Creating a React App

Use create-react-app to bootstrap a React application on your own computer

Creating a React App

Introduction

React is a user interface framework developed by Facebook. It has a quickly growing developer adoption rate and was ranked as the most loved web framework in the 2019 Stack Overflow developer survey. This article will walk you through setting up your first React app and assumes you are familiar with text editors and command line navigation.

Getting Ready

We will be using the *Node package manager (npm)*, so you will need to have *Node* installed on your computer. To check if you have Node installed, run this command in your terminal:

```
node -v
```

If you have Node installed, this command will return a version number, like `v12.18.1`.

If it's not already installed, follow the steps in Setting Up Node Locally before moving on.

When you install Node, you automatically get npm installed on your computer as well. However, npm is a separate project from Node.js, and tends to update more frequently. As a result, even if you've just installed Node (and therefore npm), it's a good idea to update your npm. Luckily, npm knows how to update itself!

To upgrade to the latest version of npm on *nix (OSX, Linux, etc.), you can run this command in your terminal:

```
sudo npm install -g npm@latest
```

To upgrade on Windows, follow the steps found in the npm documentation.

1. Setting Up the Boilerplate Application

It is possible to manually create a React app, but Facebook has created a Node package *create-react-app* to generate a boilerplate version of a React application.

Besides providing something that works out-of-the-box, this has the added benefit of providing a consistent structure for React apps that you will recognize as you move between React projects. It also provides an out-of-the-box build script and development server.

We will use *npx*, a package runner tool that comes with npm 5.2+ and higher, to install and run `create-react-app`. This will ensure that the latest version of create-react-app is used.

Open up your terminal.

- If you've previously installed `create-react-app` globally via `npm install -g create-react-app`, it is recommended that you uninstall the package first. In your terminal run these commands:

```
npm uninstall -g create-react-app
npx create-react-app myfirstreactapp
```

- If you've never installed `create-react-app` before, you can simply run this command:

```
npx create-react-app myfirstreactapp
```

- If you have *Yarn* installed, `create-react-app` will use it by default to create new projects. If you would prefer to use npm, you can append `--use-npm` to the creation command. It will look like this:

```
npx create-react-app myfirstreactapp --use-npm
```

(Feel free to replace `myfirstreactapp` with whatever name you want, as long as it doesn't contain capital letters :-))

Upon completion, you will get some quick tips on how to use the application:

```
  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 myfirstreactapp
  npm start

Happy hacking!
```

Before we run the app, let's take a look around the app structure and see what it contains.

2. React App Structure

Change directories into the app you just created, and open the app in the text editor of your choice. You should see the following file structure:

```
myfirstreactapp
├── node_modules
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
├── src
│   ├── App.css
│   ├── App.js
│   ├── App.test.js
│   ├── index.css
│   ├── index.js
│   ├── logo.svg
│   ├── serviceWorker.js
│   └── setupTests.js
├── .gititgnore
├── package.json
├── package-lock.json
└── README.md
```

create-react-app has taken care of setting up the main structure of the application as well as a couple of developer settings. Most of what you see will not be visible to the visitor of your web app. React uses a tool

called *webpack* which transforms the directories and files here into static assets. Visitors to your site are served those static assets.

Don't worry if you don't understand too much about webpack for now. One of the benefits of using create-react-app to set up our React application is that we're able to bypass any sort of manual configuration for webpack. If you're interested in delving deeper into it on your own, you can find a high-level overview of webpack's core concepts here.

.gitignore

This is the standard file used by the source control tool git to determine which files and directories to ignore when committing code. While this file exists, create-react-app did not create a git repo within this folder. If you take a look at the file, it has taken care of ignoring a number of items (even **.DS_Store** for Mac users):

```
✔ ~/Documents/Projects/myfirstreactapp
[23:20 $ cat .gitignore
# See https://help.github.com/ignore-files/ for more about ignoring files.

# dependencies
/node_modules

# testing
/coverage

# production
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

package.json

This file outlines all the settings for the React app.

- `name` is the name of your app
- `version` is the current version
- `"private": true` is a failsafe setting to avoid accidentally publishing your app as a public package within the npm ecosystem.
- `dependencies` contains all the required Node modules and versions required for the application. In the picture above, you'll see six dependencies. The first three, as you may have guessed, are for the purpose of testing. The next two dependencies allow us to use `react` and `react-dom` in our JavaScript. Finally, `react-scripts` provides a useful set of development scripts for working with React. In the

screenshot above, the `react` version specified is `^16.13.1`. This means that npm will install the most recent major version matching 16.x.x. In contrast, you may also see something like `~1.2.3` in **package.json**, which will only install the most recent minor version matching 1.2.x.

- `scripts` specifies aliases that you can use to access some of the react-scripts commands in a more efficient manner. For example, running `npm test` in your command line will run `react-scripts test --env=jsdom` behind the scenes.
- You will also see two more attributes, `eslintConfig` and `browserslist`. Both of these are Node modules having their own set of values. `browserslist` provides information about browser compatibility of the app, while `eslintConfig` takes care of the code linting.

node_modules

This directory contains dependencies and sub-dependencies of packages used by the current React app, as specified by **package.json**. If you take a look, you may be surprised by how many there are.

Running `ls -1 | wc -l` within the **node_modules/** directory will yield more than 800 subfolders. This folder is automatically added to the **.gitignore** for good reason! Don't worry, even with all these dependencies, the basic app will only be around 50 KB after being minified and compressed for production.

package-lock.json

This file contains the exact dependency tree installed in **node_modules/**. This provides a way for teams working on private apps to ensure that they have the same version of dependencies and sub-dependencies. It also contains a history of changes to **package.json**, so you can quickly look back at dependency changes.

public

This directory contains assets that will be served directly without additional processing by webpack. **index.html** provides the entry point for the web app. You will also see a favicon (header icon) and a **manifest.json**.

The manifest file configures how your web app will behave if it is added to an Android user's home screen (Android users can "shortcut" web apps and load them directly from the Android UI). You can read more about it here.

src

This contains the JavaScript that will be processed by webpack and is the heart of the React app. Browsing this folder, you see the main App JavaScript component (**App.js**), its associated styles (**App.css**), and test suite (**App.test.js**). **index.js** and its styles (**index.css**) provide an entry into the App and also kick off the **registerServiceWorker.js**. This service worker takes care of caching and updating files for the end-user. It allows for offline capability and faster page loads after the initial visit. More of this methodology is available here.

As your React app grows, it is common to add a **components/** directory to organize components and component-related files and a **views/** directory to organize React views and view-related files.

3. Starting the React App Development Server

As was stated in the success message when you ran `create-react-app`, you just need to run `npm start` in your app directory to begin serving the development server. It should auto-open a tab in your browser that points to `http://localhost:3000/` (if not, manually visit that address). You will find yourself looking at a page resembling the following image:

As stated, any changes to the source code will live-update here. Let's see that in action.

Leave the current terminal tab running (it's busy serving the React app) and open **src/App.js** in your favorite text editor. You'll see what looks like a mashup of JavaScript and HTML. This is *JSX*, which is how React adds XML syntax to JavaScript. It provides an intuitive way to build React components

and is compiled to JavaScript at runtime. We'll delve deeper into this in other content, but for now, let's make a simple edit and see the update in the browser.

Change the main paragraph text to read `Hello Codecademy!` in **App.js** and save the file.

src > JS App.js > ...

```
import React from 'react';
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>
          Hello Codecademy!
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

If you left the terminal running, you should be able to switch over to your browser and see the update:

Congratulations! You're now up and running with React. You can now begin adding functionality for your application.

Next Steps

If you'd like to learn more about create-react-app, start with the documentation on the create-react-app website.

Since an important next step after creating a React App is to set up your environment to debug it, consider checking out our React Developer Tools article. There, we use the initial skeleton created with create-react-app to get you ready to begin debugging React Apps.