

Section 3: React Basics and Working With Components

Brian E. Nguyen

October 15, 2021

Contents

1	Module Introduction	1
2	What Are Components? And Why is React All About them?	2
3	React Code is Written in a “Declarative Way”	3
3.1	How exactly is a component built?	3
3.2	The Declarative Approach	3
4	Creating a New Project	3
4.1	Preface	3
4.2	Creating the Project / Starting the Dev Server	4
4.3	The Application	4
5	Analyzing a Standard React Project	5
5.1	<code>index.js</code>	5
5.1.1	General Overview	5
5.1.2	The React DOM	6
5.1.3	The <code>render</code> Method	7

1 Module Introduction

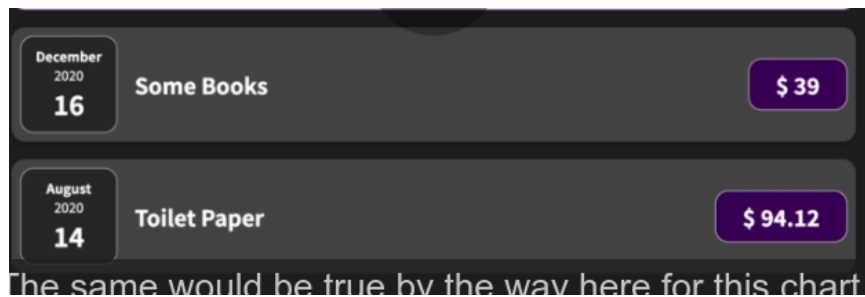
We will learn how to use everything that makes up React

- React Core Syntax & JSX
- Work with components

- Work with data

2 What Are Components? And Why is React All About them?

- React is a JavaScript library for building user interfaces
- HTML/CSS/JS are about building user interfaces as well
- We use libraries like React to simplify building user interfaces
 - This is through the use of **components**
- All user interfaces in the end are made up of components



- in this picture, these components are the same; they are just reused twice
- **components** are reusable building blocks in your user interface
 - though you don't have to reuse components
- components are made up of HTML for text, CSS for styling, and possibly JS for logic
- React embraces the concept of components because of
 1. Resusablility - not repeating yourself
 2. Separation of concerns - keeping code base small and manageable. Don't do too many things in one and the same place

3 React Code is Written in a “Declarative Way”

3.1 How exactly is a component built?

- in the end, components are built with HTML, CSS, and JS, then we combine the components to build a user interface
- mostly, they’re about HTML and JS; CSS could be a factor but it’s not too important

3.2 The Declarative Approach

- React allows you to create **reusable and reactive components** consisting of **HTML and JavaScript** (and CSS)
- React is built using the **declarative approach**, which means that you should not tell React that a certain HTML element should be created and inserted in a specific place on the UI
 - instead, you will always define the desired target state(s) and let React figure out the actual JavaScript DOM instructions
- in the end, we are essentially building our own custom HTML elements

4 Creating a New Project

4.1 Preface

- the easiest way to create a React app is through this GitHub Repo
 - <https://github.com/facebook/create-react-app>
- this has preconfigured folders with basic React code files
- the **create-react-app** tool creates a development environment for our app
- You can also visit this site to view the documentation:
 - <https://reactjs.org/>
- Make sure you have Node JS installed on your machine

4.2 Creating the Project / Starting the Dev Server

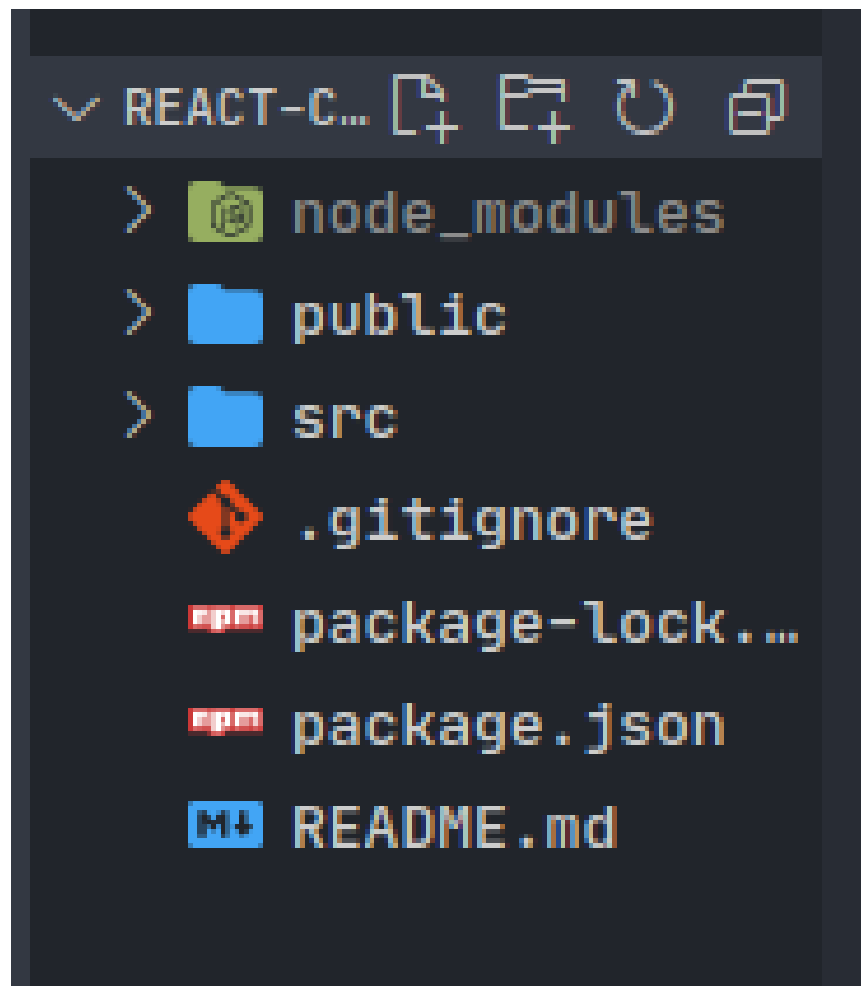
Run these commands in your terminal:

```
npx create-react-app react-complete-guide  
cd my-app  
npm start
```

After the project is created, `cd` into the project and run `npm start`. The application will automatically load up a preview of our app on `localhost:3000`

4.3 The Application

You should see something like this in your application:



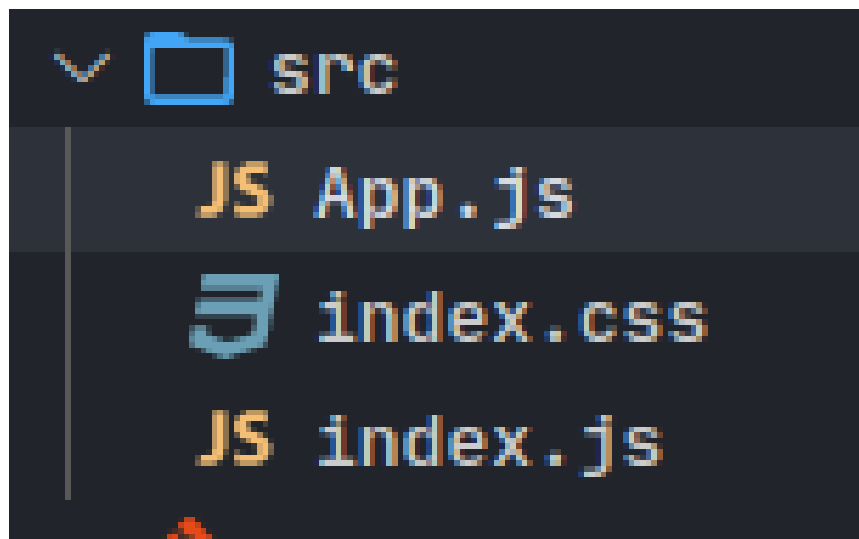
These files might change overtime, but just know that you should have a `src` folder and a `package.json` file. You should also delete all files in the `src` folder except for the following:

- `App.js`
- `index.css`
- `index.js`

Inside each of these files, replace their code with the ones on this repo. The reason is that this version is the original one that is created with `create-react-app`. Just to be sure that the packages are updated, run `npm i` in your terminal.

5 Analyzing a Standard React Project

Let's have a look at the `src` folder because that is where we will spend the majority of our time



- the most important takeaway: *React code is just JavaScript code*

5.1 `index.js`

5.1.1 General Overview

- let's start with the `index.js` file

```
import ReactDOM from "react-dom";

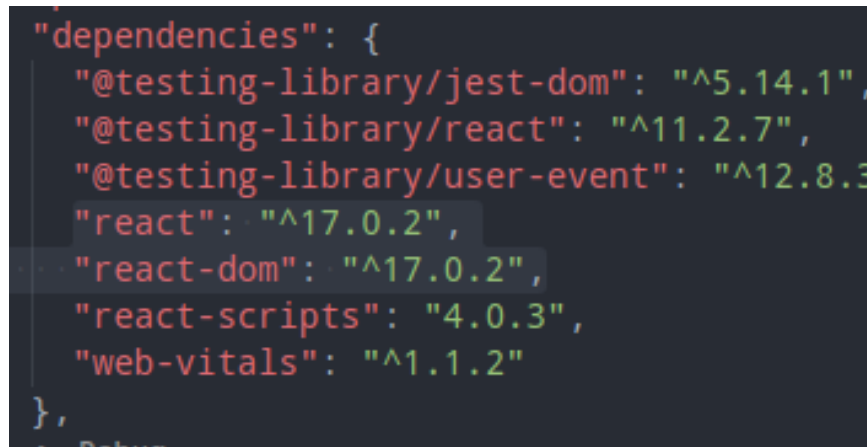
import "./index.css";
import App from "./App";

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

- this file is the first one that is executed whenever the page is loaded
- we will write code that's easy to read and has syntactic sugar, but this kind would actually run in the browser
- the `npm start` command will take our code and transform it before it is delivered to the browser
 - an example is `import "./index.css"`, because that's not actual JavaScript. You can't import CSS into JS
- another example of invalid JS is `<App />`

5.1.2 The React DOM

- in this file, we are importing *ReactDOM* from `react-dom`
- also, in the `package.json` file, you would see two dependencies: `react` and `react-dom`



```
"dependencies": {
  "@testing-library/jest-dom": "^5.14.1",
  "@testing-library/react": "^11.2.7",
  "@testing-library/user-event": "^12.8.3",
  "react": "^17.0.2",
  "react-dom": "^17.0.2",
  "react-scripts": "4.0.3",
  "web-vitals": "^1.1.2"
},
```

- while technically these are two separate packages, these encompass the React library

5.1.3 The render Method

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

- the `render` method takes two arguments

1. The App Component

The `App` component is the main component that encompasses all other components

```
function App() {  
  return (  
    <div>  
      <h2>Let's get started!</h2>  
    </div>  
  );  
}  
  
export default App;
```

In this, we are returning HTML code inside of a JavaScript file. This is a feature called **JSX**

2. The root Element DOM API

the default JavaScript DOM API. It calls the `root` ID inside of the `public/index.html` file, which is the actual file that's used to display our React code

This is what it looks like in the `index.html` file

```
<body>  
  <noscript>You need to enable JavaScript to run this app.</noscript>  
  <div id="root"></div>  
  ...
```