Open in app ↗

Medium Search Write

Webpack and Babel — What are they, and how to use them with React.

Aswin G · Follow
6 min read · Dec 11, 2019

360

Starting a new project is not something one does often, compared to how much time we spend on projects that we have already started. It is, perhaps this, that makes me frequently overlook the configuration that goes into turning a blank slate to a wire frame of what you want.

Starting a React project is probably one of the most zero-configuration experiences you could have, thanks to the excellent `create-react-app` tool. But while using it to start a new blank React project for the nth time a few days ago, I realized apart from having a foggy idea, I never delved into what *exactly* goes on under the hood. So I did, and now I can make new React projects, free of magical tools. This article is the result of what I've learned.

Webpack and Babel — Tools we can't code without

We'll be configuring both of these for our React project, so first here's a quick explanation what these tools are.

Webpack is a **bundler** that uses a dependency graph to bundle our code and assets (incuding static assets such as images) into a 'bundle' which we can then deploy.

Creating a huge monolithic JavaScript file is ugly and difficult to maintain, but multiple JavaScript files require multiple requests to fetch, and can add significant overhead. The solution is to write code splitting it into as many files as you need, and using `require()` (and as we'll soon see, `import`) statements to connect them together as we see fit. When we `require` something from a file, that becomes a dependency of that file. All our interconnected files and assets form a **dependency graph**. Then we use a bundler to to parse through these files and connect them appropriately based on the `require` and `import` statements and perform some more optimizations on top of it to generate one (sometimes more than one) JavaScript files which we can then use.

Webpack can also load non-js files, such as static assets, JSON files, CSS and more.

Babel, on the other hand, is a JavaScript compiler, sort of. It doesn't compile JavaScript the same way gcc compiles C++. Instead it compiles newer JavaScript into older JavaScript. Technically, it is a *transpiler.*

What this means you can write your JavaScript using the latest features the language has to offer and Babel will compile your code into JavaScript that will run without issues on most browsers, even if they don't support the cutting edge standards. Specifically in the case of React, your code will be in

JSX format, which of course is not a standard supported by browsers, and you'll need Babel to compile it down to regular JavaScript.

Configuring a new React app with Babel and Webpack

You'll need to have Node and NPM installed. These are the only requirements.

Create a new folder and run `npm init -y` in it to initialize a blank project. (npm asks you a few configuration questions if you don't use the `-y` option, you can do that if you don't want to change them in your package.json later.)

Now you'll have a `package.json` file which looks like this:

```json
{
  "name": "test-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

webpack-babel-react-1.json hosted with ❤ by GitHub view raw

Now let's install Webpack and Babel and add them as dependencies in `package.json`

```
npm i webpack webpack-cli webpack-dev-server --save-dev
```

This installs Webpack and the `webpack-cli`, which is a command line interface to use Webpack. `webpack-dev-server` is a simple server that we can use during development.

```
npm i @babel/core @babel/preset-env @babel/preset-react
```

This installs the core Babel package and two of Babel's presets. Babel works by using 'plugins' to it's core package to transform and compile the code you give it. `preset-env` is such a plugin that compiles down ES6+ (That is, versions of JavaScript that implement ECMAscript 2015 and newer standards) to ES5 standards-compatible JavaScript. If that didn't make sense, know that it just makes your JavaScript work with browsers that don't natively support these new features yet.

`preset-react` is similarly another plugin that allows Babel to recognize and compile React code.

Time to configure Webpack.

Create a new file called `webpack.config.js` and add the following:

```
1   const path = require('path')
2
3   module.exports = {
4       "entry": "./src/index.js",
5       "output": {
6           path: path.resolve(__dirname, 'dist'),
7           filename: "bundle.js"
8       },
9       module: {
10          rules: [
11              {
12                  test: /\.(js|jsx)$/,
13                  exclude: /node_modules/,
14                  use: "babel-loader"
15              },
16              {
17                  test: /\css$/,
18                  use: [
19                      {
20                          loader: "style-loader"
21                      },
22                      {
23                          loader: "css-loader"
24                      }
25                  ]
26              }
27          ]
28      }
29  }
```

webpack-babel-react-2.js hosted with ❤ by GitHub view raw

Let's go over what this does. Path is a handy module that comes with your Node installation that is used here to create an absolute path that is to be set as the output path for Webpack in line 6. `entry` is the file from which Webpack starts parsing your project. If you come from a background of C-like languages, think of it as the `main()` function for your code.

Under `module` , we specify some rules that should be used for different kind of files. Loaders are plugins for Webpack that are used for preprocessing files before they are bundled by webpack. For `.js` and `.jsx` files, we tell Webpack to use `babel-loader` which makes Webpack run these files through Babel before bundling them. For `.css` files we chain two loaders. `style-loader` allows us to import CSS into our JavaScript files, and `css-loader` allows us to use `@import` and `url()` in our CSS code. I won't be using CSS in this little sample project, but I would consider these loaders essential.

(Check out the full list of loaders here: https://webpack.js.org/loaders/. Webpack also has many more configuration options, including a no-configuration option that runs with all default configurations, even if there is no `webpack.config.js` file.)

In case you haven't guessed it already, we have to install these loaders before they can actually work.

```
npm i babel-loader style-loader css-loader --save-dev
```

Now let's create `.babelrc` , the configuration file for Babel. It's nice and simple, with it's purpose obvious — tell babel to use the plugins we installed earlier.

```
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]
}
```

webpack-babel-react-3.babelrc hosted with ❤ by GitHub view raw

Now it's time to install React and get to coding!

```
npm i react react-dom --save
```

Now let's create a `src` folder and inside that, write all our actual code. An `html` file is must, followed by the `index.js` which we specified as the entry point for Webpack and an `App.js` which will be the actual starting point for our app, since `index.js` just renders the contents of your app into the HTML element you specify.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

webpack-babel-react-4.html hosted with ❤ by GitHub view raw

```js
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

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

webpack-babel-react-5-index.js hosted with ❤ by GitHub view raw

```
import React from 'react';

class App extends React.Component {
    render() {
        return (
            <div>Welcome to React!</div>
        );
    }
}

export default App;
```

webpack-babel-react-6-App.js hosted with ❤ by GitHub view raw

Finally we have to install `html-webpack-plugin` and update our Webpack configuration to use it.

```
npm i html-webpack-plugin --save-dev
```

```
const path = require('path')
const HtmlWebpackPlugin = require('html-webpack-plugin')

module.exports = {
    "entry": "./src/index.js",
    "output": {
        path: path.resolve(__dirname, 'dist'),
        filename: "bundle.js"
    },
    module: {
        rules: [
            {
                test: /\.(js|jsx)$/,
                exclude: /node_modules/,
                use: "babel-loader"
            },
            {
                test: /\css$/,
                use: [
                    {
                        loader: "style-loader"
                    },
                    {
                        loader: "css-loader"
                    }
                ]
            }
        ]
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: "./src/index.html"
        })
    ]
}
```

webpack-babel-react-7-webpack.config.js hosted with ❤ by GitHub view raw

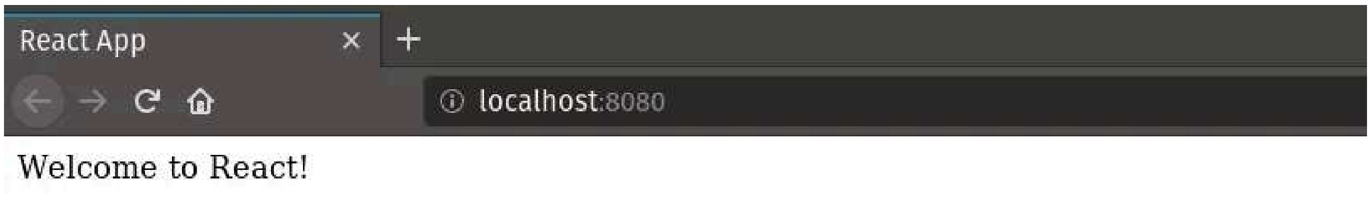
This plugin automatically creates an HTML file that serves the JavaScript bundles created by Webpack.

As a last step, we modify the `Scripts` inside our `package.json` to include:

```
  "start": "webpack-dev-server --mode development --open --hot"
```

This allows us to run `npm start` to start the Webpack development server. The `open` and `hot` options opens our React app in a browser once it's ready and enables hot reloading, respectively.

That should give us our final working React app.



Welcome to React!

Alternatives to Webpack

While Babel is pretty much the de-facto tool used for JavaScript compilation, Webpack does have some alternatives, such as Parcel(https://parceljs.org/).

Even npm has alternatives, such as yarn (https://yarnpkg.com/lang/en/), which is the what I prefer over npm.

Each of them have their pros, cons and differences, and it's up to you to explore them and choose the right tool for the job.

Conclusion

If you are a beginner it might be worth it to just use `create-react-app` or a similar tool for whatever framework/library you are working with to skip the (admittedly) tedious initial configuration, but if you are building a project that you plan to take to completion, I feel it is important to have control over every aspect of it to avoid bloat and to allow for micromanagement, and in these cases knowing your node_modules can be a good thing.

All files discussed in this article can be found here: https://github.com/agzuniverse/webpack-babel-react

Find me on **GitHub** and **Twitter**.

JavaScript React Webpack Babel Create React App

Written by Aswin G

Follow

120 Followers

I like to code. CS student. Occasionally blogs, I like to think I can play the guitar.

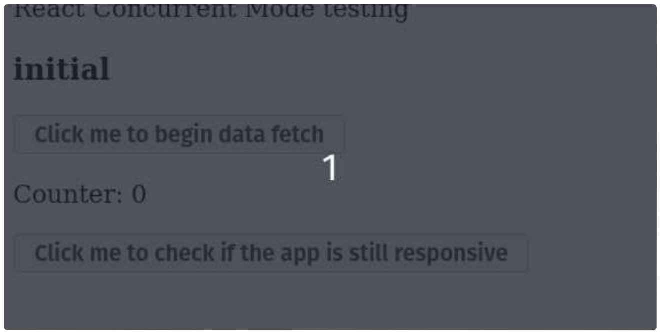
More from Aswin G

Aswin G

Creating a Middleware in Golang for JWT based Authentication

Find out how to create a middleware for a Golang web application, and how to use this...

Apr 15, 2020 283


React Concurrent Mode testing
initial
Click me to begin data fetch
1
Counter: 0
Click me to check if the app is still responsive

Aswin G

Concurrent React Mode: Using Suspense and useTransition to...

Originally posted on Hackernoon.

Apr 15, 2020 15

Aswin G

Getting Avengers Endgame tickets with Python

It's not very often when a movie generates so much hype that you know ahead of time that...

May 14, 2019 376

Aswin... in We've moved to freeCodeCamp.org/ne...

What I experienced at Google Summer of Code

This article is a quick summary of my experience participating in and completing...

Aug 24, 2018 896 1

See all from Aswin G

Recommended from Medium

Pradeep Tiwari (Solution Architect)

A Comprehensive Comparison: Vite vs. Webpack

As a solution architect, optimizing performance is a cornerstone of my role. In...

Mar 23 3

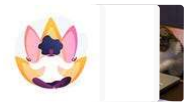
Shubham Lohar

Best Practices for Securing Node.js Applications in Production

Node.js is a popular choice for server-side application development due to its efficient...

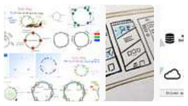
Jun 23

Lists



Stories to Help You Grow as a Software Developer

19 stories · 1167 saves



General Coding Knowledge

20 stories · 1343 saves

Medium's Huge List of Publications Accepting...

Generative AI Recommended Reading

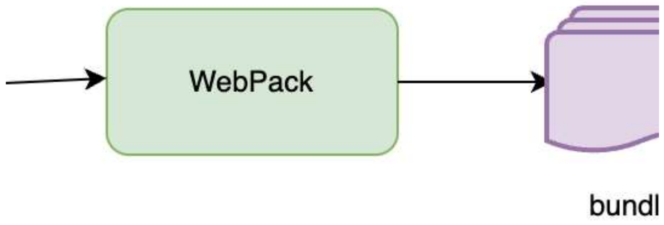
312 stories · 3017 saves 52 stories · 1169 saves

M Mia Hossain

How To Create a React App with Vite

This tutorial is for those who want to create a React project using Vite for the very first tim...

Feb 1 71

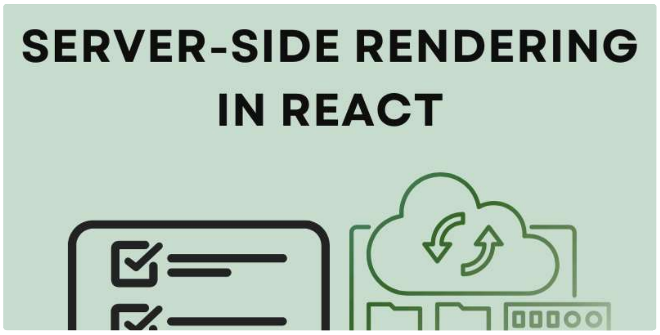


H Himanshu Gaur

Webpack and the Angular compiler

What is the role of webpack?

Jan 22 1



Shikha Rajput

Client-Side Rendering (CSR) and Server-Side Rendering (SSR) in...

Have you ever wondered why some websites load faster than the others? The answer lies i...

Jan 8 6

Has San

Difference Between Redux and Redux Toolkit

In the realm of state management for React applications, Redux has become a powerful...

Jan 3 32

See more recommendations