
  return (
    <div>
      <h1> I Like React </h1>
    </div>
  );
}
```

We use the **const** keyword rather than the **let** or **var** keyword because a function is considered a constant. **const** means code that will not be changed.

## JavaScript XML (JSX)

The HTML code in the Components render method is known as JSX. JSX is a syntax extension to JavaScript that produces React "elements" that is used for rendering content.

JSX allows us to write HTML tags directly within the JavaScript code. The JSX is translated into JavaScript code down to **React.createElement()** calls.

For example The html code:

```
<div>
  <h1> I like React</h1>
</div>
```

Would be translated into

```
React.createElement(
  "div",
  null,
  React.createElement(
    "h1",
    null,
    " I like React"
  )
);
```

JSX always need a top html tag element like <div>, if you want to include additional html tags.

```
<div>
  <h1> I like React</h1>
  <h1> I like Programming</h1>
</div>
```

## Rendering a Component

The **ReactDOM.render()** function is used to render a component.

The ReactDOM.render() function takes two arguments, a HTML code or React components and an HTML element.

The purpose of the render function is to display the specified HTML code inside the specified HTML element.

You can render a component in the index.js file by specifying the component name as a HTML tag and calling the **ReactDOM.render** method.



```
ReactDOM.render(  
  <React.StrictMode>  
    <MyApp />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

Alternately you can render JSX hardcoded

```
ReactDOM.render(<div>  
  <h1> I like React</h1></div>  
, document.getElementById('root'));
```

Or equate JSX to a const variable and render it.

```
const myElement = <div><h1> I like React</h1></div >;  
ReactDOM.render(  
  myElement  
, document.getElementById('root'));
```

## props

**Props** sends data to other components as parameters. **Props** stand for properties. Components can send props and receive **props**. **Props** cannot be changed once they are received and are read only.

In this example we send a name as a **prop** to the MyApp component.

```
<MyApp name="Tom Smith" />
```

Here is the complete render example sending an **prop** to a app

```
ReactDOM.render(  
  <React.StrictMode>  
    <MyApp name="Tom Smith" />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

The MyApp class component receives the props and renders the name from the props object members.

The name value from the props object is placed in curly brackets like this {props.name} and the <h2> is now this:

```
<h2>My Name is {props.name} </h2>
```

Our MyApp class component is now this:

```
import React, { Component } from 'react';

class MyApp extends Component {
  render() {
    return (
      <div>
        <h1>I like React</h1>
        <h2>My Name is {props.name}</h2>
      </div>
    );
  }
}
```

### Passing props to class components that have constructors

A constructor in a class component is used to receive prop and initialize other variables. If a class component has a constructor then it must receive props.

```
import React, { Component } from 'react';

class MyApp extends Component {

  constructor(props) {
    super(props);
  }
}
```

```

render() {
  return (
    <div>
      <h1>I like React</h1>
      <h2>My Name is {props.name}</h2>
    </div>
  );
}
}

```

functional components also receive props

```

function MyApp(props) {
  return (
    <div>
      <h1>I like React</h1>
      <h2>My Name is {props.name}</h2>
    </div>
  );
}

```

## LESSON2 Homework Question 1

Create the Name App in React. When you call the App component from index.js send the App Component the name, city and states as props. In the App component controller uses the received props display the name, state and city for rendering.

Note:

Although the web browser will automatically update when you make changes to your react code. there may be instances where you want to start fresh.

To return control back to the command prompt: type ctrl C [^C]

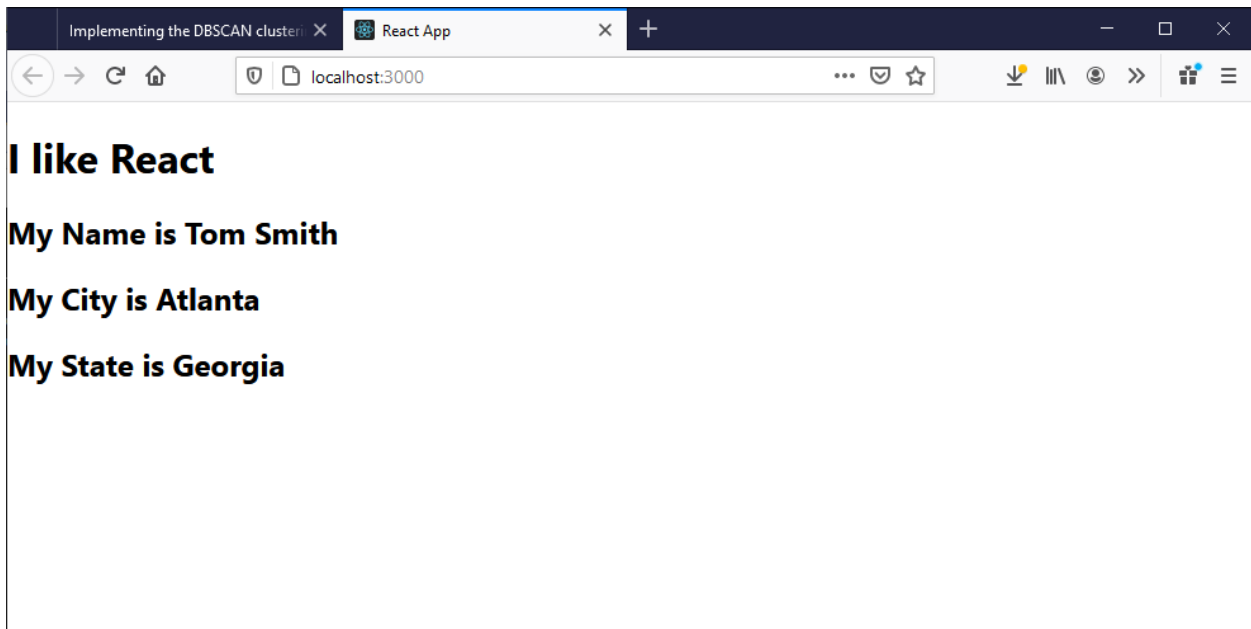
Terminate batch job (Y/N)? Y

And you will have the command line back again.

To restart the app type:

```
npm start
```

You should get something like this:



## States

A class component has a state object that allow a class component to store values. Here is a state object that stores information about a person.

```
this.state = {  
  name: "Tom Smith",  
  city: "Atlanta",  
  state: "Georgia"  
};
```

When the state object changes, the component re-renders by calling the **render** method. Only class components can have states. Functional components do not have states.

## reading state values

You can read values from the state object using the state object reference and the member variable, and embed into the HTML tag's like this

```
<h1>Name: {this.state.name} </h1>
```

Or use a constant like this

```
const name = this.state.name
```

then embed the const value into the HTML tag

```
<h1>Name: {name} </h1>
```

Alternatively you can read all the values from the state object into constant variables like this

```
const {name, state, city} = this.state
```

And then put into the HTML tags like this

```
<div>
  <h1>Name: {name} </h1>
  <h1>State: {state} </h1>
  <h1>City: {city} </h1>
</div>
```

## Changing state values

To change a value in the state object, use the **setState** method.

When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).

You will change a state value like this:

```
this.setState({name: "Sue Jones"});
```

Here is an example of a class component having a state to store details about a person. The render method will now display all the information about the person when it renders.

```

import React, { Component } from 'react';

class Person extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: "Tom Smith",
      city: "Atlanta",
      state: "Georgia"
    };
  }
  render() {

    const {name, state, city} = this.state

    return (
      <div>
        <h1>Name: {name} </h1>
        <h1>State: {state} </h1>
        <h1>City: {city} </h1>
      </div>
    );
  }
}

```

To use the above Person class it must be called from index.js or another Component

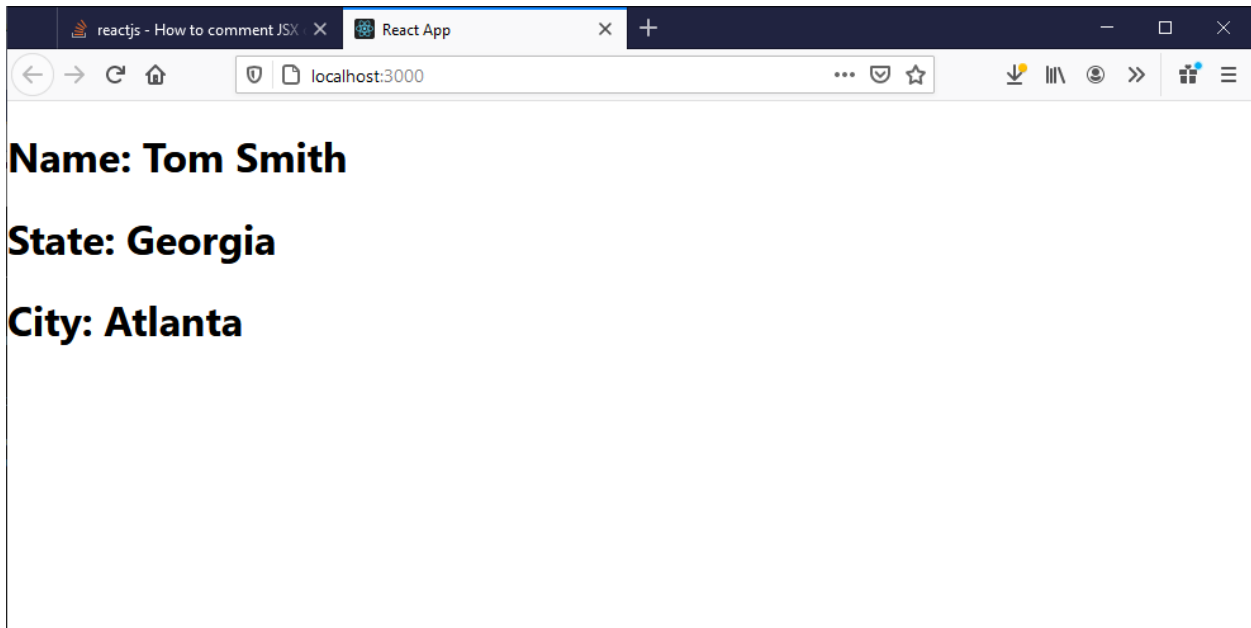
### Todo:

Change index.js so that it calls Person Component rather than App Component

You will need to put `import Person from './Person'`; at the top of your index.js file.

You do not need to send any values to the person class because it has all the values initialized in its state.

You should get something like this when you run your program.



### Initializing a Component state with props from another component

There may be situation when you want to initialize state data from props. Props can send data to a component that can be used to initialize it's state.

For example we can send data to a person component in the index.js and then the Person component can store this data in its state. The advantage is the props data that is sent to the Components can be stored in its state where it can be displayed and changed for later times. Prop data cannot be changed but state data can.

```
ReactDOM.render(  
  <React.StrictMode>  
    <Person name="Tom Smith" city="Atlanta" state="Georgia" />  
  </React.StrictMode>,  
  document.getElementById('root')  
);
```

The Person Component will receive the props from the index.js file when the Person component is called. Here is the Person Component that receives props and stores them it is state.

```

import React, { Component } from 'react';

class Person extends Component {
  constructor(props) {
    super(props);
    this.state = {
      name: props.name,
      city: props.city,
      state: props.state
    };
  }
  render() {

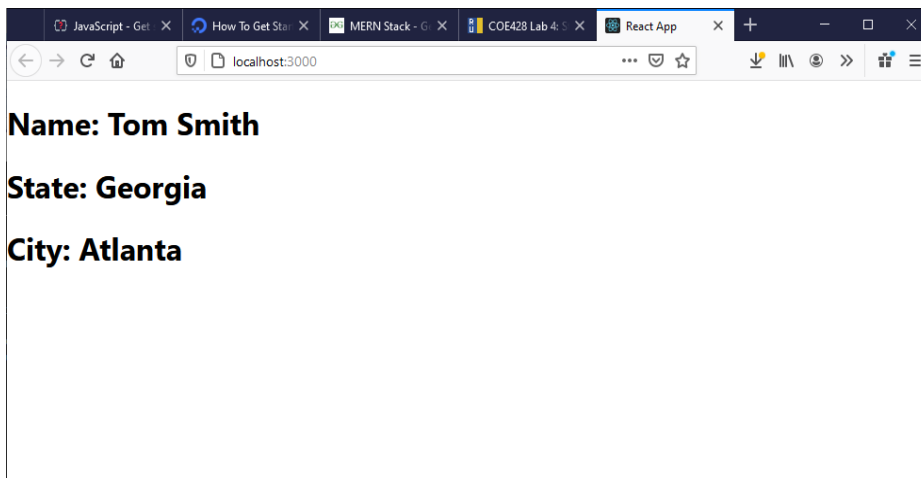
    const { name, state, city } = this.state

    return (
      <div>
        <h1>Name: {name} </h1>
        <h1>State: {state} </h1>
        <h1>City: {city} </h1>
      </div>
    );
  }
}

export default Person;

```

Running the program you should get something like this:

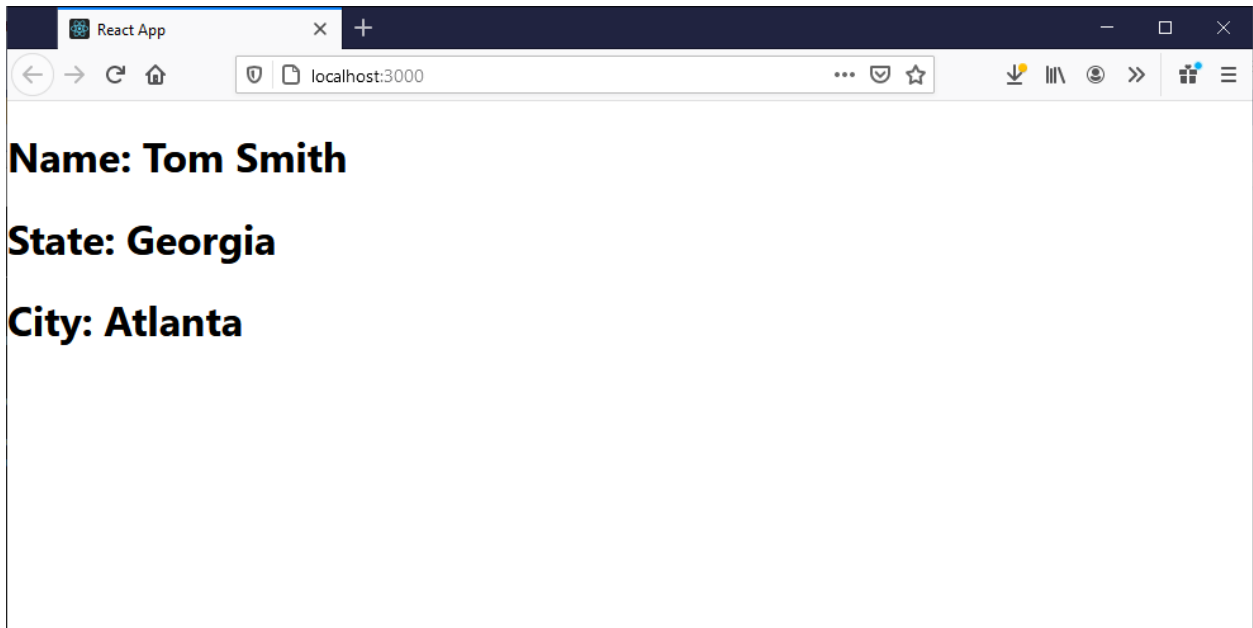




## LESSON2 Homework Question 2

Call the App component from index.js, send the name, city and states as props. In the App component controller send the received props to the Person Component and have the Person Component to store the name, state and city in its state for rendering.

You should get something like this:







## Lesson 3 BookStore App Components

We will have the following components:

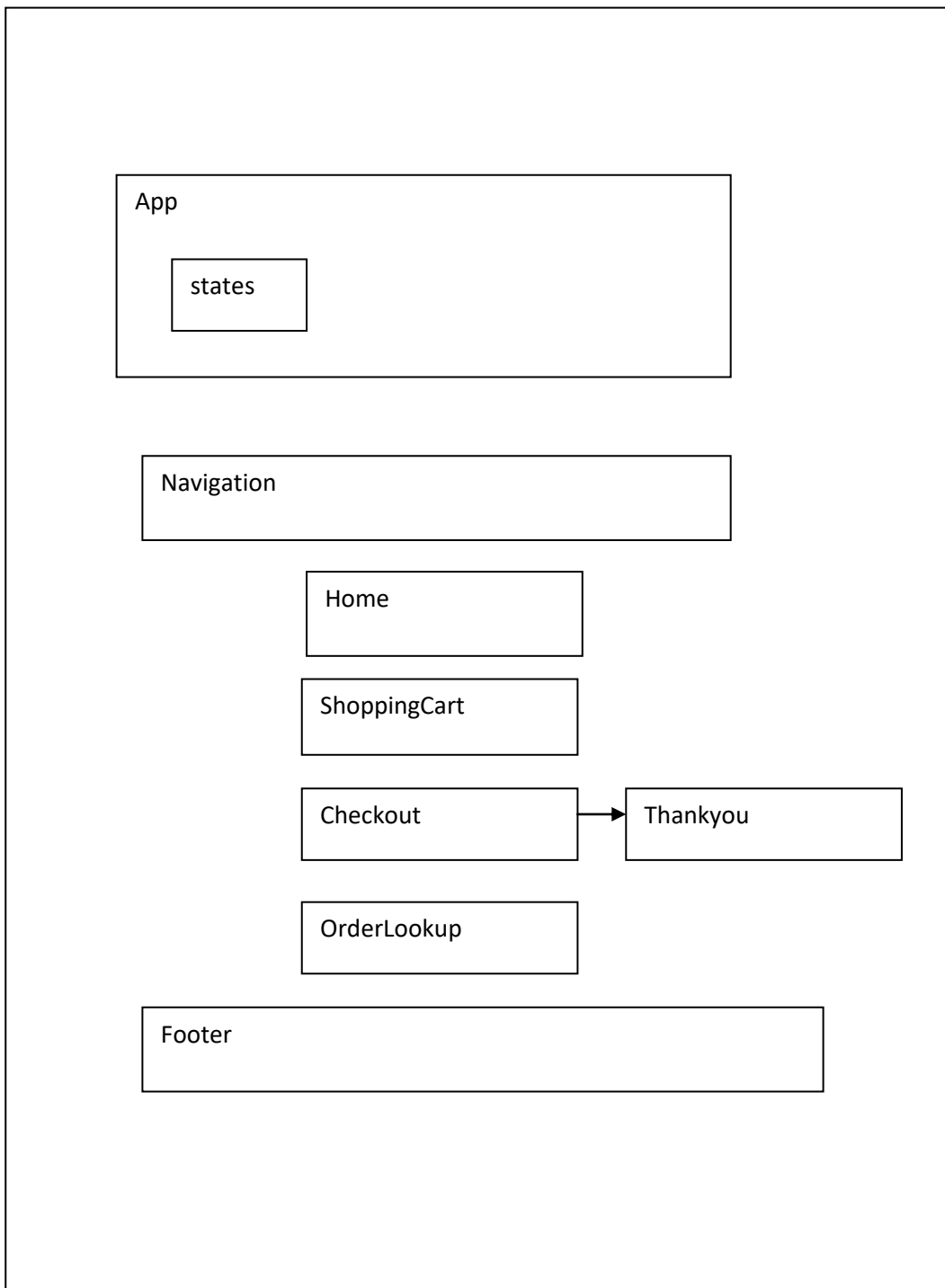
Component	Description
Home	Display books for sale
ShoppingCart	Display books ordered
Checkout	Enter customer details to finalize order
Thankyou	Show final order and store order in a Firebase data base
LookupOrders	Show previous orders stored in Firebase data base.

A finished Bookstore web page may look like this:

ISBN	Title	Author	Price	Image	Action
12345678	Alice in Wonderland	Lewis Carrol	\$23.67		Buy Book
876555445	Snow White and Seven Drawfs	Brothers Grimm	\$14.95		Buy Book
876555445	Peter Pan	Frank Baum	\$24.95		Buy Book
876555445	Wizard of Oz	J. M Barrie	\$29.95		Buy Book

Copyright (c) 2021 by booksRus.com

Our BookStoreApp Components block diagram:



## Routing

Our first step is to make the preliminary individual components and then link them up with navigation buttons. We will use routing to navigate between all components

You need to install the react-router-dom module from node.js into your application

Using NPM like this on the command line

```
npm install --save react-router-dom
```

The next step is to replace all the App.js code with the following code. The App Component will provide all the routing between all the other components. It will use a **BrowserRouter** Component, a nested **Navigation** Component and a **Switch** Component, and **Footer** component.

The **BrowserRouter** handles the routing. The **Switch** component is used to select the component to render. We will select between the **ShoppingCart**, **Checkout** and **OrderLookup** components. The **Checkout** component will automatically call the **Thankyou** page when the **Checkout** page submit button is pressed. The **Navigation** component produces the link buttons. The **Footer** component provides the Footer message.

Here is the App.js file that is use for navigation. Replace App.js in your src folder with this code:

```
// App.js
import './App.css';
import React, { Component } from 'react';
import { BrowserRouter, Route, Switch } from 'react-router-dom';
import Home from './Home';
import ShoppingCart from './ShoppingCart';
import Checkout from './Checkout';
import Thankyou from './Thankyou';
import Navigation from './Navigation';
import Footer from './Footer';
import OrderLookup from './OrderLookup';
```

```

class App extends Component {
  render() {
    return (
      <BrowserRouter>
        <div>
          <Navigation />
          <Switch>
            <Route path="/" component={Home} exact />
            <Route path="/shoppingCart"
              component={ShoppingCart} />
            <Route path="/checkout" component={Checkout} />
            <Route path="/orderlookup" component =
              {OrderLookup} /> />
            <Route component={Error} />
          </Switch>
          <Footer />
        </div>
      </BrowserRouter>
    );
  }
}

```

```
export default App;
```

Here is the Navigation component that provides the link buttons. The NavLink Component provides the Navigation button and the Component to link to. Make a Navigation.js file and put it into your src folder.

```

// Navigation.js

import React from 'react';
import { NavLink } from 'react-router-dom';

const Navigation = () => {
  return (
    <div>
      <NavLink to="/">| Home </NavLink>
      <NavLink to="/shoppingCart">| Shopping Cart </NavLink>
      <NavLink to="/checkout">| Checkout </NavLink>
      <NavLink to="/orderlookup">| Order Lookup |</NavLink>
    </div>
  );
}

```

```
export default Navigation;
```

copyright © 2021 [www.onlineprogramminglessons.com](http://www.onlineprogramminglessons.com) For student use only

The Footer component is just used to render the footer message. Make a Footer.js file and put it into your src folder.

```
// Footer.js

import React from 'react';

const Footer = () => {
  return (
    <div>
      <p>Copyright (c) 2021 by booksRus.com</p>
    </div>
  );
}

export default Footer;
```

The home Component will display the books for sale, but for now just prints the message "books for sale". Make a Home.js file and put it into your src folder.

```
// Home.js

import React, { Component } from 'react';

class Home extends Component {
  render() {
    return (
      <div>
        <h1>Books for Sale</h1>
      </div>
    );
  }
}

export default Home;
```

## Lesson3 Homework

Complete the skeleton components ShoppingCart, Checkout, Thankyou, OrderLookup and Error page. You can make a Functional Component like the Error Component like this:

```
// Error.js

import React from 'react';

const Error = () => {
  return (
    <div>
      <p>Error: Page does not exist!</p>
    </div>
  );
}

export default Error;
```

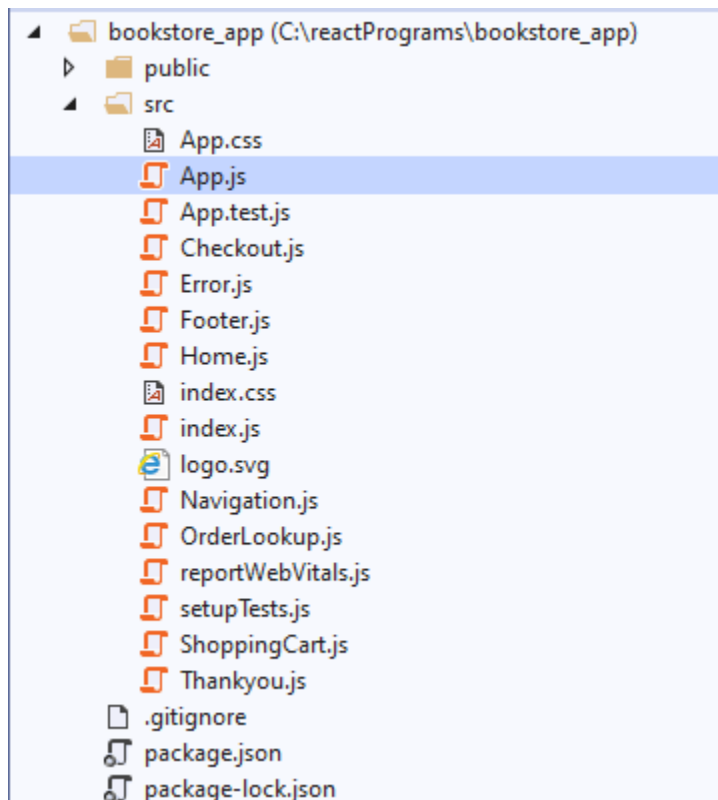
The Shopping Cart Functional Component would look like this:

```
// ShoppingCart.js
import React from 'react';
const ShoppingCart = () => {
  return (
    <div>
      <h1>Shopping Cart</h1>
      <p>Home page body content</p>
    </div>
  );
}
export default ShoppingCart;
```

Do the same for Checkout, Thankyou and OrderLookup and make the functional Components and put in your soured file.

A functional component does the same thing a class component does but it is more simpler to code and uses less overhead for React to execute. We use functional components just when we need to renders something simple that does not require **props** and **states**

Your src folder should now look like this:



Note:

To return back to the command type `ctrl C` [`^C`]

Terminate batch job (Y/N)? Y

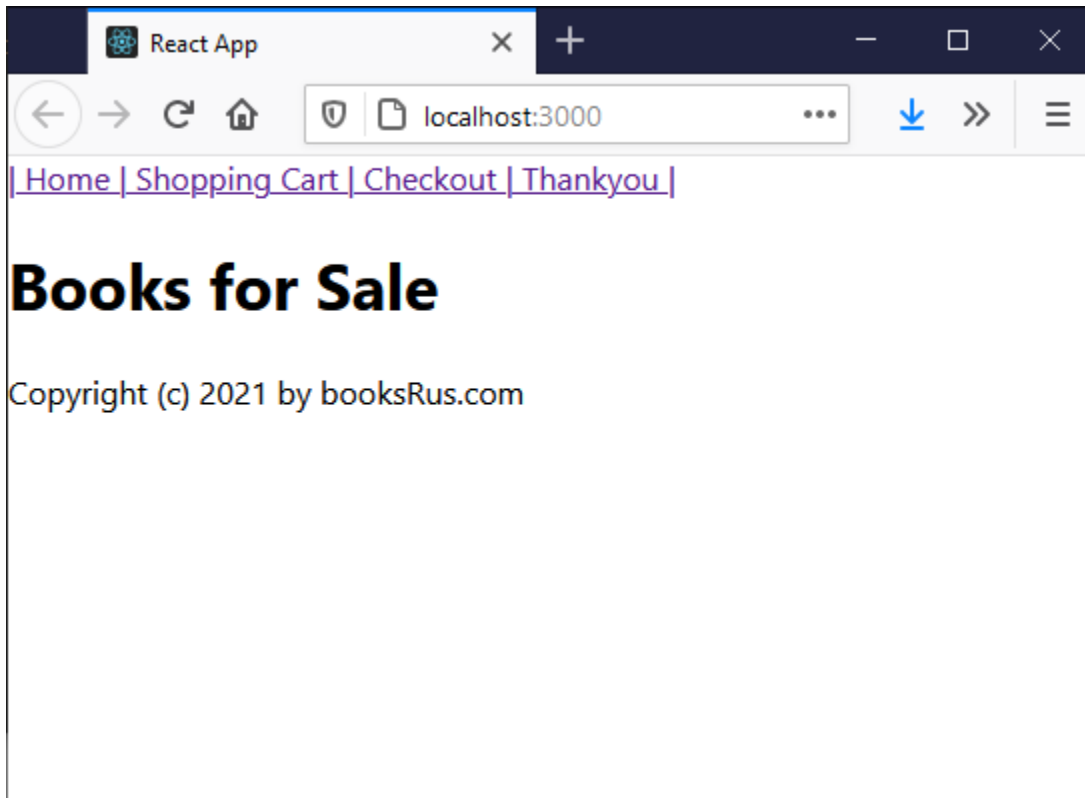
And you will have the command line back again.



We can now run the bookstore app:

```
npm start
```

You should see something like this

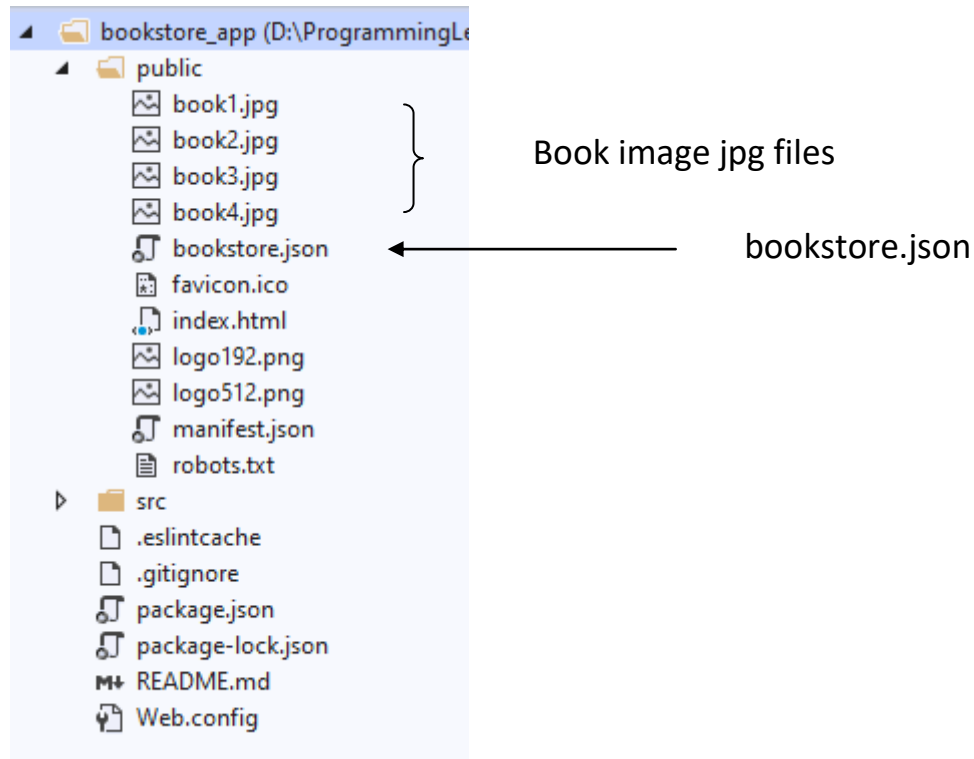


## Lesson4 Displaying Books for Sale

The **Home** Component will be used to display the books for sale. The **App** Component will first read a list of books contained in a **json** file and the Home component will display the books for sale in a html table. Storing the books for display in a json file is a good idea, since we only need to update the json file to display new books. Here is the bookstore.json file to be stored in the public folder.

```
{
  "books": [
    {
      "isbn": 123456789,
      "title": "Alice in Wonderland",
      "author": "Lewis Carrol",
      "price": 23.67,
      "image": "book1.jpg"
    },
    {
      "isbn": 876555445,
      "title": "Snow White and Seven Drawfs",
      "author": "Brothers Grimm",
      "price": 14.95,
      "image": "book2.jpg"
    },
    {
      "isbn": 234567544,
      "title": "Peter Pan",
      "author": "Frank Baum",
      "price": 24.95,
      "image": "book3.jpg"
    },
    {
      "isbn": 534536447,
      "title": "Wizard of Oz",
      "author": "J. M Barrie",
      "price": 29.95,
      "image": "book4.jpg"
    }
  ]
}
```

The bookstore.json file is stored in the public folder. Put the above bookstore.json file in the public folder. The public folder will also store the book images. You should download 4 book images from the internet, call them book1.jpg, book2.jpg, book3.jpg and book4.jpg. Put these jpg file also in the public folder where the bookstore.json file is. Your public folder should now look like this:



### CartItem object (CartItem.js)

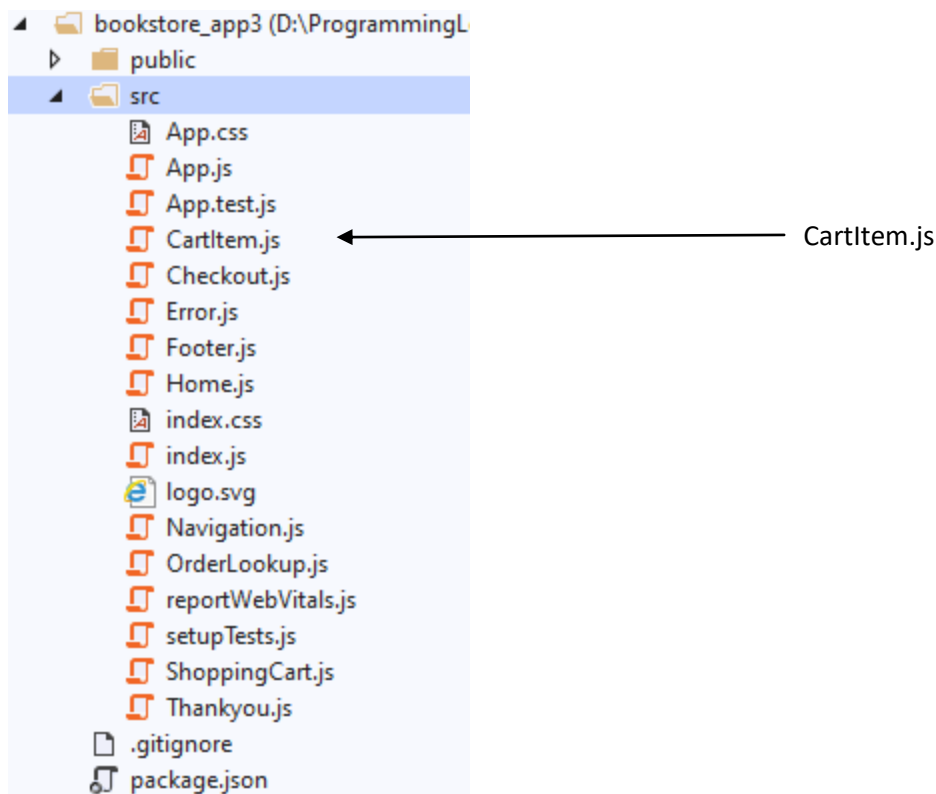
The CartItem object is used to store Book and Shopping Cart details. We make a JavaScript object, to store both the book and cart item details. The book and cart information is basically the same so we can use the CartItem object for both.

```
// CartItem.js
function CartItem(isbn, title, price, image, qty, total) {
  this.isbn = isbn;
  this.title = title;
  this.price = price;
  this.image = image;
  this.qty = qty;
  this.total = total;
}

export default CartItem;
```

**to do:**

Put the CartItem in a JavaScript file called CartItem.js in the **src** folder. Make sure you have **export default CartItem;** at the end of you file, so that your react project knows about this file.



## Updating App Component (App.js)

We now update the App Component code to read in the json file that contains the book. The App Component will store all the data for the book **store** app. This is a better approach rather than having each component store its own data. The App component will send the data to each component via **props**. The **App** component will also handle the **buy** and **remove** button clicks.

## App Component Constructor App.js

The App Component Constructor stores the books, cart and customer as state variables. Where the books state variable is an array to store the books from the book data loaded from the bookstore.json file. The cart state variable is an array to store the shopping cart item for books that have been ordered. The customer state variable is an object that will store the customer details. The App constructor will also receives props. The props are read only values that cannot be changed.

```
constructor(props) {  
  super(props)  
  this.state = {  
    books: [],  
    cart: [],  
    customer: {}  
  };  
}
```

**To do:** add constructor to your App component (App.js)

## Loading the book data from the bookstore.json file

Next we load in the json books file (bookstore.js) that has the books to display for sale. The **componentDidMount** method is used to load the books from the bookstore.json file, it is invoked immediately after a component is mounted. It uses the JavaScript **fetch** function to load the bookstore.json file.

```

// load books
componentDidMount() {
  fetch('./bookstore.json')
    .then(response => response.json())
    .then(data => {
      this.setState({ books: data.books });
    })
}

```

Once the data is read from the bookstore.json file the book store data is sent to the books array by calling the setState function. When the setState function is called, the **render** method is automatically called to display the books in a html table by calling the Home component.

**To-do:** Add the componentDidMount() methods to the App Component. Put the componentDidMount() method right after the App Component Constructor. Download 4 book images from the internet, call them book1.jpg, book2.jpg, book3.jpg and book4.jpg. Put these jpg file's in the public folder.

### Updating the App Component render method

We need the App component to send the book data to the Home Component as a prop so we can display the books for sale.

```

<Route path="/" render={() => <Home books={this.state.books} />} exact />

```

When we send a prop to a Component using Routing we use **render** rather than component = {Component name}

The complete **App** component **render** method will now look like this

```
render() {
  return (
    <BrowserRouter>
      <div>
        <Navigation />
        <Switch>
          <Route path="/" render={() => <Home books={this.state.books} />} exact />
          <Route path="/shoppingCart" component={ShoppingCart} />
          <Route path="/checkout" component={Checkout} />
          <Route path="/orderlookup" render={() => <OrderLookup />} />
          <Route component={Error} />
        </Switch>
        <Footer />
      </div>
    </BrowserRouter>
  );
}
```

### To do:

Update the render method in the App.js as shown above. Make sure you still have **export default App**; at the bottom of your App.js file.

Here is the Complete App.js file:

```
// App.js
import './App.css';
import React, { Component } from 'react';
import { BrowserRouter, Route, Switch } from 'react-router-dom';
import Home from './Home';
import ShoppingCart from './ShoppingCart';
import Checkout from './Checkout';
import Thankyou from './Thankyou';
import Navigation from './Navigation';
import Footer from './Footer';
import OrderLookup from './OrderLookup';

class App extends Component {

  constructor(props) {
    super(props)
    this.state = {
      books: [],
      cart: [],
      customer: {}
    };
  }

  componentDidMount() {
    fetch('./bookstore.json')
      .then(response => response.json())
      .then(data => {
        this.setState({ books: data.books });
      })
  }
}
```

```

render() {
  return (
    <BrowserRouter>
      <div>
        <Navigation />
        <Switch>
          <Route path="/" render={() => <Home books={this.state.books} />} exact />
          <Route path="/shoppingCart" component={ShoppingCart} />
          <Route path="/checkout" component={Checkout} />
          <Route path="/orderlookup" render={() => <OrderLookup />} />
          <Route component={Error} />
        </Switch>
        <Footer />
      </div>
    </BrowserRouter>
  );
}
}

export default App;

```

## rendering books data

We now update **Home** Component (Home.js) to display the book data Display books for sale. The first thing we do is add a constructor to receive the book data as a prop from the App component.

```

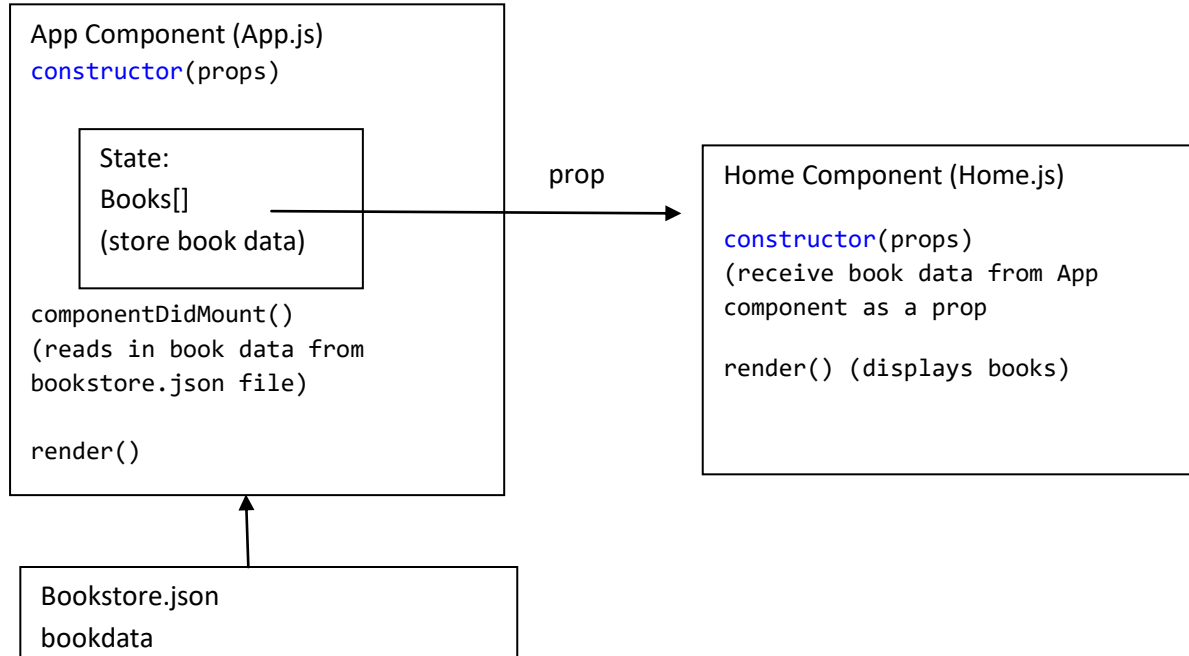
constructor(props) {
  super(props)
}

```

**To do:** Add the above constructor to your Home Component Home.js

The next thing we do we update the render method to display the book data in HTML tables. All the book data is sent from the App component as a prop. The book data was originally stored in a json file bookstore.json. The App component reads in the bookstore.json file in its **componentDidMount()** method and then stores the book data in its state as an array of books. The books array stores individual book objects having properties: isbn, title, author, price and image. The book objects are made automatically from the bookstore.json file when it is loaded in. You can visualize the data flow like this:





Here is the Home component render method:

```

render() {
  return (
    <div>
      <h1>Buy Books</h1>
      {
        <table>
          <thead>
            <th>ISBN</th>
            <th>Title</th>
            <th>Author</th>
            <th>Price</th>
            <th>Image</th>
            <th>Action</th>
          </thead>
          <tbody>
  
```

```

    {
    this.props.books.map((book)=> {
      return (
        <tr>
          <td>{book.isbn}</td>
          <td>{book.title}</td>
          <td>{book.author}</td>

          <td>${book.price.toFixed(2)}</td>
          <td>
            <img src={book.image} height='100' width='100' />
          </td>
          <td>
            <button onClick={(e) =>
              this.handleClick(book.isbn, e)}>Buy Book</button>
          </td>
        </tr>
      );
    })
  }
</tbody>
</table>
}
</div>
)
}

```

**To do:** Add the above render function to your **Home** component (Home.js) .  
 Make sure you still haven **export default Home;** at the bottom of you Home.js file.

You should now run the app




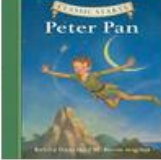
```
npm start
```

You should get something like this:

React App localhost:3000

[Home](#) | [Shopping Cart](#) | [Checkout](#) | [Order Lookup](#)

## Buy Books

ISBN	Title	Author	Price	Image	Action
12345678	Alice in Wonderland	Lewis Carrol	\$23.67		Buy Book
87655445	Snow White and Seven Drawfs	Brothers Grimm	\$14.95		Buy Book
87655445	Peter Pan	Frank Baum	\$24.95		Buy Book
87655445	Wizard of Oz	J. M Barrie	\$29.95		Buy Book

Copyright (c) 2021 by booksRus.com

Here is the complete Home Component Home.js file:

```
// Home.js
import React, { Component } from 'react';

class Home extends Component {
  constructor(props) {
    super(props)
  }
}
```

```

render() {
  return (
    <div>
      <h1>Buy Books</h1>
      {
        <table>
          <thead>
            <th>ISBN</th>
            <th>Title</th>
            <th>Author</th>
            <th>Price</th>
            <th>Image</th>
            <th>Action</th>
          </thead>
          <tbody>
            {
              this.props.books.map((book) => {
                return (
                  <tr>
                    <td>{book.isbn}</td>
                    <td>{book.title}</td>
                    <td>{book.author}</td>

                    <td>${book.price.toFixed(2)}</td>
                    <td>
                      <img src={book.image} height='100' width='100' />
                    </td>
                    <td>
                      <button onClick={(e) =>
                        this.handleClick(book.isbn, e)}>Buy Book</button>
                    </td>
                  </tr>
                );
              });
            }
          </tbody>
        </table>
      }
    </div>
  );
}

export default Home;

```

## Lesson 4 Homework

Add styling to the html table in the App.css file.

You can style tables like this:

```

td, th {
  border: 1px solid blue;
}

```





```

table {
  table-layout: fixed;
  width: 80%;
  border-collapse: collapse;
  border: 3px solid blue;
  text-align: center;
  margin: auto
}

```

You should get something like this:

The screenshot shows a web browser window titled 'React App' at the URL 'localhost:3000'. The page content includes a navigation menu with links for Home, Shopping Cart, Checkout, and Order Lookup. Below the navigation is a section titled 'Buy Books' containing a table with the following data:

ISBN	Title	Author	Price	Image	Action
12345678	Alice in Wonderland	Lewis Carrol	\$23.67		<input type="button" value="Buy Book"/>
876555445	Snow White and Seven Drawfs	Brothers Grimm	\$14.95		<input type="button" value="Buy Book"/>
876555445	Peter Pan	Frank Baum	\$24.95		<input type="button" value="Buy Book"/>
876555445	Wizard of Oz	J. M Barrie	\$29.95		<input type="button" value="Buy Book"/>

Copyright (c) 2021 by booksRus.com

## Lesson 5 Ordering Books and Shopping Cart

The **App** Component stores the books for sale, shopping cart and customer data in its state object.

Here is the App Component constructor and state object:

```
constructor(props) {
  super(props)
  this.state = {
    books: [],
    cart: [],
    customer: {}
  };
}
```

### Buying Books

The App Component has a **handleBuyClick** method that is used to store the ordered books in the **cart** member of the state object. The **handleBuyClick method** is called by the Home Controller when the buy book button is clicked. Here is the **handleBuyClick(isbn)** method:

```
// buy button clicked
handleBuyClick(isbn) {
  alert("buy: " + isbn);

  for (let i = 0; i < this.state.books.length; i++) {
    let book = this.state.books[i];

    if (book.isbn === isbn) {
      this.setState({
        cart:
          this.state.cart.concat
            (new CartItem(book.isbn, book.title,
              book.price, book.image, 1, book.price))
      });
    }
  }
}
```

When the **handleBuyClick** method is called by the Home Controller the books are looked up by the **isbn** number, then a new cart item is instantiated (created) with the book data. The newly created cartItem is added to the cart.

Note we have used cart concat rather than cart push.

```
//this.state.cart.push(new CartItem(book.isbn, book.title,  
//  book.price, book.image, 1, book.price));
```

**Todo:** Type in or copy/paste and put the handleBuyClick(isbn)  
In App.js right after the componentDidMount() method

You also need to put the CartItem import at the top of the App.js file

```
import CartItem from './CartItem';
```

## Sending the props to the Home Controller

The App Component in a **route** sends a reference to the book data as a **books prop** and a reference to buy book button event handler to the Home component as a **onClick prop**. Again we use **render** rather than **component** when we send **props** to a component when using a router.

```
<Route path="/" render={() => <Home books={this.state.books}  
onClick={(i) => this.handleBuyClick(i)} />} exact />
```

This is the reference to the book data books prop

```
books={this.state.books}
```

This is the reference to buy book button event handler onClick prop

```
onClick={(i) => this.handleBuyClick(i)} />
```

**Todo:** update the App component render method with the onClick **prop**.

## Handling the BuyButton click

The **Home** Component uses the book data stored in the App Controller to display the books for sale. The Home Component renders a Buy Book button that is used to buy a book when it is clicked. The Home Component has received a reference to the **handleBuyClick** method located in the App Controller as a **onClick prop**.

The Home component uses the reference to the **handleBuyClick** method of the App component to notify the App component to store the book selected when a buy button is clicked on.

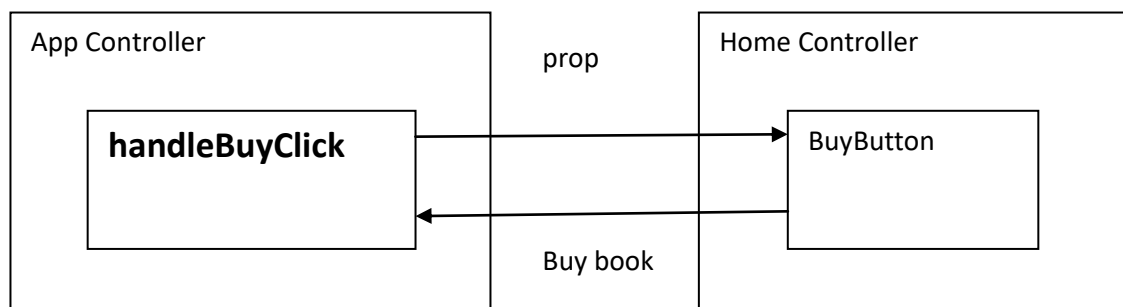
The home component references the **handleBuyClick** function reference as a received **onClick prop** in its render method.

```
<td>  
  <button onClick={(e) => this.props.onClick(book.isbn, e)}>Buy Book</button>  
</td>
```

**Note:** the button component **onClick** handler calls the **handleBuyClick** method using **this.props.onClick(book.isbn, e)** when the button is clicked.

**To do:** update this button tag in your Home Controller

When the buy button is clicked the **isbn** number is sent to the **handleBuyClick** function located in the App component. The App Controller uses the isbn number to locate the book data and then store a new cart item in the shopping cart.





## Shopping Cart Component

The shopping cart displays the books selected for purchase as well as total of the order.

The App Component sends the **cart** state member to the Shopping cart component as a cart **prop** so the books in the shopping cart can be displayed.

**Todo:** You need to update the Shopping cart router so that it sends the cart to the Shopping Cart Component.

```
<Route path="/shoppingCart" render={() => <ShoppingCart cart={this.state.cart} />}/>
```

Here is the complete Shopping Cart Code

```
// ShoppingCart.js
import React, { Component } from 'react';

class ShoppingCart extends Component {

  constructor(props) {
    super(props)
  }

  // display shopping Cart
  render() {
    return (
      <div>
        <h2>Your Shopping Cart</h2>
        {
          <table>
            <thead>
              <th>ISBN</th>
              <th>Title</th>
              <th>Price</th>
              <th>Image</th>
              <th>Quantity</th>
              <th>Total</th>
              <th>Action</th>
            </thead>

```

```

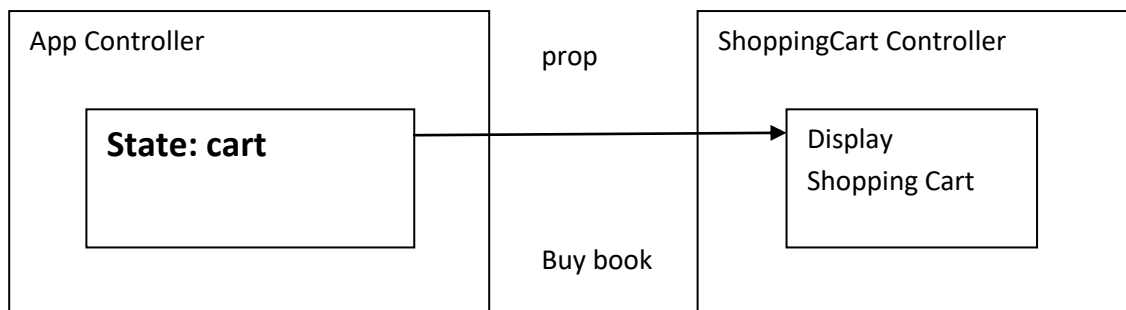
  {
    this.props.cart.map((item) => {
      return (
        <tr>
          <td>{item.isbn}</td>
          <td>{item.title}</td>

          <td>${item.price.toFixed(2)}</td>
          <td>
            <img src={item.image}
              height='100' width='100' alt={item.image} />
          </td>
          <td>{item.qty}</td>

          <td>${parseFloat(item.total).toFixed(2)}</td>
          <td><button onClick={e =>
            this.props.onClick(item.isbn, e)}>
            Remove Book</button>
          </td>
        </tr>
      );
    })
  }
</tbody>
</table>
}
</div>
)
}
}

```

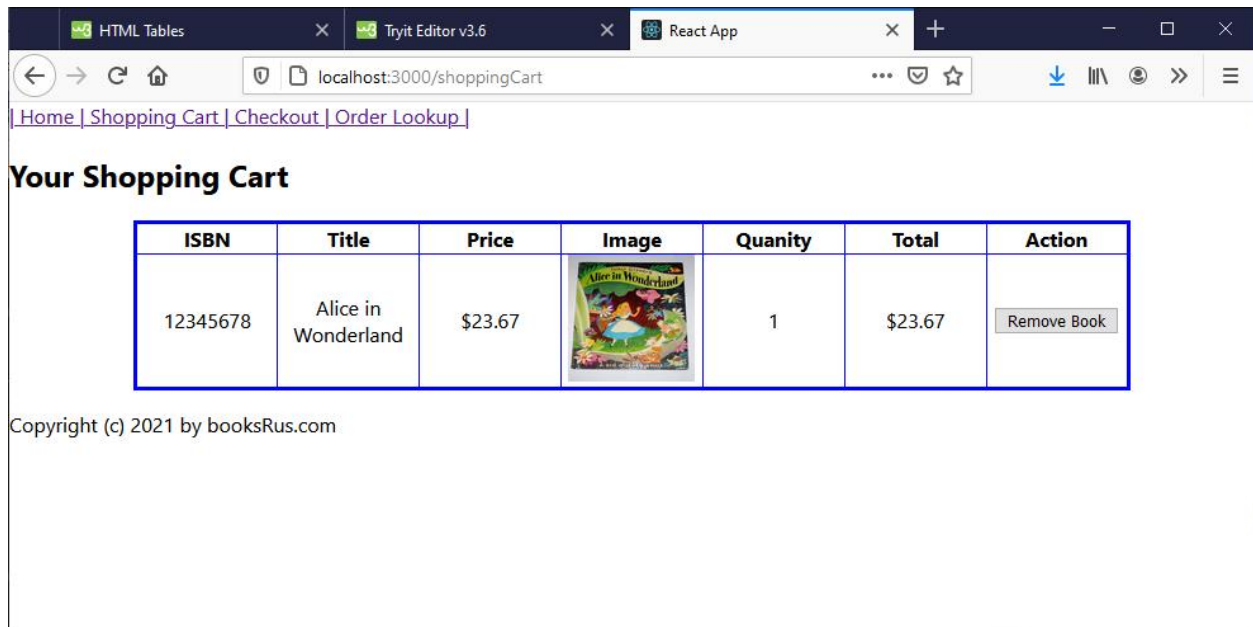
`export default ShoppingCart;`



**to do:**

Update the shopping cart code.

When you buy a book the shopping cart should look like this.



## Removing Books

The App Controller has the **handleRemoveClick** method to remove previous ordered books. The Shopping cart has a Remove Buy button that will remove a book if it is clicked.

The **handleRemoveClick** function receives the isbn number from the shopping cart, when the Remove button is clicked. Here is the **handleRemoveClick** method located in App Controller.

```

// remove button clicked for shopping cart
handleRemoveClick(isbn) {
  alert('remove: ' + isbn);

  for (let i = 0; i < this.state.cart.length; i++) {

    let item = this.state.cart[i];
    if (item.isbn === isbn) {
      let newcart = this.state.cart; // copy
      newcart.splice(i, 1);
      this.setState({ cart: newcart }); // render refresh
      break;
    }
  }
}
}

```

Basically when the **isbn number** is received it is searched for in the cart array and then removes from the cart when found.

**To do:** Type in or copy/paste in the **handleRemoveClick** method and put in App Controller right below the **handleBuyClick** method.

### Handling the RemoveButton click

The App component sends a reference to the **handleRemoveClick** method to the **ShoppingCart** component as a **onClick** prop.

```

<Route path="/shoppingCart" render={() => <ShoppingCart
cart={this.state.cart} onClick={(i) => this.handleRemoveClick(i)} />}/>

```

**To do:** update the **shoppingCart** route in the App component render method

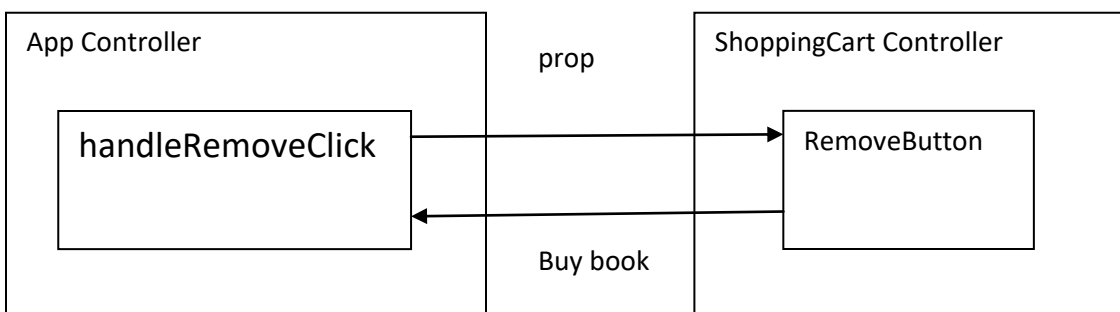
The **ShoppingCart** receives the **reference to the** **handleRemoveClick** method as a **onClick prop**.

The received **props** object contains the **onClick** member that has the reference to the **handleRemoveClick** function located in the App component.

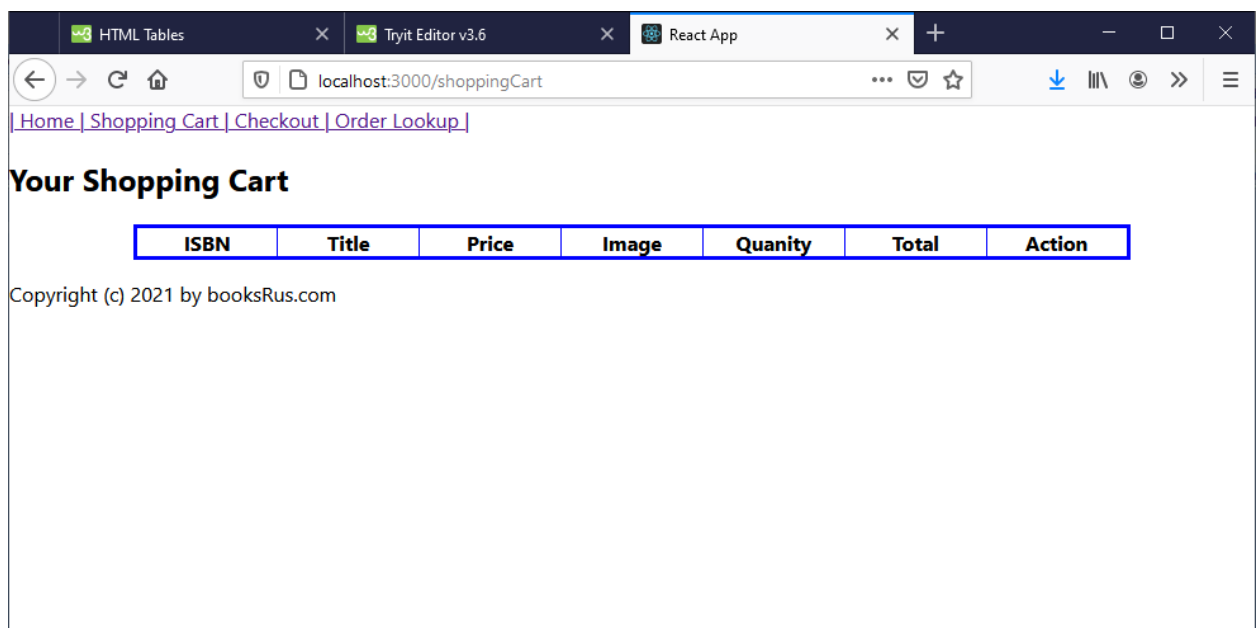
The Shopping Cart **onClick** event handler will send the isbn number of the book as well as the event **e** to the App component **handleRemoveClick** function.

```
<button onClick={e => this.props.onClick(item.isbn, e)}>Remove Book</button>
```

**Note:** the **button** component **onClick** handler calls the `removeBuyClick` method using `this.props.onClick(book.isbn, e)` when the **button** is clicked.



The Shopping cart will look like this when the book is removed:



## Here is the final App.js file

```
// App.js
import './App.css';
import React, { Component } from 'react';
import { BrowserRouter, Route, Switch } from 'react-router-dom';
import Home from './Home';
import ShoppingCart from './ShoppingCart';
import Checkout from './Checkout';
import Thankyou from './Thankyou';
import Navigation from './Navigation';
import Footer from './Footer';
import OrderLookup from './OrderLookup';
import CartItem from './CartItem';

class App extends Component {

  constructor(props) {
    super(props)
    this.state = {
      books: [],
      cart: [],
      customer: {}
    };
  }

  componentDidMount() {
    fetch('./bookstore.json')
      .then(response => response.json())
      .then(data => {
        this.setState({ books: data.books });
      })
  }

  // buy button clicked
  handleBuyClick(isbn) {
    alert("buy: " + isbn);

    for (let i = 0; i < this.state.books.length; i++) {
      let book = this.state.books[i];

      if (book.isbn === isbn) {
        this.setState({
          cart:
            this.state.cart.concat
              (new CartItem(book.isbn, book.title,
                book.price, book.image, 1, book.price))
        });
      }
    }
  }
}
```

```

// remove button clicked for shopping cart
handleRemoveClick(isbn) {
  alert('remove: ' + isbn);

  for (let i = 0; i < this.state.cart.length; i++) {

    let item = this.state.cart[i];
    if (item.isbn === isbn) {
      let newcart = this.state.cart; // copy
      newcart.splice(i, 1);
      this.setState({ cart: newcart }); // render refresh
      break;
    }
  }
}

render() {
  return (
    <BrowserRouter>
      <div>
        <Navigation />
        <Switch>

          <Route path="/" render={() => <Home books={this.state.books}
            onClick={i => this.handleBuyClick(i)} />} exact />
          <Route path="/shoppingCart" render={() => <ShoppingCart
            cart={this.state.cart} onClick={i => this.handleRemoveClick(i)} />} />
          <Route path="/checkout" component={Checkout} />
          <Route path="/orderlookup" render={() => <OrderLookup />} />
          <Route component={Error} />
        </Switch>
        <Footer />
      </div>
    </BrowserRouter>
  );
}
}

export default App;

```

## Lesson 5 Homework.

Make sure you can buy and remove books before proceeding.

Add the feature so that when the button is clicked the quantity increments if the book has been previously purchased before hand.

You need to copy the cart first, then adjust the quantities

```
let newcart = this.state.cart; // copy
```

then update the cart with the copy

```
this.setState({ cart: newcart }); // render refresh
```

Add the feature so that when a book is removed it decrements the quantity then the quantity is decremented. When the quantity reaches zero then remove the book from the cart.

You need to copy the cart first, then adjust the quantities

```
let newcart = this.state.cart; // copy
```

then update the cart with the copy

```
this.setState({ cart: newcart }); // render refresh
```

In the Shopping component break up the render method into reusable sub components constant functions. Put all subcomponents in the ShoppingCart.js file.



## Lesson 6 Checkout Order

The Checkout App displays a form where the customer enters all their details. The Checkout Component stores the customer data as a **state**. When the submit button is pressed, the customer data will be sent as a **prop** to the App Component for storage. The App component will then call the Thankyou page directly.

Here is the updated Checkout component

```
import React, { Component } from 'react';

class Checkout extends Component {

  constructor(props) {
    super(props)

    this.state = {
      name: '',
      street: '',
      city: '',
      state: '',
      email: '',
      phone: '',
      nameerr: '',
      streeter: '',
      cityerr: '',
      stateerr: '',
      emailerr: '',
      phoneerr: ''
    }
  }

  handleChange = event => {
    const { name, value } = event.target

    this.setState({
      [name]: value,
    })
  }
}
```

```

submitForm = () => {

    // validate form
    let error = false;
    if (this.state.name === "") {
        this.setState({ namerr: "please enter a name" })
        error = true;
    }
    else this.setState({ namerr: "" })

    if (this.state.street === "") {
        this.setState({ streeterr: "please enter a street" })
        error = true;
    }
    else this.setState({ streeterr: "" })

    if (this.state.city === "") {
        this.setState({ cityerr: "please enter a city" })
        error = true;
    }
    else this.setState({ cityerr: "" })

    if (this.state.state === "") {
        this.setState({ stateerr: "please enter a state" })
        error = true;
    }
    else this.setState({ stateerr: "" })

    if (this.state.email === "") {
        this.setState({ emailerr: "please enter a email" })
        error = true;
    }
    else this.setState({ emailerr: "" })

    if (this.state.phone === "") {
        this.setState({ phoneerr: "please enter a phone" })
        error = true;
    }
    else this.setState({ phoneerr: "" })
}

```

```

// no error
if (!error) {
  this.props.handleSubmit(
    {
      name: this.state.name,
      street: this.state.street,
      city: this.state.city,
      state: this.state.state,
      email: this.state.email,
      phone: this.state.phone
    }
  )
}

// display checkout form
render() {
  const { name, street, city, state, email, phone } = this.state;

  return (
    <form>
      <table>
        <tr><td>
          <label>Name</label>
        </td><td>
          <input
            type="text"
            name="name"
            value={name}
            onChange={this.handleChange} />
        </td>
        <td>
          {this.state.nameerr}
        </td>
        </tr>
        <tr>
          <td>
            <label>Street</label>
          </td>
        </tr>
      </table>
    </form>
  )
}

```

```

        <td>
            <input
                type="text"
                name="street"
                value={street}
                onChange={this.handleChange} />

        </td>
        <td>
            {this.state.streeterr}
        </td>
    </tr>

    <tr>
        <td>
            <label>City</label>
        </td>
        <td>
            <input
                type="text"
                name="city"
                value={city}
                onChange={this.handleChange} />

        </td>
        <td>
            {this.state.cityerr}
        </td>
    </tr>

    <tr>
        <td>
            <label>State</label>
        </td>
        <td>
            <input
                type="text"
                name="state"
                value={state}
                onChange={this.handleChange} />

        </td>
        <td>
            {this.state.stateerr}
        </td>
    </tr>

```

```

        <tr>
            <td>
                <label>Email</label>
            </td>
            <td>
                <input
                    type="text"
                    name="email"
                    value={email}
                    onChange={this.handleChange} />
            </td>
            <td>
                {this.state.emailerr}
            </td>
        </tr>

        <tr>
            <td>
                <label>Phone</label>
            </td>
            <td>
                <input
                    type="text"
                    name="phone"
                    value={phone}
                    onChange={this.handleChange} />
            </td>
            <td>
                {this.state.phoneerr}
            </td>
        </tr>
        <tr>
            <td colspan="2" >
                <input type="button" value="Submit" onClick={this.submitForm} />
            </td>
        </tr>
    </table>
</form>
    );
}
}

```

**export default** Checkout;

**todo:** type in or copy/paste the above whole Checkout component and put into the Checkout.js file. Replace the Checkout function with the entire Checkout component class.

### How the Checkout Component works:

Every time data is entered into one of the text boxes the data is collected by the **Handlechange** function. The received **event.target object** contains the textbox name and value. The appropriate customer data is then updated, by calling the **setState** function. When the **setState** function is called the Checkout component is re-rendered..

Here is the **handleChange** event function that receives and stores the check box data. It updates the state variable that has the same name as the text box with the received value.

```
handleChange = event => {
    const { name, value } = event.target

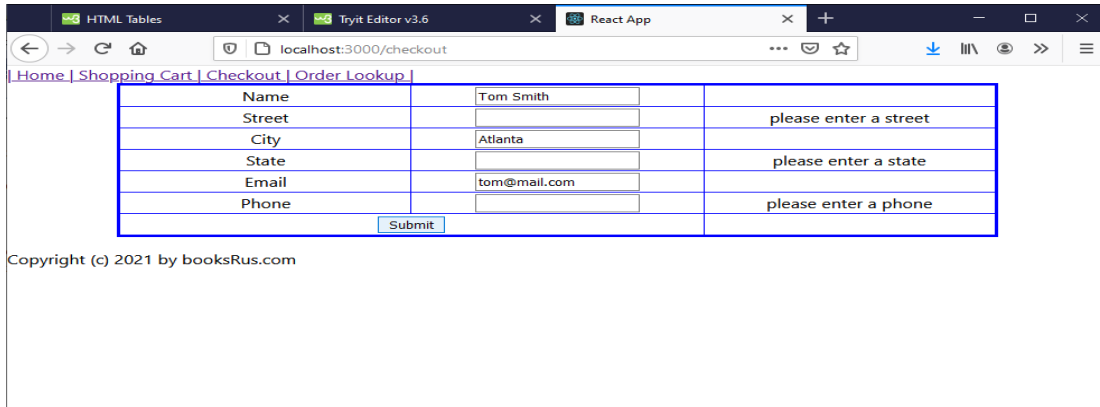
    this.setState({
        [name]: value,
    })
}
```

### submitting data

When the submit button is pressed the data is validated as shown in the following code snippet.

```
submitForm = () => {
    // validate form
    let error = false;
    if (this.state.name == "") {
        this.setState({ namerr: "please enter a name" })
        error = true;
    }
}
```

If there is an Error then the appropriate error message is set and then the Checkout Component renders and the error message is then displayed.



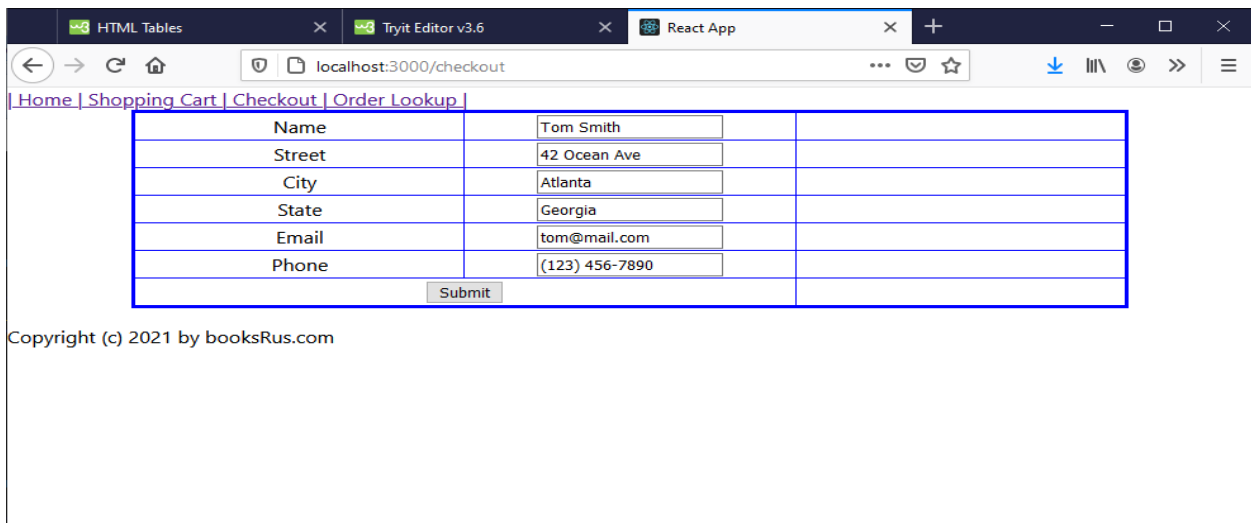
If there is no error then a customer object is created and the App component **handleSubmit** function is called using the **prop** reference.

The **handleSubmit** function handler is sent previously by the App component as a **prop** to the Checkout component when it is called by the router.

```
<Route path="/checkout" render={() =>
  <Checkout handleSubmit={this.handleSubmit} /> />
```

**Todo:** update the above checkout route in the app.js file

Checkout form with no errors:



If there is no Error then the customer data object is attached to a **prop** to be sent to and received by the App Component.

```

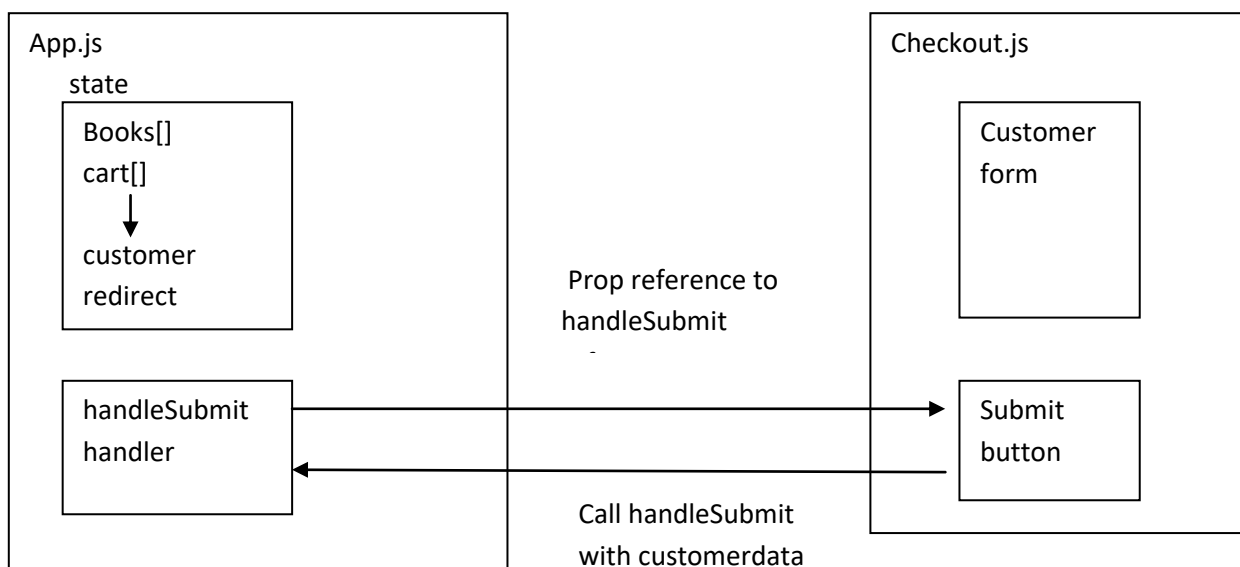
// no error
if (!error) {
  this.props.handleSubmit(
    {
      name: this.state.name,
      street: this.state.street,
      city: this.state.city,
      state: this.state.state,
      email: this.state.email,
      phone: this.state.phone
    }
  )
}

```

When the submit button in the Checkout Component is pressed the handle submit handler located in the App Component is called from the Checkout Component. The Checkout also sends the customer data as a prop to the App Component.

The handleSubmit function of the App component receives the customer object created previously by the Checkout component.

In the App component the stored shopping cart is attached to the received customer object and the customer object is then stored in the state.





Also in the handleSubmit function, the **redirect** state variable is also set to true. When the redirect state variable is true, then the App component is rendered and the thankyou page is displayed automatically. Here is the handleSubmit function of the App component.

```
// handle check out submit button
handleSubmit = customer => {
  alert("submit customer " + customer)
  //customer.order = Object.assign({}, this.state.cart );
  customer.order = this.state.cart
  alert(JSON.stringify(customer))
  this.setState({ customer: customer })
  this.setState({redirect:true})
}
```

**To do:** Add the handle Submit function handler to the App component located inApp.js Put right after the handleRemoveClick(isbn) function.

Here is the render method that includes the thankyou page redirect.

```
// display checkout form
render() {

  if (this.state.redirect) {
    return (
      <div>
        <Thankyou customer={this.state.customer}
          cart={this.state.cart} />;
      </div>);
  }

  return (
    <BrowserRouter>
      <div>
        <Navigation />
        <Switch>

          <Route path="/" render={() =>
            <Home books={this.state.books} onClick={(i) =>
              this.handleBuyClick(i)} />} exact />

          <Route path="/shoppingCart" render={() =>
            <ShoppingCart cart={this.state.cart}
              onClick={(i) => this.handleRemoveClick(i)} />}/>
        </Switch>
      </div>
    </BrowserRouter>
  );
}
```

copyright © 2021 [www.onlineprogramminglessons.com](http://www.onlineprogramminglessons.com) For student use only

```

    <Route path="/checkout" render={() =>
      <Checkout handleSubmit={this.handleSubmit} /> />

    <Route path="/orderlookup" render={() =>
      <OrderLookup />} />

    <Route component={Error} />
  </Switch>
  <Footer />
</div>
</BrowserRouter>
);
}
}

```

**Todo:** update your App component render method in App.js

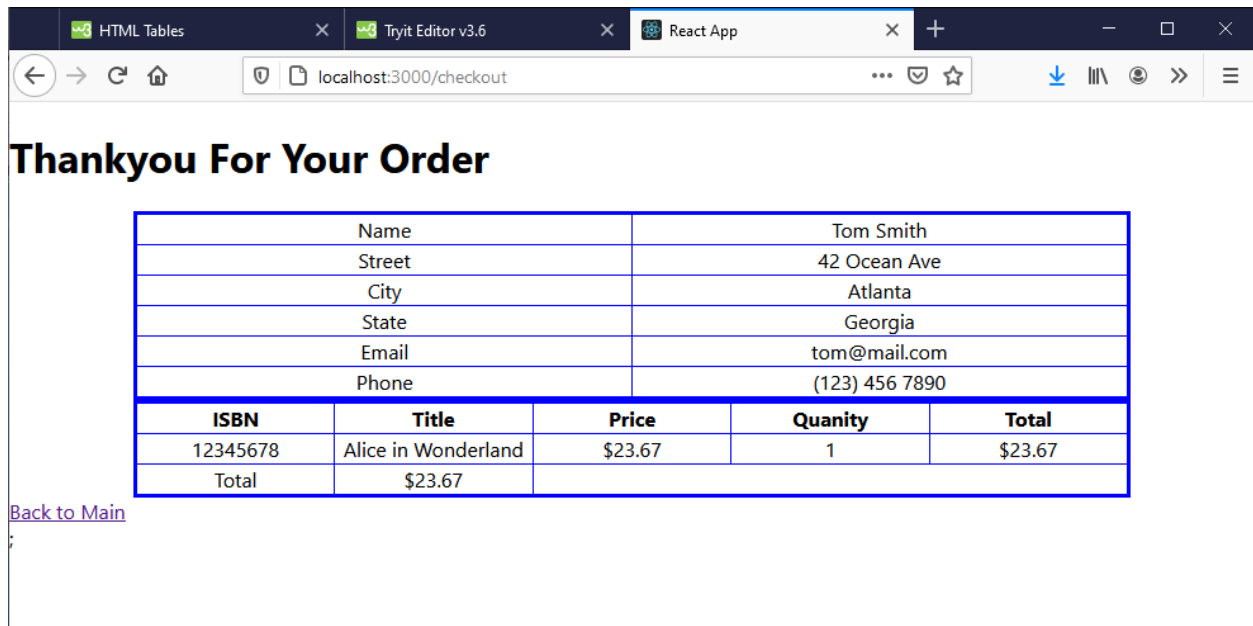
## Lesson 6 Homework

Before proceeding make sure everything is working.

In the Checkout component break up the render method into sub component constant functions. Put all subcomponents in the Checkout.js file.

## Lesson 7 Thank you page and storing orders in Firebase Database

The Thankyou page will display the customer details and the current book order.



The Thankyou page also stores the customer and book order in the firebase data base.

A database is permanent storage where order details can be stored retrieved again at a later time. We will store the customer details and book order in the firebase database.

We will use the Firebase RealTime Database rather than the cloud Firestore since it is more complicated to use. The RealTime Database also runs on the cloud and easily talks to React. The Firebase database uses JSON to store and retrieve records. To use the Firebase database, you will first need to register and obtain an **apiKey** and a **product-id** from the Firebase web site.

at <https://firebase.google.com/>

This may be a complicated process to do but I'm sure you will be able to get one, eventually. If not you probably you can use ours. You also need to get the configuration code for your project. The configuration code is displayed when you get your API key.

## Storing customer orders in the firebase data base

The first thing you need to do is add the Firebase configuration code in the Firebase.js file like this:

```
// firebase.js
import firebase from 'firebase'
import 'firebase/firestore';

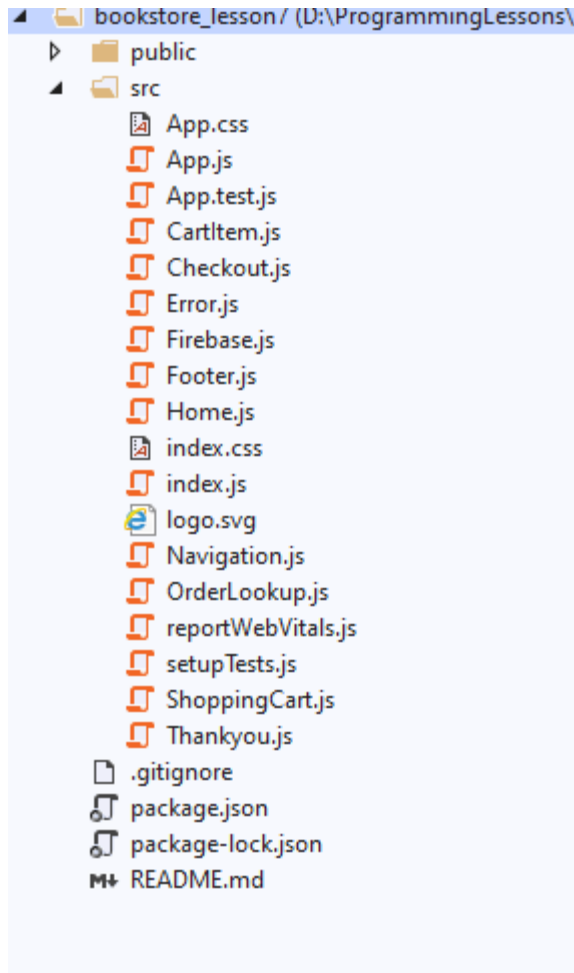
const config = {
  apiKey: "AIzaSyAyEsdsHgSA5v4PsgCviRv_fggK5zTimK4",
  authDomain: "testfire-5e6b0.firebaseio.com",
  databaseURL: "https://testfire-5e6b0.firebaseio.com",
  projectId: "testfire-5e6b0",
  storageBucket: ""
};

firebase.initializeApp(config);

export default firebase;
```

**todo:** put the above code in a Firebase.js java file in your src file

your src folder should now look like this:



Our apiKey is: **AlzaSyAyEsdsHgSA5v4PsgCviRv\_fgkK5zTimK4**

You can put yours in instead of ours.

Our project-id is: **testfire-5e6b0**

Later you will need to use your project-id rather than use ours.

The Firebase.js file must import firebase:

```
import firebase from 'firebase'  
import 'firebase/firestore';
```

We then initialize the data base with the above **config** object with this statement.

```
firebase.initializeApp(config);
```

You will also need to import the firebase module from the command line like this:

```
npm install firebase
```

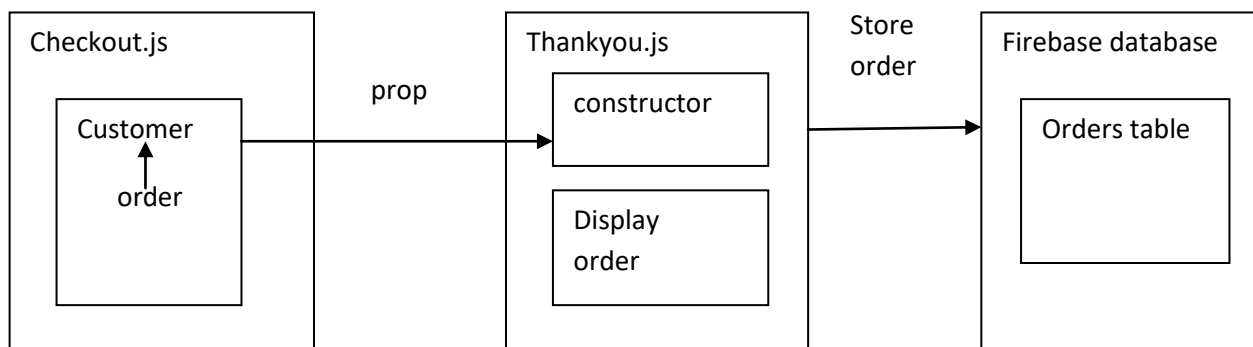
**To do:** install firebase module.

### Thankyou component constructor

The Thankyou component constructor will write the customer order details to the firebase database.

The Thankyou component constructor receives the customer details and book order through the **props** sent by the Thankyou page.

For convenience the received customer object contains the customer details and order details as a sub object combined together.



```
import React, { Component } from 'react';  
import firebase from './Firebase';
```

```
class Thankyou extends Component {
```

```
  constructor(props) {  
    super(props)
```

```

    // store order in data base
    const ref = firebase.database().ref();
    const orders = ref.child("orders");
    orders.push(this.props.customer)
    alert(JSON.stringify(this.props.customer))
  }

```

**todo:** Update Thankyou.js as a class Component and then type in or copy/paste the Thankyou constructor and add firebase import into your Thankyou.js file.

### How the Thankyou page constructor works

The customer object received from the **props** contains the customer details and the book order as a sub object in the customer object.

In the Thankyou component constructor we first get a reference to the firebase data base

```
const ref = firebase.database().ref();
```

We then get a reference to our data base table **“orders”**

```
const orders = ref.child("orders");
```

If the database table does not exist it is then automatically create for you, when we insert our first data record.

A data base may have many tables.

We then use the Firebase **push** function to put the customer order into the “orders” data base table.

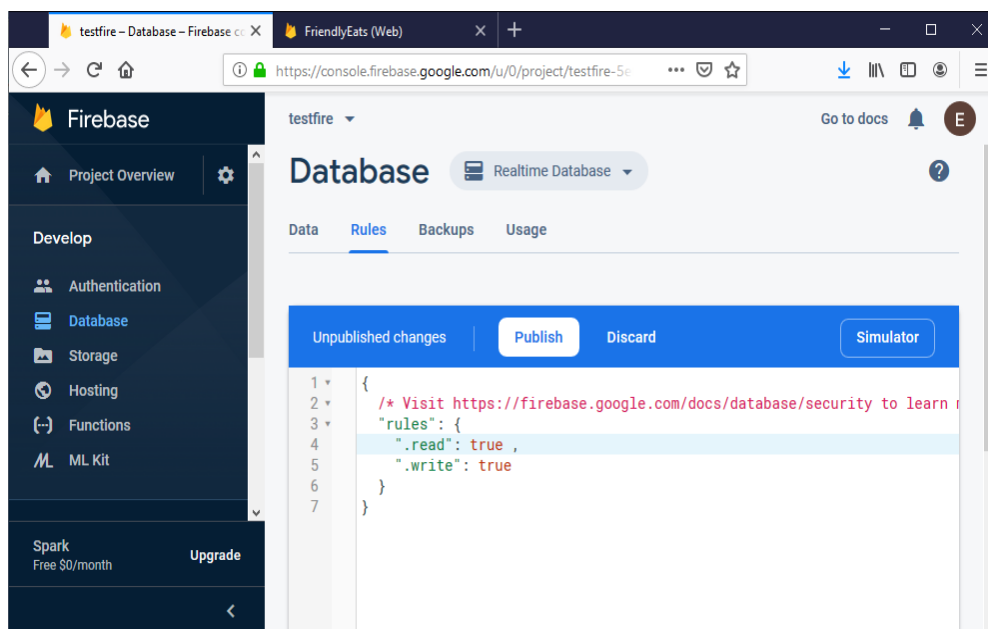
```
orders.push(this.props.customer)
```

Before running the program you need to set the **read** and **write** permissions for the data base. If you are using our firebase data base then this is already done for you.

Go to the firebase console and click on rules, and change read and write to true then click on the publish button that automatically appears. You can go to your console with this URL containing your project-id:

<https://testfire-5e6b0.firebaseio.com/>

The database console should appear like this:



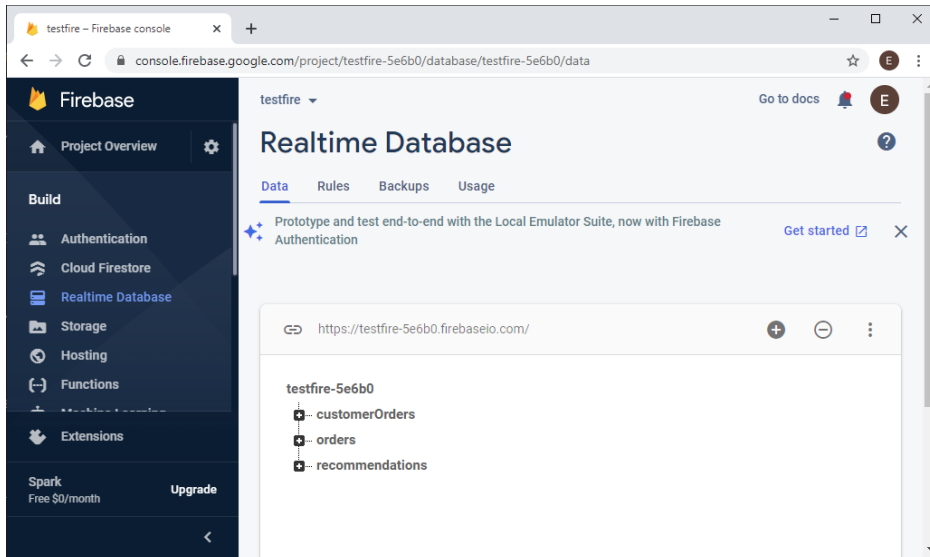
### Viewing data on the firebase data base

To view the tables in your firebase data base go to

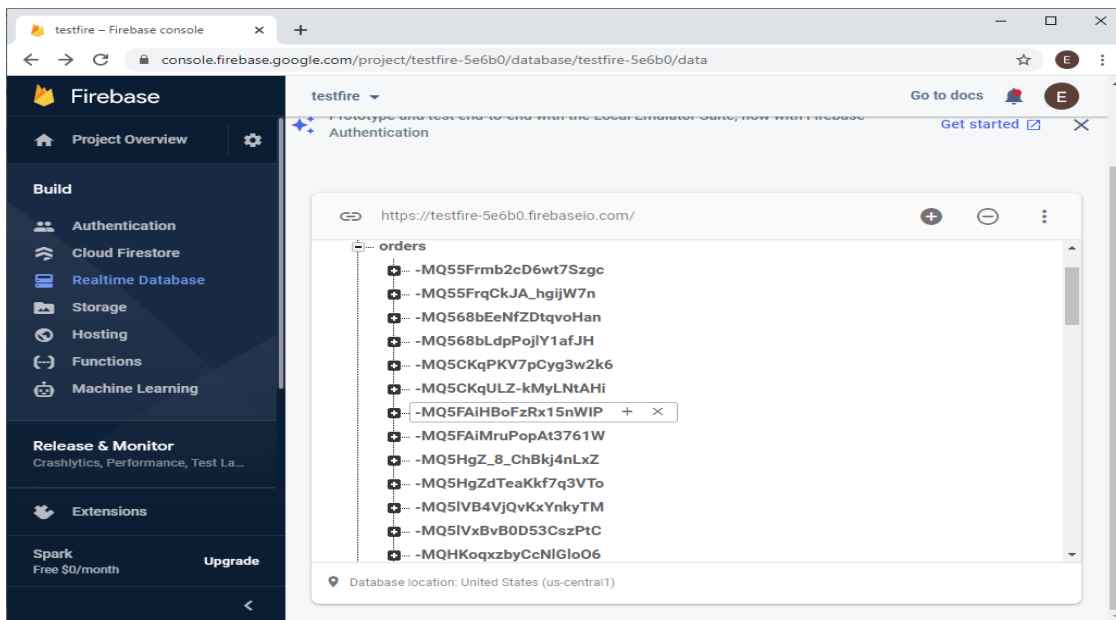
<https://testfire-5e6b0.firebaseio.com/>

You should get a list of table names like this

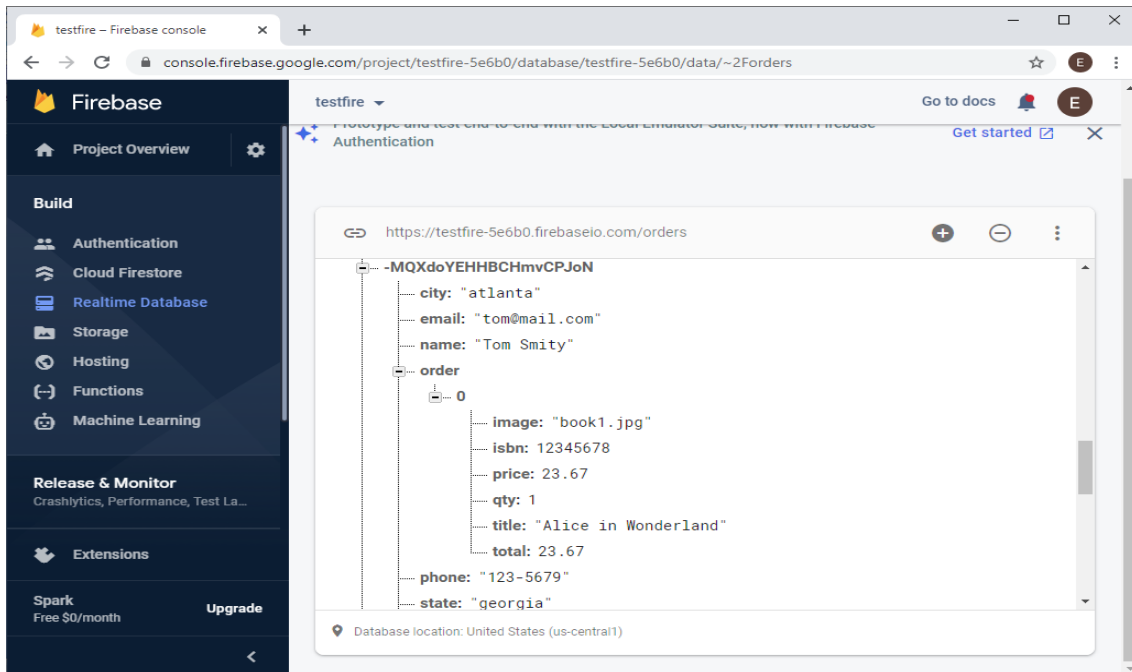




To see what's in the orders table click on orders. You will get a list of records id's like this.



To see what's in a particular record just click on one of the record ids and you will get something like this:



## Thankyou page Render method

The render method of the Thankyou page will calculate the order total then display the customer details and then the book order which is essentially the shopping cart.

```
// display order
render() {

  // calculate total
  let total = 0;
  this.props.cart.forEach((item) => {
    total += item.total
  });

  return (
    <div>
      <h1>Thankyou For Your Order</h1>

      <table>
        <tr><td>
          <label>Name</label>

        </td>

```

```

        <td>
            {this.props.customer.name}
        </td>
    </tr>

    <tr>
        <td>
            <label>Street</label>
        </td><td>
            {this.props.customer.street}

        </td>

    </tr>
    <tr>
        <td>
            <label>City</label>
        </td>

        <td>
            {this.props.customer.city}
        </td>
    </tr>

    <tr>
        <td>
            <label>State</label>
        </td>
        <td>
            {this.props.customer.state}
        </td>

    </tr>
    <tr>
        <td>
            <label>Email</label>
        </td>

        <td>
            {this.props.customer.email}
        </td>
    </tr>

    <tr>

```

```

        <td>
            <label>Phone</label>
        </td>

        <td>
            {this.props.customer.phone}
        </td>
    </tr>

</table>

{/* display cart in a table*/}
<table>

    <thead>
        <th>ISBN</th>
        <th>Title</th>
        <th>Price</th>
        <th>Quantity</th>
        <th>Total</th>
    </thead>

    <tbody>
    {
        this.props.customer.order.map((item) => {
            return (
                <tr>
                    <td>{item.isbn}</td>
                    <td>{item.title}</td>

                    <td>${item.price.toFixed(2)}</td>
                    <td>{item.qty}</td>

                    <td>${parseFloat(item.total).toFixed(2)}</td>

                </tr>
            );
        })
    }
</tbody>

```

```

        <tr>
          <td>Total</td>
          <td>${total.toFixed(2)}</td>
        </tr>

      </table>
      <a href="/">Back to Main</a>
    </div>
  );
}
}

```

`export default Thankyou;`

todo: type in or copy the render method in your Thankyou.js file.

Run the program. Your Thankyou page should look like this:



## Thankyou For Your Order

Name	tom smith			
Street	42 ocean ave			
City	atlanta			
State	grorgia			
Email	tom@mail.com			
Phone	967-1111			
ISBN	Title	Price	Quantity	Total
12345678	Alice in Wonderland	\$23.67	1	\$23.67
87655445	Wizard of Oz	\$29.95	1	\$29.95
Total	\$53.62			

[Back to Main](#)

;

## Lesson 7 Homework

### Question 1

Before proceeding make sure everything is working.

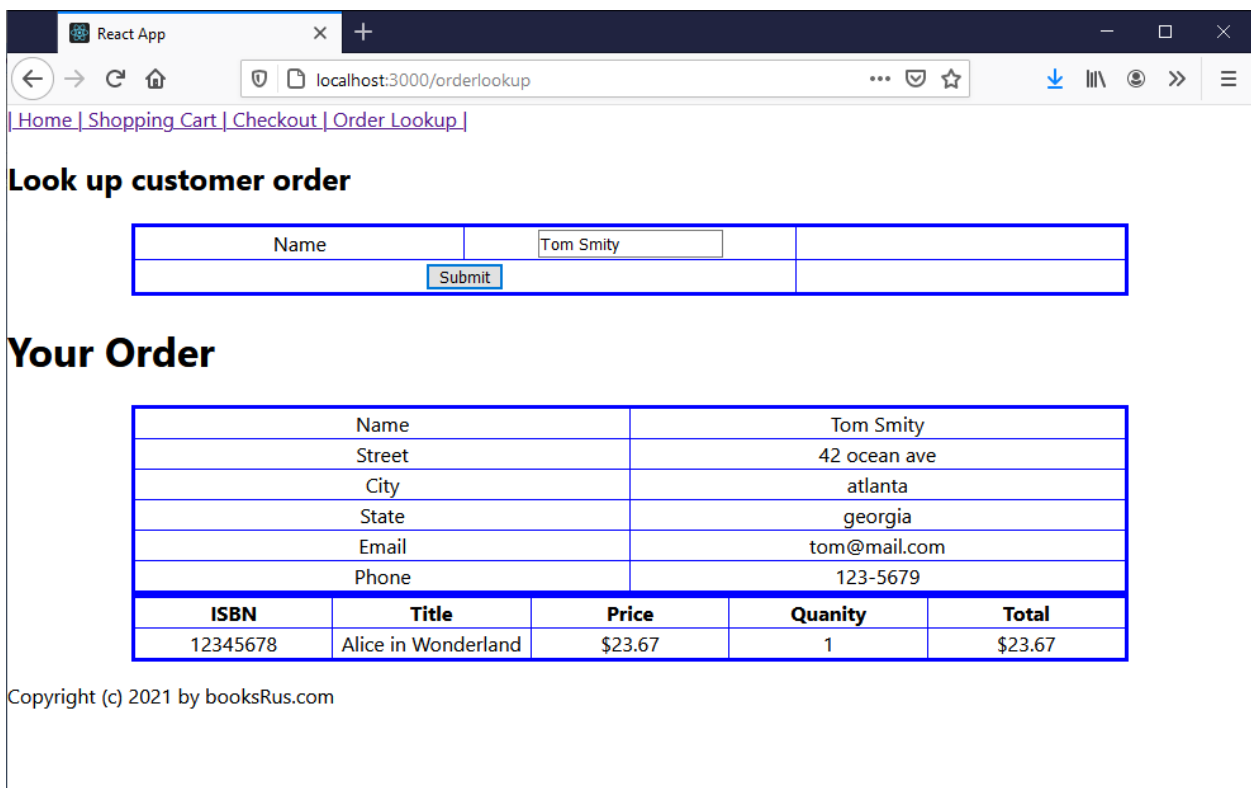
In the Thankyou component break up the render method into sub component constant functions. Put all subcomponents in the Thankyou.js file.

## Lesson 8 Lookup Orders

The Lookup Order Component allows the customer to look up previous order.

The orders were previously stored in the firebase database "orders" table.

All we need to do is have a small form to enter the customer name that when submitted will search the order table for the customer records stored in the orders table.



The screenshot shows a web browser window with the URL `localhost:3000/orderlookup`. The page has a navigation bar with links for Home, Shopping Cart, Checkout, and Order Lookup. The main content area is titled "Look up customer order" and contains a form with a "Name" input field containing "Tom Smity" and a "Submit" button. Below the form is a section titled "Your Order" which displays a table of order details.

Name	Tom Smity			
Street	42 ocean ave			
City	atlanta			
State	georgia			
Email	tom@mail.com			
Phone	123-5679			
ISBN	Title	Price	Quantity	Total
12345678	Alice in Wonderland	\$23.67	1	\$23.67

Copyright (c) 2021 by booksRus.com

The OrderLookup Component constructor **state** stores the lookup name, the name validation error message and select customer order object,

OrderLookup Component constructor also store a reference to the order table in our firebase data base.

```

// OrderLookup.js
// Look up customer order from data base
import React, { Component } from 'react';
import firebase from './Firebase';

class OrderLookup extends Component {

  constructor(props) {
    super(props);
    this.state = {
      name: '',
      namerr: '',
      customer: null
    };

    // reference to data base
    this.ref = firebase.database().ref();
  }
}

```

Todo: update the OrderLookup class and constructor

### handle change event handler

The name form capture's the lookup name in a handleChange event.

```

// get customer name
handleChange = event => {
  const { name, value } = event.target

  this.setState({
    [name]: value,
  })
}

```

**To do:** add the handle change event handler right after the OrderLookup constructor.



## Submit button handler

When the submit button is pressed, we get a reference to the orders data base table from the firebase database

```
const orderstable = this.ref.child("orders");
```

We need to store a reference to the this reference because it is not available in the call back function

```
const mythis = this; // store reference to this
```

The name is then extracted from the state object

```
const name = this.state.name.trim();
```

We then request to extract all the records from the orders table.

```
ordersdb.once("value", function (snapshot) {
```

The orders table is then extracted from the firebase data base.

```
const orders = snapshot.val()
```

We then search the orders table for the customer name. Once a customer name is found, the state customer is set with the customer object. and then the render method is called to render the customer order.

```
for (var order in orders) {
  const customer = orders[order];

  if (customer.name === name) {
    mythis.setState({ customer: customer });
    return;
  }
}
```

Here is the complete Submit function:

```
// submit button event handler
submitForm = () => {

    // validate form
    if (this.state.name === "") {
        this.setState({ namerr: "please enter a customer name" })
    }
    else {
        this.setState({ namerr: "" })

        // get orders table from data base
        const orderstable = this.ref.child("orders");
        const mythis = this; // store reference to this
        const name = this.state.name.trim();

        orderstable.once("value", function (snapshot) {

            const orders = snapshot.val()
            alert(JSON.stringify(orders))

            // look for customer name
            for (var order in orders) {

                // get customer order
                const customer = orders[order];

                // check for customer name
                if (customer.name === name) {
                    mythis.setState({ customer: customer });
                    return;
                }
            }

            // no customer found
            mythis.setState({ customer: null });

        });
    }
}
```

**Todo:** add the Submit function to the **OrderLookup** Component

## Render method

The render method displays the lookup name form if no customer is looked up, and displays the lookup name form and the customer order details when a customer is looked up. We have used an if statement in the render method to enforce this logic. Although there are other ways to do this like using the conditional and &&. We find using an if statement is more convenient than using the conditional and &&. We must not display the customer details when the lookup customer located in the state object is null.

The customer object contains the order details as a sub object customer.order;

Here is the render method

```
// display customer order
render() {

    const { name } = this.state;

    if (this.state.customer===null) {
        return (
            <div>
                <h2>Look up customer order</h2>
                <form>
                    <table>
                        <tr>
                            <td>
                                <label>Name</label>
                            </td>
                            <td>
                                <input
                                    type="text"
                                    name="name"
                                    value={name}

                                    onChange={this.handleChange} />
                                </td>
                            <td>
                                {this.state.nameerr}
                            </td>
                        </tr>
                    </table>
                </form>
            </div>
        );
    }
}
```

```

        <tr>
            <td colspan="2" >
                <input type="button"
value="Submit" onClick={this.submitForm} />
            </td>
        </tr>
    </table>
</form>
</div>
    );
}

return (
    <div>
        <h2>Look up customer order</h2>
        <form>
            <table>
                <tr><td>
                    <label>Name</label>
                </td><td>
                    <input
                        type="text"
                        name="name"
                        value={name}
                        onChange={this.handleChange} />
                </td>
                <td>
                    {this.state.nameerr}
                </td>
            </tr>
            <tr>
                <td colspan="2" >
                    <input type="button"
value="Submit" onClick={this.submitForm} />
                </td>
            </tr>
        </table>
    </form>

    <h1>Your Order</h1>
    <table>
        <tr><td>
            <label>Name</label>
        </td>
    </tr>
    </table>
    </div>
    );
}

```

```

        <td>
            {this.state.customer.name}
        </td>
    </tr>

    <tr>
        <td>
            <label>Street</label>
        </td><td>
            {this.state.customer.street}

        </td>

    </tr>

    <tr>
        <td>
            <label>City</label>
        </td>

        <td>
            {this.state.customer.city}
        </td>
    </tr>

    <tr>
        <td>
            <label>State</label>
        </td>

        <td>
            {this.state.customer.state}
        </td>

    </tr>
    <tr>
        <td>
            <label>Email</label>
        </td>

        <td>
            {this.state.customer.email}
        </td>
    </tr>

```

```

        <tr>
          <td>
            <label>Phone</label>
          </td>

          <td>
            {this.state.customer.phone}
          </td>
        </tr>

</table>

{/* display cart in a table*/}
<table>

  <thead>
    <th>ISBN</th>
    <th>Title</th>
    <th>Price</th>
    <th>Quantity</th>
    <th>Total</th>
  </thead>

  <tbody>
    {
      this.state.customer.order.map((item) => {
        return (
          <tr>
            <td>{item.isbn}</td>
            <td>{item.title}</td>
            <td>${item.price.toFixed(2)}</td>
            <td>{item.qty}</td>
            <td>${parseFloat(item.total).toFixed(2)}</td>
          </tr>
        );
      })
    }
  </tbody>
</table>
</div>
);
}
}
}

```

```
export default OrderLookup;
```

**todo: add the render method to the `OrderLookup` Component**

## Lesson8 Homework

### Question 1

Before proceeding make sure everything is working.

In the `OrderLookup` component break up the render method into sub components functions . Put all subcomponents in the `Checkout.js` file.

### Question 2

In the render method replace the if statement with the conditional and `&&`.

### Question 3

In situations where there are many customer orders with the same name show all book orders for that customer. You can use a next button or just show all the different orders.

End