Published in JavaScript in Plain English

Chan Jing Hong Follow

Apr 14, 2020 · 4 min read ★ · Listen

How to Publish Your React component on npm

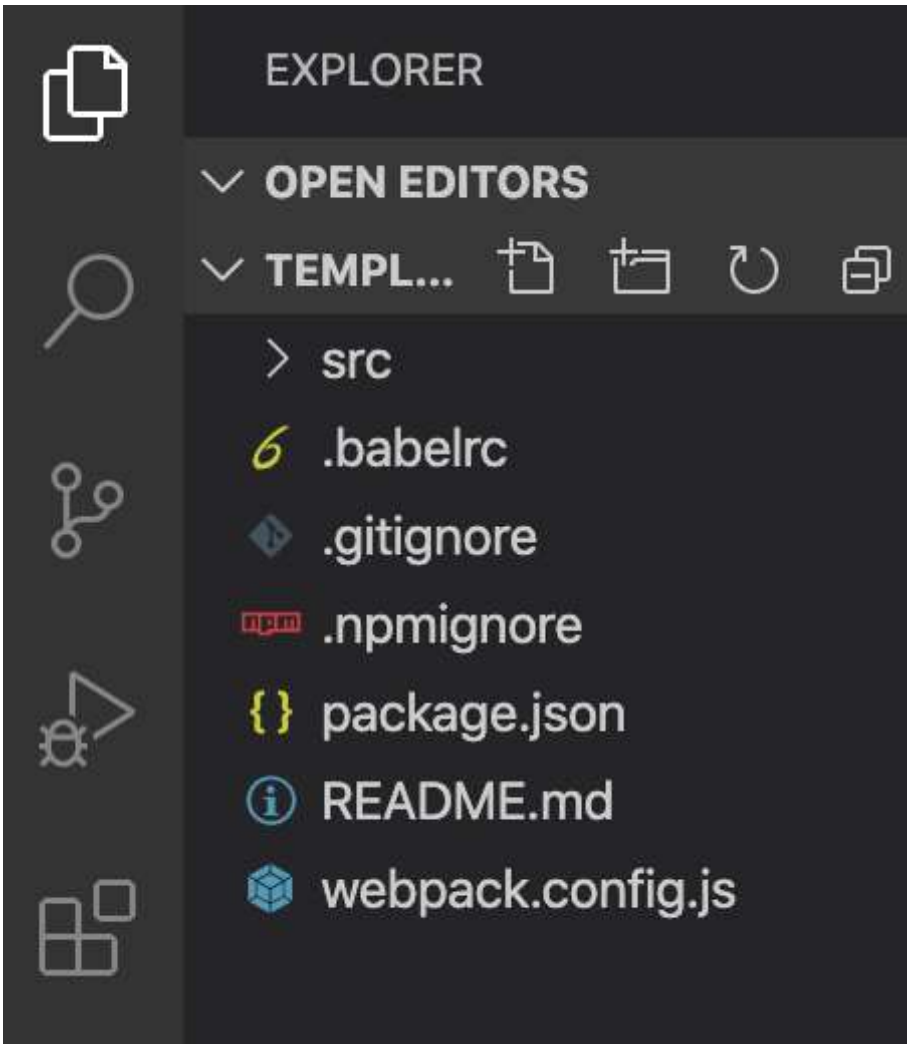
This tutorial is for the more experienced React developers. If you're just looking to learn how to build a React app, there are tons and tons of tutorials available online!

Ok, so you've read tutorials, figured out how to set up a React project using `create-react-app` , learned how to install npm packages, and finally created your very own functional React app, and to that that I say congratulations!

come up with some of your own cool React components! And now you're wondering, how might I share that with the rest of the world?

Packaging your React component

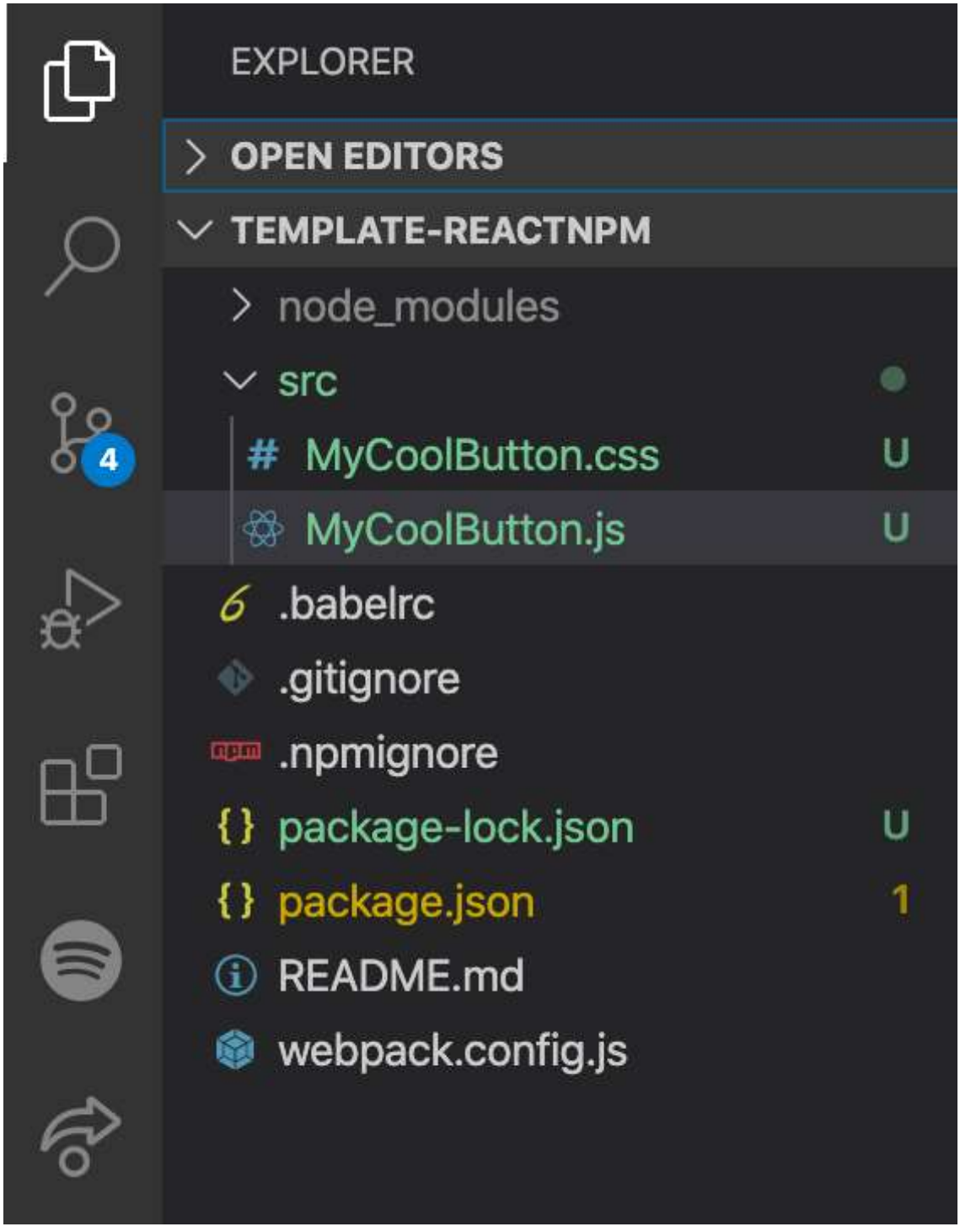
Before publishing anything to npm, the first step is to put your component files into a nice, clean package. I've created a template to make everything easier to follow, go ahead and download this first. And once you're done, open the folder in your favorite code editor.



my favorite one is definitely Visual Studio Code

1. Adding your React component

`MyCoolButton.js` . Make sure to also include all the files that are required by your component. In this case, I've also added in `MyCoolButton.css` .



Add your component files into /src folder

2. Sort out dependencies

The next thing you have to do is to figure out if your component requires any other

```
// This component also requires 'react',
// but it has already been included in the
// list of dependencies in package.json
import React from 'react';

// This component requires prop-types
import PropTypes from 'prop-types';

import './MyCoolButton.css';

const MyCoolButton = ({ type, title, onClick }) => (
  <button
    type={type}
    className="container"
    onClick={onClick}
  >
    {title}
  </button>
);

MyCoolButton.propTypes = {
  title: PropTypes.string.isRequired,
  type: PropTypes.string,
  onClick: PropTypes.func,
};

MyCoolButton.defaultProps = {
  type: 'button',
  onClick: () => {},
};

export default MyCoolButton;
```

Now let's open up `package.json`, and add the dependencies in. Normally you would add your dependencies under `dependencies`, but in this case, you have to add them into `peerDependencies` and `devDependencies`. This is how your `package.json` should look like.

```
    "license": "MIT",
    "scripts": {
        "build": "webpack"
    },
    "peerDependencies": {
        "prop-types": "^15.6.0",
        "react": "^16.0.0",
        "react-dom": "^16.0.0"
    },
    "devDependencies": {
      "prop-types": "^15.6.0",
        "babel-core": "^6.21.0",
        "babel-loader": "^7.1.4",
        "babel-preset-env": "^1.6.1",
        "babel-preset-react": "^6.16.0",
        "babel-preset-stage-0": "^6.24.1",
        "css-loader": "^3.5.1",
        "path": "^0.12.7",
        "react": "^16.0.0",
        "react-dom": "^16.0.0",
        "style-loader": "^1.1.3",
        "webpack": "^4.5.0",
        "webpack-cli": "^3.2.1"
    },
    "dependencies": {}
}
```

Notice that the list of dependencies is empty

After that, run `npm install` (or if you use yarn, `yarn install`) to install the required dependencies.

3. webpack.config.js

Next up, we'll use Webpack to bundle our React components into a nice CommonJS module. Click here to learn more about Webpack and what it does. Let's start by opening up `webpack.config.js` .

```
const path = require('path');

module.exports = {
```



```
    libraryTarget: 'commonjs2',
  },
  module: {
    rules: [
      {
        test: /\.js?$/,
        exclude: /(node_modules)/,
        use: 'babel-loader',
      },
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader'
        ]
      }
    ],
  },
  resolve: {
    alias: {
      'react': path.resolve(__dirname, './node_modules/react'),
      'react-dom': path.resolve(__dirname, './node_modules/react-dom'),
    }
  },
  externals: {
    // Don't bundle react or react-dom
    react: {
      commonjs: "react",
      commonjs2: "react",
      amd: "React",
      root: "React"
    },
    "react-dom": {
      commonjs: "react-dom",
      commonjs2: "react-dom",
      amd: "ReactDOM",
      root: "ReactDOM"
    }
  }
};
```

This is the file that contains the different configurations that Webpack will use to

`output` — This specifies the path to the output file. You should update the `filename` to match your component's file name.

`module.rules` — This is an array of rules that we are applying to our module. The first rule looks for any `.js` file and tries to transpile it using `babel-loader`. However, the second rule is only relevant if your component uses `css`. If your component uses any `css`, you will have to add this in. Click here to find out more about `css-loader` and `style-loader`.

> ***Important Note***
>
> *If your component uses `.sass` or `.scss`, you would have to also include `sass-loader` by adding it to your Webpack module rules as well as adding it to your list of `devDependencies` in `package.json`. Click here to see a list of loaders that you might need to use.*

I have only gone through the options that are relevant to this tutorial. Check out the full list of options here.

4. Bundle em' up

Run `npm run build` (or if you use yarn, `yarn build`). This should generate a folder called `/lib` which contains your freshly packaged component, in this case, `MyCoolButton.js`.

5. Test your component

Before you publish it to the world, it would make sense to take your package for a test drive first (or if you're absolutely confident that it will work, feel free to skip ahead to **Publishing To NPM**).

Run `npm pack`. This will generate a `.tgz` file at the root directory.

Open up any React application that you want to test your new package, and then run `npm install path_to_tgz_file`. Replace `path_to_tgz_file` with your actual path.



```
import React from 'react';
import MyCoolButton from 'mycoolbutton';

function App() {
 return (
    <div>
      <MyCoolButton title="Click Me!"/>
    <div>
  )
}

export default App;
```

Does it work? Great! Let's move on to actually publishing it to the world.

Publishing to NPM

Alright so now you have your `/lib` folder with your newly packaged component ready to go, the next thing to do it just publishing it to npm.

Run `npm login` and login with your npm account. Create one here if you don't already have one.

After you're logged in, the final thing to do is `npm publish` , and you're all set!

Links

- Published npm package

React

- https://reactjs.org/

Npm

- https://www.npmjs.com/signup
- https://www.npmjs.com/package/@material-ui/core
- https://www.npmjs.com/package/react-router-dom
- https://www.npmjs.com/package/react-spinners

Yarn

- https://yarnpkg.com/

Webpack

- https://webpack.js.org/
- https://webpack.js.org/loaders/css-loader/
- https://webpack.js.org/loaders/style-loader/
- https://webpack.js.org/loaders/sass-loader
- https://webpack.js.org/loaders/