Section 3: React Basics and Working With Components

Brian E. Nguyen

October 16, 2021

Contents

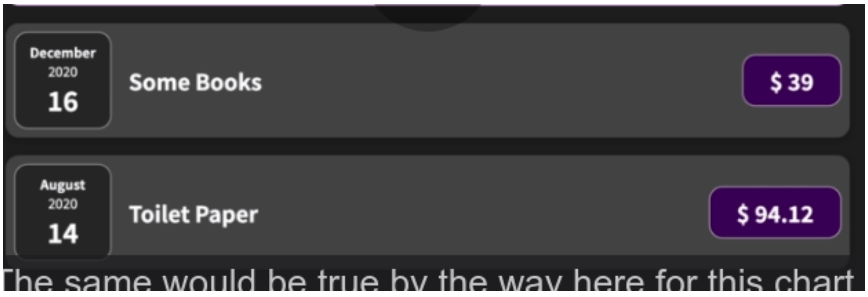
1 Module Introduction

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

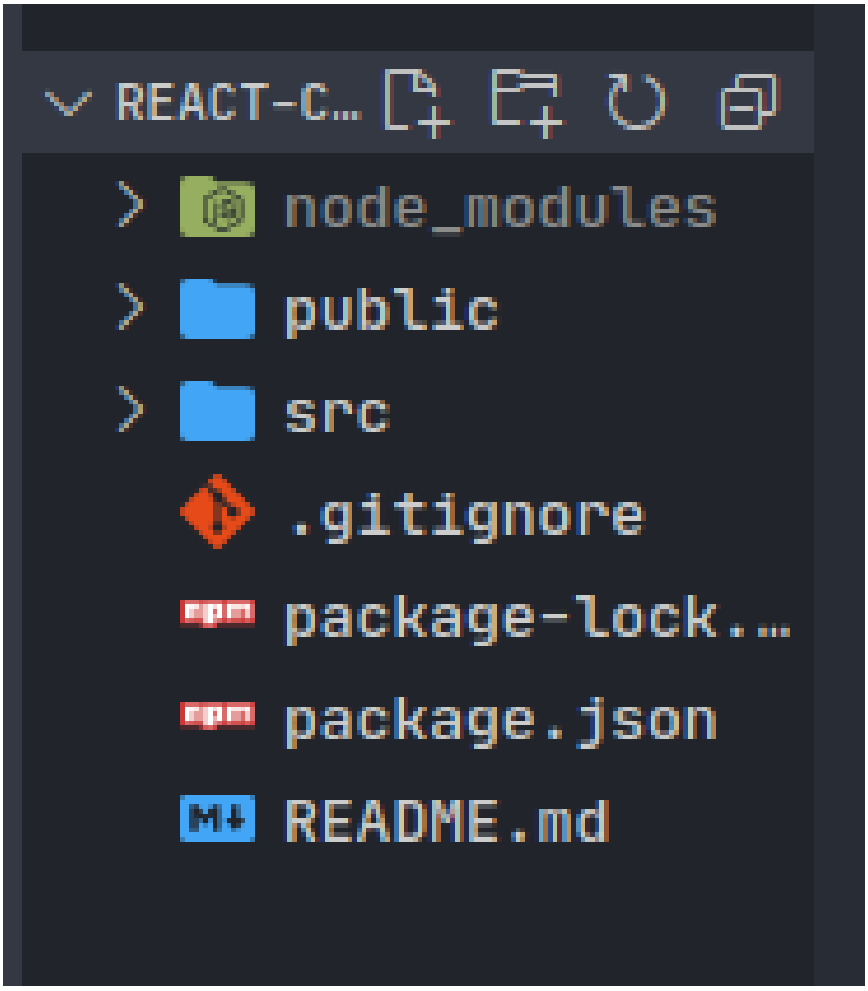
Run these commands in your terminal:

```
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:



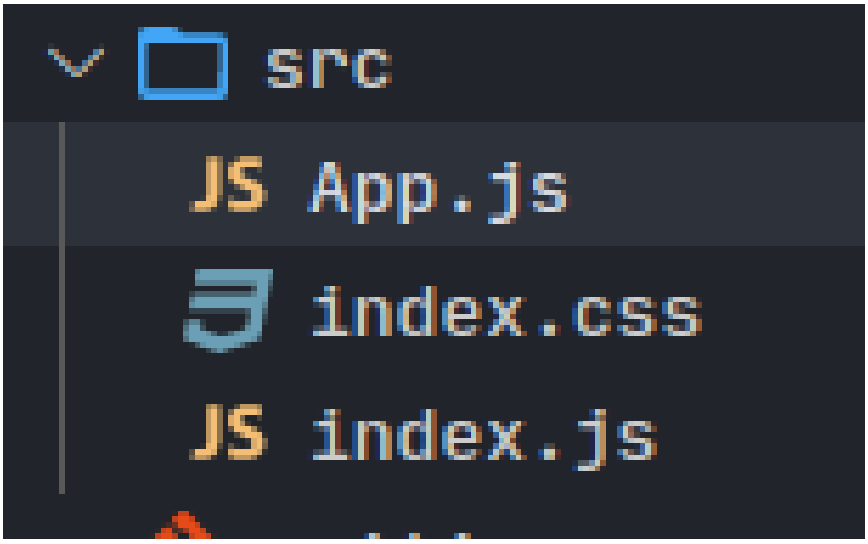
These files might change overtime, but just know that you should have a `src` folder and a `package.json` file. You should also delete all files in the `src` folder except for the following:

- 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"));
```

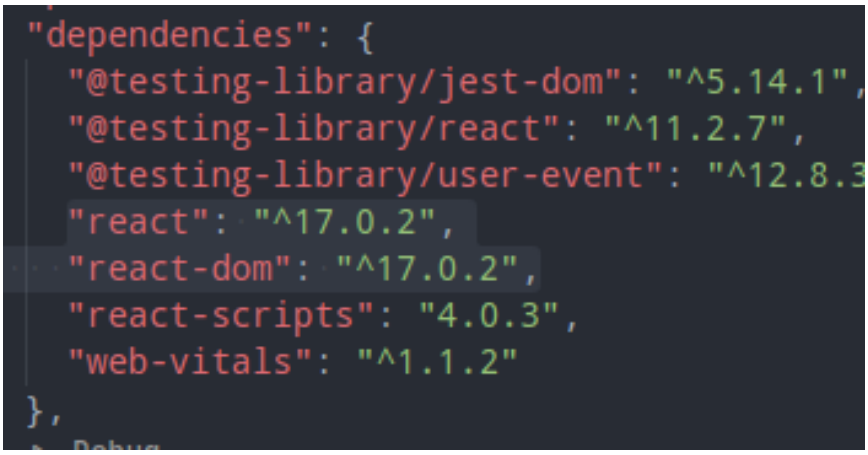
- this file is the first one that is executed whenever the page is loaded
- we will write code that's easy to read and has syntactic sugar, but this kind would actually run in the browser
- the `npm start` command will take our code and transform it before it is delivered to the browser

  - 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`



- while technically these are two separate packages, these incompass 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

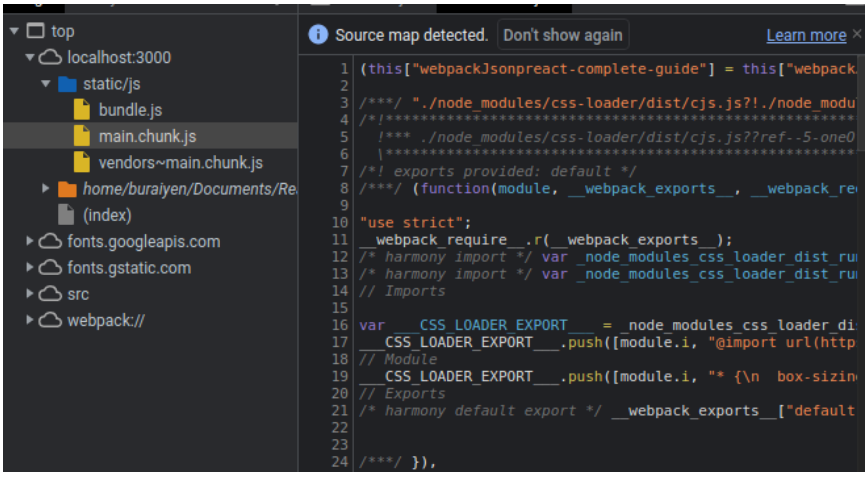
   the default JavaScript DOM API. It calls the `root` ID inside of the *public/index.html* file, which is the actual file that's is used to display our React code

   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>
  ...
```

6 Introducing JSX

- JSX stands for *JavaScript XML*
- the `npm start` command transforms our React code into browser-friendly code. This is what it looks like in the browser:



- inside of the `main.chunk.js` file, we see a function called `App`

```
function App() {
  return /*#__PURE__*/Object(react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_0__["j
    children: /*#__PURE__*/Object(react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_0
      children: "Let's get started!"
    }, void 0, false, {
      fileName: _jsxFileName,
      lineNumber: 4,
      columnNumber: 7
    }, this)
  }, void 0, false, {
    fileName: _jsxFileName,
    lineNumber: 3,
    columnNumber: 5
  }, this);
}
```

- this is the transformed code that runs in the browser

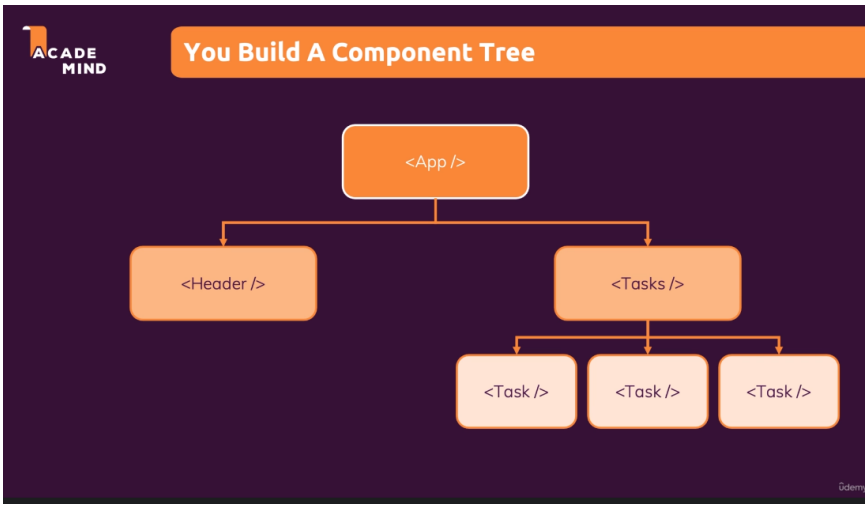
7 How React Works

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

export default App;
```

- we have HTML code inside of `App()`
- we can build our custom HTML elements with React
- if we make changes to our JSX code while the development server is running, then those changes will automatically update in the browser

8 Building a First Custom Component

- it is best practice to put components into their own files, so you have one file per component
- React projects have dozens and hundreds of components in the end, *and that's completely normal*
- inside of the *src* folder, let's create a *components* folder to hold our components source files
  - we don't put `App.js` inside of it because it's a special type of component
- in the end, we are building a component tree

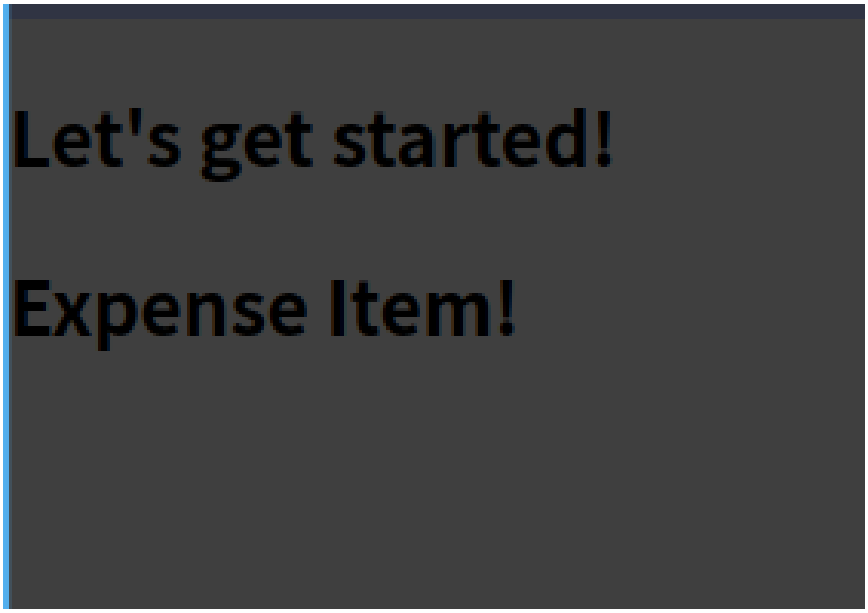


1. let's add a new file inside of *components* folder called `ExpenseItem.js`
   - it's a React convention to name files starting with a capital character, and every sub-word starts with a captial letter
   - keep in mind that a React component is just a JavaScript function
2. lets's write our `ExpenseItem()` function in our `ExpenseItem.js`

```
const ExpenseItem = () => {
  return <h2>Expense Item!</h2>;
};

export default ExpenseItem;
```

1. after we are done with our function, we need to write an export statement
2. we will then import that component in `App.js` and use it

```
import ExpenseItem from "./components/ExpenseItem";

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



9 Writing More Complex JSX Code

- let's tweak the HTML code in the `ExpenseItem()` function
- what if we want to add a title, amount, and the date?

- we can only have 1 root element inside of our function
- if we need multiple HTML elements in our function, then we need to surround them inside parentheses

```
const ExpenseItem = () => {
return (
<div>
<div>Date</div>
<div>
<h2>Title</h2>
<div>Amount</div>
</div>
</div>
);
};

export default ExpenseItem;
```

10 Adding Basic Styling

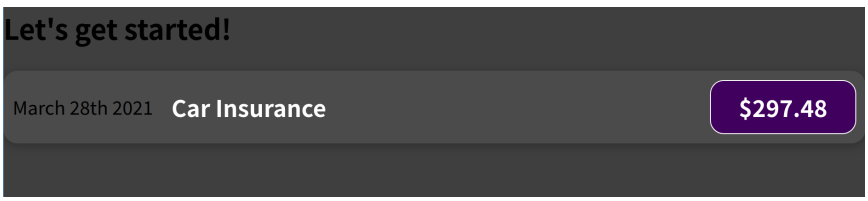
- we still use CSS for styling, but there's nothing "React-specific" about it
- to add CSS to a component, we would first need to create a CSS file that has the same name as the component
  - in this case, it would be `ExpenseItem.css`
- the code is provided at this link
- back in `ExpenseItem.js`, we will import the CSS

```
// ExpenseItem.js
import "./ExpenseItem.css";

const ExpenseItem = () => {
  return (
      <div>
        <div>Date</div>
      <div>
```

```
            <h2>Title</h2>
            <div>Amount</div>
          </div>
        </div>
    );
    };

    export default ExpenseItem;
```

- when we work with adding classes to elements, *we don't use* `class`, *but rather* `className`
- it's strange, but keep in mind that the code inside of our function is not really HTML
  - this is JSX syntax invented by the React team
  - under the hood, it's still JavaScript code
- let's add the classes to the JSX code

```
import "./ExpenseItem.css";

const ExpenseItem = () => {
  return (
    <div className='expense-item'>
      <div>March 28th 2021</div>
      <div className='expense-item__description'>
        <h2>Car Insurance</h2>
        <div className='expense-item__price'>$297.48</div>
      </div>
    </div>
  );
};

export default ExpenseItem;
```



11 Outputting Dynamic Data & Working with Expressions in JSX

- we won't have just one expense, but many, and where the data is not hardcoded
- let's create some temporary variables to output data inside of our `ExpenseItem.js`

```
// ExpenseItem.js
import "./ExpenseItem.css";

const ExpenseItem = () => {
  // Note: months start at 0
  const expenseDate = new Date(2021, 2, 28);
  const expenseTitle = "Car Insurance";
  const expenseAmount = 297.48;
  return (
    <div className='expense-item'>
      <div>March 28th 2021</div>
      <div className='expense-item__description'>
        <h2>Car Insurance</h2>
        <div className='expense-item__price'>$297.48</div>
      </div>
    </div>
  );
};

export default ExpenseItem;
```

- you can replace hardcoded data with open and closing curly braces
- inside of the curly braces you can run basic JavaScript expressions, like `1+1`
- let's insert our variables into the curly brackets

```
import "./ExpenseItem.css";

const ExpenseItem = () => {
  // Note: months start at 0
```

```
  // Date object is not a string
  const expenseDate = new Date(2021, 2, 28);
  const expenseTitle = "Car Insurance";
  const expenseAmount = 297.48;
  return (
    <div className='expense-item'>
      <div>{expenseDate.toISOString()}</div>
      <div className='expense-item__description'>
        <h2>{expenseTitle}</h2>
        <div className='expense-item__price'>${expenseAmount}</div>
      </div>
    </div>
  );
};

export default ExpenseItem;
```

12 Passing Data via "Props"

- how can we make the `ExpenseItem` component reusable?
  - in `App.js` we can just copy the component multiple times, but they all output the same data
- let's create data outside of our `ExpenseItem` component
  - insert this data inside of your `App.js` component