React Tutorial

Implementing a simple React application

Express is a React is a JavaScript library used for building fast and interactive user interfaces. It was developed by Facebook in 2011 and is currently the most popular JavaScript library for building user interfaces. React is made of Components which are simply pieces of the user interface. When building React applications, you create a series of independent, isolated, and re-usable components that are then combined to create complex user interfaces. A component is typically implemented as a JavaScript class that has some state and a render method. The state is the data that we want to display when the component renders. The render method is responsible for describing what the UI should look like. The output of the render method is a React element which is a simple plain JavaScript object that maps to a DOM element.

React keeps a copy of the actual DOM in memory and is called the Virtual DOM. When a change is made to our data inside of our React application, the Virtual DOM will compare itself and its changes to the actual DOM being displayed to the user. If React finds that the Virtual DOM has changed, it will then automatically update the actual DOM to reflect those changes.

Stated simply: change the state of our components and React will automatically find where we were using that data on the HTML DOM and update it immediately. This replaces the need for jQuery, selectors, and binding event listeners to HTML DOM elements. React does that for us!

Contents

[Setting Up the Environment 2](#_Toc57986302)

[Creating the Project 3](#_Toc57986303)

[Understanding the Project Layout 4](#_Toc57986304)

[src (source) Folder 5](#_Toc57986305)

[Creating Our Application 7](#_Toc57986306)

[Creating Our Main Component 7](#_Toc57986307)

[Installing Bootstrap 9](#_Toc57986308)

[Creating the Title Navigation Component 11](#_Toc57986309)

[Creating BookViewer Component 14](#_Toc57986310)

[Modular Sections 18](#_Toc57986311)

[File Pathing in React 18](#_Toc57986312)

# Setting Up the Environment

First, we need to install a package that will help us create a starter template for our React applications. Inside of your terminal or command prompt, enter the following npm command:



NOTE: if you are on a Mac you may need to append the word “sudo“ to the beginning of this command for administrative privileges.

NOTE: The following extensions are optional and not required.

Once this is installed, we can move onto to setting up a Visual Studio Code with a few helpful extensions. Open Visual Studio Code and click on the extension’s icon on the far-left side of the screen:

Graphical user interface, text, application

Description automatically generated

Once the side panel opens search for and install each of the extensions below:

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Prettier – Code Formatter –

Prettier is used for automatically formatting the code within your JavaScript files on every save. It can help keep your code easy to read and contain consistent line breaks, tab spaces, and line lengths.

Simple React Snippets –

Simple React Snippets provides a series of shortcuts for creating components and UI logic. You will find notes within this tutorial specifying what shortcut to use and how!

# Creating the Project

First thing we need to do is create our project. Go to a folder on your computer where you would like to store your React applications. In this folder, open a gitbash or a terminal and enter the following command:

NOTE: “dcc-library” in the command below is what the name of the React project will be. Feel free to change “dcc” with your name!



Graphical user interface, text, application

Description automatically generated

As soon as you hit enter a new folder will appear that has the same name as the project name you specified in the above command. This is where your new React application project template is being created.

Once it completes, you will see the “Happy hacking!” message. You will then want to open the new folder that was just created in Visual Studio Code.

# Understanding the Project Layout

Once you open your project in Visual Studio Code, you will see the following folder and file structure in your explorer:

Text

Description automatically generated

**node\_modules -**

Node modules is a folder that stores all the projects third party package dependencies. As we can see here, when using the “create-react-app” command, the template comes with and is dependent on some third-party packages.

**public –**

The public folder contains the base html file that the entire React application will be “injected” into. If you open that folder you will see an index.html file. Within that index.html file is a tag with the id of root. Basically, when the React application compiles down to html, it will be inserted into this div tag that has that id of “root”. You can see a snippet of that html file below:

<body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

    <!--

      This HTML file is a template.

**src –**

The src folder contains the main inner working files of our React application. We will be covering this folder in more detail within the next section.

**.gitignore –**

Since we used the “create-react-app” command, we cloned down a GitHub repository that already contained most of the code we see here! We will be using this .gitignore for our own GitHub repository once we get our source control set up!

**package-lock.json –**

**package.json –**

As is with any node application, the package.json file contains information about our node application such as name, versioning, dependencies, and scripts.

# src (source) Folder

If you open and expand the src folder you will see the following files:

A picture containing text, electronics, screenshot

Description automatically generated

Most of this is automatically generated code for the template project. Since we are going to building our own project from scratch, lets delete everything within this folder.

At this point your src folder should be completely empty. Next, create a new file called “index.js” within the src folder and add the following code:

import React from 'react';

import ReactDOM from 'react-dom';

const jsxElement = <h1>Our React App</h1>;

console.log(jsxElement);

ReactDOM.render(jsxElement, document.getElementById('root'));

Here we are importing two different things, creating a JSX element, and rendering our element to the div tag with the id of “root” on the index.html file within the public folder.

import React from 'react';

“Import React from ‘react’;” is importing the React library into this index.js file. Remember, because we created our React application with the create-react-app command, the React library is automatically installed!

You will notice we never actually reference this “React” variable anywhere within the file. Even though we do not explicitly use this variable it is being used.

Notice how we are creating a variable called “jsxElement” and setting it equal to some html.

const jsxElement = <h1>Our React App</h1>;

console.log(jsxElement);

When we run this in the browser, what do you think will log out to the console?

Intuitively you would think it would print “<h1>Our React App</h1>” but that is not actually what it will print.

Let’s start our React application and take a look at the console. To start our React application, open up a terminal within Visual Studio Code and run the command “npm start”:

A picture containing text, screenshot, black

Description automatically generated

Once the application has successfully started and you have opened your console, you should now see this:

Graphical user interface, text, application

Description automatically generated

Notice that what it logged to the console is an object and its typeof property is “react.element”!

That is what this React variable:

import React from 'react';

will do for us when we write HTML on our JavaScript files. It actually compiles our HTML (that is written on JavaScript files) into React Elements. This is called JSX. Every component you write will contain some JSX. JSX is what describes the structure and composition of our user interface.

import ReactDOM from 'react-dom';

This import allows us to access the React.DOM.render() function.

ReactDOM.render(jsxElement, document.getElementById('root'));

This function takes in two arguments. The first argument is the JSX element/component we are trying to render in the root div. The second argument is a reference to the document which in this case, is the index.html file that is in the public folder. It finds the “div” tag that has the id of “root” and injects our JSX element/component within it.

In our last example, we have created a variable and set it equal to some JSX (HTML). In a real React application we need to create components. Components are small pieces of UI separated and contained in their own class of function. Please refer to the React lecture slide show for the main difference between functional and class components before proceeding forward.

# Creating Our Application

## Creating Our Main Component

In every React application there is a root component. This root component is the highest component in the component tree. Every other component is in someway a child of this root component. We will see what I mean by this the further we proceed into the tutorial.

For now, lets create a new folder in our src folder. Name this new folder “components”. By convention, all components we make will go in this folder.

Once you have created that folder, add a new file to that folder called “app.jsx”:

Graphical user interface, text

Description automatically generated

NOTE: when creating new files in our React applications, follow a camelCasing naming convention. “app.jsx”, “appComponent.jsx” “appCompPage.jsx”

This is going to be our main component or our “root” component. Add the following code to the app.js file:

import React, { Component } from 'react';

class App extends Component {

    state = {  }

    render() {

        return (

            <h1>Our React App Using a Component</h1>

        );

    }

}

export default App;

Great work! We have just created our first component. This is a class component or also known as a “stateful” component. Currently, we are not using state within this component, but we will be shortly. For now, we want to tell the ReactDOM.render() function in the index.js file to use this component as the root of our application.

Go to the index.js file and change the following code:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './components/app';

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

First, we are importing our component into this index.js file. Notice the file path, we use “./” to access things inside the current directory and the drill down into where the “app.jsx” file exists. It is important to be aware of file pathing and how it works. Please see the [File Pathing in React](#_File_Pathing_in) section for further discussion.

If your application was still running from earlier in the tutorial, then you can simply save the file and see the changes in your browser! If your application is not running, then open a terminal and run “npm start”.

NOTE: if you are still not seeing the changes try refreshing the browser or checking for errors in your terminal.

As you can see, we are no longer using that “jsxElement” variable as our application. Instead, we are using the App component as our main entry point for our React application.

If we were to visualize our current component tree, we would see:



That is it! It is the only component within our application! It is highlighted in blue to display that it is a stateful component (even though we are not using state yet). Let’s keep working to change that.

## Installing Bootstrap

We are going to utilize bootstrap within our React application. To install bootstrap, we need to install the third-party package “bootstrap”.

In order to do this, we will need to shut down our application if it is currently running. If it is running, click in the terminal and hit “ctrl c” on Windows or “cmd c” on Mac.

NOTE: you may be prompted: “Terminate batch job (Y/N)?” Just enter “Y” and hit enter or hit “ctrl c” or “cmd c” again and it will shut down.

You can read more about the bootstrap package and what it includes here: <https://www.npmjs.com/package/bootstrap>

Within your terminal run the command:



Finally, to make bootstrap globally available in everyone of our components, open up the index.js file and add the following line of code:

import React from 'react';

import ReactDOM from 'react-dom';

import App from './components/app';

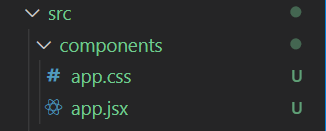
import 'bootstrap/dist/css/bootstrap.css';

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

Awesome! We can now use any bootstrap styles within our JSX (HTML/JavaScript) code!

## Creating The App Components CSS File

Now that we have bootstrap installed, we can start to utilize some more stylization into our application. Create a new file directly in the components folder called “app.css”.



Add the following styles to the app.css file:

.col-align {

  text-align: center;

}

.center {

  margin: 0;

  position: absolute;

  top: 50%;

  left: 50%;

  -ms-transform: translate(-50%, -50%);

  transform: translate(-50%, -50%);

}

.row-spacer {

  margin-bottom: 2em;

}

These styles will now be globally available to any child component of the App component and since the App component is the root component of the application, these styles will be available everywhere!

We will be using these periodically through out the tutorial.

## Creating the Title Navigation Component

Lets create a new component to act as our “title bar” or “navigation bar”. Create a new folder in the components folder called “TitleBar”.

Within this folder create two new files. One called “titleBar.js” and another called “titleBar.css”:



The titleBar.jsx will be our title bar/navigation component. The titleBar.css is where we will contain all styles specific to the TitleBar component’s JSX code.

Inside of the titleBar.jsx, add the following code:

import React from 'react';

function TitleBar(props) {

    return (

        <div className="row row-spacer">

            <div className="col-md-12" style={{padding: 0}}>

                <div>

                    <h1>dCC Library</h1>

                </div>

            </div>

        </div>

     );

}

export default TitleBar;

Here we have created a functional component. This is a component that is only used to present information and is also called a presentational component. We use functional components when we do not have a need for state within the component.

This functional component returns a series of JSX code. As you can see, we are using bootstraps grid system layout (with the use of the row and col-\*-\* classes) to organize the structure of our component. Please refer to this link for a review on the bootstrap grid layout: <https://www.w3schools.com/bootstrap/bootstrap_grid_system.asp>

Notice how for each HTML tag that has a class attached to it, we are using “className=…” rather than “class=…”. This is because the “class” keyword is reserved in JavaScript files and so we must use “className” when trying to link a CSS class as an attribute of an HTML tag. Also notice how we have used one of the styles located in the app.css file “row-spacer”. This gives a little space between each row in our application.

In here we have an “<h1> tag that represents the title of our application. Feel free to change “dcc” to your name!

Now that we have the skeleton of this component completed, we now need to introduce this component within our component tree so it can be rendered on the screen as part of our application. Remember how we said that the App component was going to be our “root” component. We need to go to the app.js file and include this component within the App components JSX code.

Go to the app.js file and make the following changes:

import React, { Component } from 'react';

import TitleBar from './TitleBar/titleBar';

class App extends Component {

    state = {  }

    render() {

        return (

            <div className="container-fluid">

                <TitleBar />

            </div>

        );

    }

}

export default App;

Here we are importing the TitleBar component into the app.js file. Since we have just imported the TitleBar component, we can now use it as if it were its own HTML tag within the JSX code:

                <TitleBar />

We create a TitleBar component within a “<div className=”container”>” tag. Because our TitleBar container uses the bootstrap “row” class, the TitleBar component must be inside a div with the “container” class. This is per bootstraps requirement. Again, for review on bootstraps grid system, see the link above.

At this point, if you save your both the titleBar.jsx and app.jsx files, you should see your title appear in your browser!

Graphical user interface, application

Description automatically generated

This is a very simple title bar for our application! If we were to now take a look at the component tree we would see this:

Diagram

Description automatically generated

TitleBar is now a child of the App component because the TitleBar was created in (nested in) the App components JSX!

Let’s add some more style to this to make it look a little better. We are now going to use the titleBar.css file to add some custom CSS styles.

Add the following code to the titleBar.css file:

.titlebar-nav {

  background-color: lightblue;

  width: 100%;

}

h1 {

  padding-bottom: 0.2em;

  padding-left: 1em;

}

Now, lets add these styles to some of the HTML tags in our titleBar.js file’s JSX code.

NOTE: we are statically importing the titleBar.css file into the titleBar.js file. We must import the CSS file in order to use the custom styles we have defined on it.

Add the following code to the titleBar.js file:

import React from 'react';

import './titleBar.css';

function TitleBar(props) {

    return (

        <div className="row row-spacer">

            <div className="col-md-12" style={{padding: 0}}>

                <div className="titlebar-nav">

                    <h1>dCC Library</h1>

                </div>

            </div>

        </div>

     );

}

export default TitleBar;

Once you save, you should now see a blue background behind our title. Feel free to change the color theme inside the CSS file to what you prefer!

## Creating BookViewer Component

We are going to create a component that allows us to look at books in our library’s collections. Before proceeding, look at the example of what we are about to build here:

Text

Description automatically generated

As you can see, we are going to have a couple buttons, a book, and some stateful interaction within this next component.

First, we are going to code this component directly in the App component. We are doing this to because the App component has the state we need to use on order to achieve this interactive viewer. Once we are done with the base implementation, we will split the book viewer logic into its own component and pass the state from the App component down to our BookView component.

Inside of the app.js file, we need to create a constructor for the App class so we can initialize some of our book data as member (class) variables. Add the following code to the app.js file:

import React, { Component } from 'react';

import TitleBar from './TitleBar/titleBar';

class App extends Component {

    constructor(props) {

        super(props);

        this.books = [

            {title: 'Ready Player One', author: 'Ernest Cline'},

            {title: 'All the Light We Cannot See', author: 'Anthony Doerr'},

            {title: 'The First and Last Freedom', author: 'Jiddu Krishnamurti'},

        ];

        this.state = {

          bookNumber: 0

         }

      }

    render() {

        return (

...

Here we have a created the standard class constructor for a class component. We are creating a variable called “books” and setting it equal to an array of objects.

Notice also that we have moved our state variable initialization to within the constructor and added a stateful variable “bookNumber”. This “bookNumber” variable is going to represent the index number of the book in the books array that we are currently viewing.

Remember, because we are working in class components (JavaScript classes) we must use the “this” keyword to reference functions and variables on this class.

Now, add the following changes to the render method in the App component:

…

render() {

        return (

            <div className="container-fluid">

                <TitleBar />

                <div className="row row-spacer">

                    <div className="col">

                        {/\*Button here to move to the previous book viewed\*/}

                    </div>

                    <div className="col">

                        {/\*Display Book with Cover here\*/}

                    </div>

                    <div className="col">

                        {/\*Button here to move to the previous book viewed\*/}

                    </div>

                </div>

            </div>

        );

    }

…

Here we have added another row (the first row is within the TitleBar component) and divided that row into three (3) columns. The side columns are going to be used for our buttons and the middle column will be used for visualizing the current book.

Let’s show the first books title and author in the middle column. Add the following code to the middle column:

                    <div className="col-md-4">

                        {/\*Button here to move to the previous book viewed\*/}

                    </div>

                    <div className="col-md-4">

                        <h1>{this.books[this.state.bookNumber].title}</h1>

                        <h4>{this.books[0].author}</h4>

                    </div>

                    <div className="col-md-4">

                        {/\*Button here to move to the previous book viewed\*/}

                    </div>

We have created to header tags and placed some content within them. Notice how we are using the (“{js content here}”) curly brackets to house our JavaScript code. When we want to mix JavaScript variables or function calls in our JSX, we put that JavaScript inside these “{}” curly brackets.

Within both header tags we are referencing our “books” variable we declared and initialized in the constructor with “this.books”.

In the <h1> tag we are accessing an index of the books array using the stateful variable “bookNumber” with “this.state.bookNumber”.

In the <h4> tag we are accessing the index of the books array using the hard-coded value of zero (0).

Both approaches, in terms of the application starting for the first time, will produce the same result. That is because we initialize the stateful variable “bookNumber” to the value of zero (0) in the constructor of the class.

If we were to change the value of the “bookNumber” variable, the <h1> tag would change what book title it is displaying. However, the <h4> tag would continue to show the same book author (what ever book is at index 0).

For that reason, we would not want to hard code the index number in either of the header tags. Change the <h4> to use the stateful variable “bookNumber” rather than the hard-coded number of zero (0):

                    <div className="col-md-4">

                        <h1>{this.books[this.state.bookNumber].title}</h1>

                        <h4>{this.books[this.state.bookNumber].author}</h4>

                    </div>

When you save this file and take a look in the browser you should now see:

Graphical user interface, text

Description automatically generated

# Modular Sections

## File Pathing in React