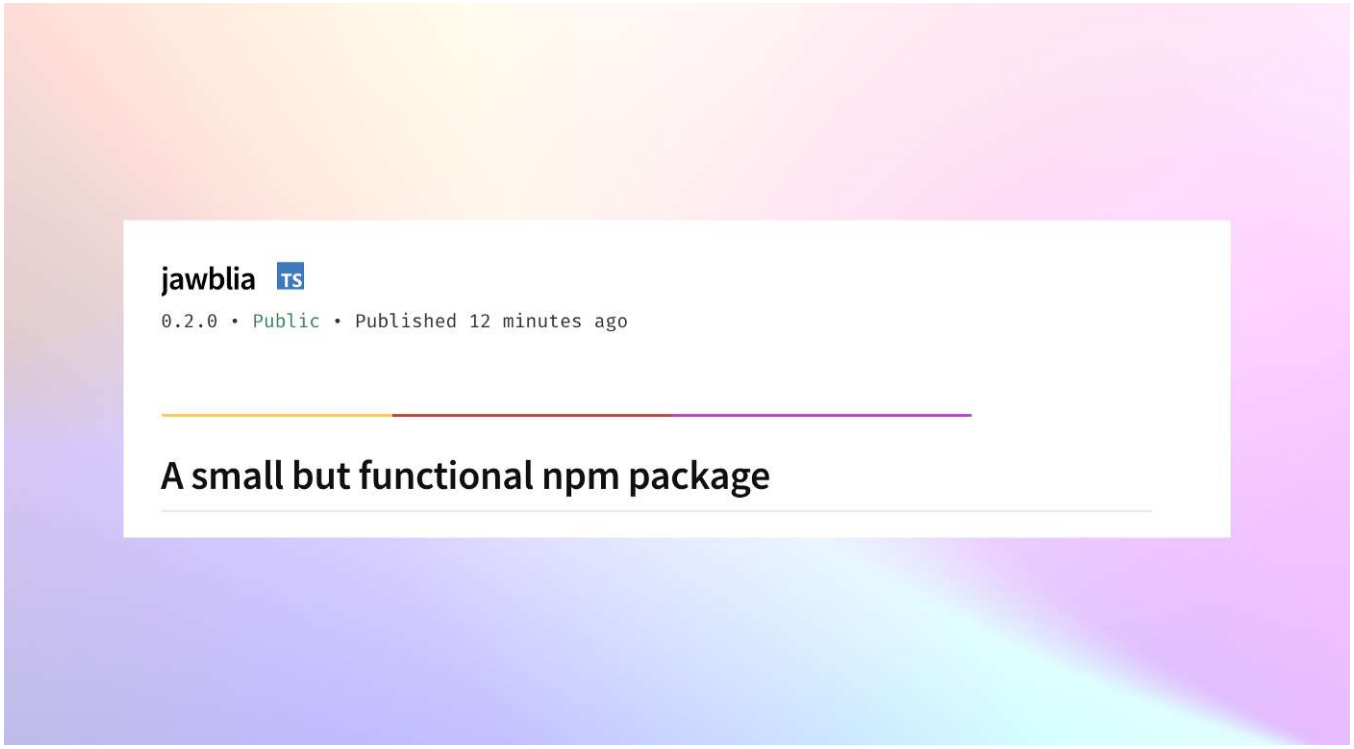
Published in Level Up Coding

JB Follow

Apr 1, 2021 · 7 min read · Listen

Publish React components as an npm package



This article will review how to publish React components as an npm package with Babel 7 (the latest version at the time of writing) and common errors.

I found myself copy-pasting my React components from project to project and wanted to create an npm package so I could import them easily. To do that, I had to learn how to publish an npm package. It was hard to find much updated info online about publishing React components with Babel 7, and I was getting plenty of build errors, so I decided to write this as a reference. This is going to assume React competence, but little-to-no experience using npm.

Take a look at the final npm package and GitHub repo

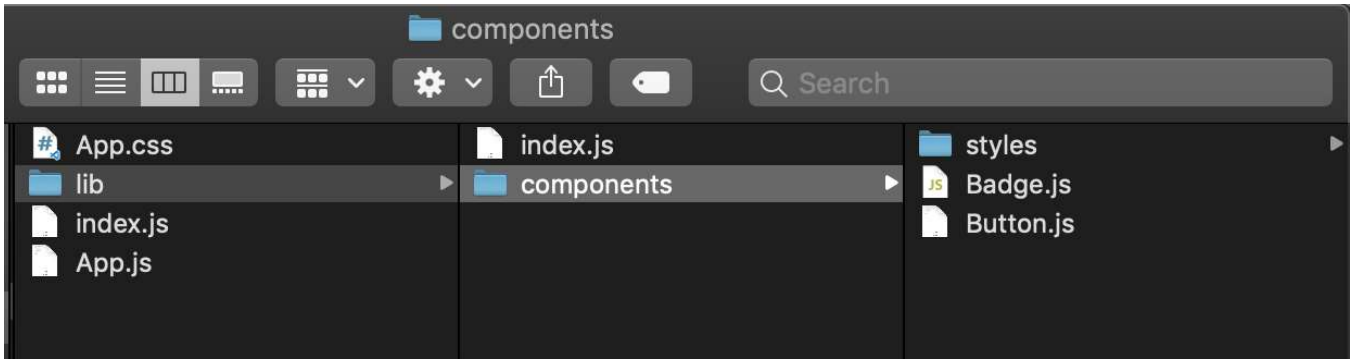
Pre-conditions:

- A React app. I created the boilerplate for this article with `npx create-react-app npm-test`. Run the app on your local machine.

1. Create and isolate components to publish

In the boilerplate app, I went into the **src** folder and deleted everything besides App.js, app.css, and index.js.

I also added a folder called `lib` that will store everything I want to publish on npm. Inside `lib`, there is a folder called `components` to store the component elements and a file called `index.js` to export them. (This seems to be a standard, based on other tutorials and the Carbon's library.)



Inside the components folder, I create new files called `Button.js` and `Badge.js`, which will be the components to use from the npm package. The code for these components isn't very important, but I'll list it below.

> Button.js

```
import React from 'react';

const Button = (props) => {
   return (
      <button className={`btn btn--${props.kind} CTA`}
        data-id={props.id}
        type={props.type}
        name={props.name}
        value={props.value}
        disabled={props.disabled}
        onClick={props.handleClick}>
```

```
export default Button;
```

> Badge.js

```
import React from 'react';

const Badge = (props) => {
   return (
      <div className={`badge ${!props.value ? 'badge--none' :''} `}
        <h4 className="heavy">{props.value || 0}</h4>
      </div>
     )
}

export default Badge;
```

> *IMPORTANT: To style these components, they will either need 1. css imported directly into the component, 2. inline styles, which can be applied directly to the components or with the help of a css-in-js package like styled-components, or 3. pre-compiled and bundled sass/scss, which I believe can be applied in a React app with a sass-loader but is outside the scope of this article.*

These components are both in the `components` folder. Then, we'll add them to the index.js file:

```
import Badge from './components/Badge';
import Button from './components/Button';

export { Badge, Button };
```

1b. Create a repo for the components

This is technically optional, since components don't have to be on Github for them to be published in npm. But it's very convenient, because the README for your repo will automatically be populated as the package's documentation in npm.

2. Install Babel and build the `dist` folder

```
npm install -save @babel/polyfill
```

Using the versions of babel that begin with @ signs is important for matching the presets. Either way, using unscoped versions of Babel (without @ signs) and scoped presets (designated with @ signs) will cause build errors.

In the top-level folder of your project, add a file called `babel.config.json` and add the following presets:

```
{
 "presets": [
  [
   "@babel/env",
    {
     "targets": {
     "edge": "17",
     "firefox": "60",
     "chrome": "67",
     "safari": "11.1"
      },
   "useBuiltIns": "usage",
   "corejs": "3.6.5"
    }
],
   "@babel/preset-react"
]
}
```

`@babel/env` tells the browser which versions it should target, and `@babel/preset-react` allows Babel to compile JSX.

In `package.json` , under `scripts` , replace the build script with the following:

```
"build": "rm -rf dist && NODE_ENV=production babel src/lib --out-dir
dist --copy-files";
```

This will copy the `src/lib` to a new folder called `dist` . This folder is invisible but will be added to your root folder after build.



```
Successfully compiled 3 files with Babel (864ms).
(base) ~/npm-test
juliabell$ ls -a
./                  .gitignore          node_modules/       src/
../                 README.md           package-lock.json   yarn.lock
.DS_Store           babel.config.json   package.json
.git/               dist/               public/
```

fun fact! you can customize the color scheme of your CLI

3. Alter the package.json for publishing

This is the good part! The `package.json` must be changed to publish to npm.

This is the first part of my `package.json`:

```
"name": "npm-test",
"version": "0.1.0",
"private": true,
```

The `name` here has to be a unique name that hasn't been taken by an existing npm package (you can check if a name is taken using npm search). `version` is the package version, and must be changed whenever it's republished. Version syntax indicates major, minor, and patch releases and more about it can be found here in the npm docs.

`description`, `keywords`, and `author` are all optional fields that will give potential end users a better idea of the package. Full `package.json` here.

```
"name": "jawblia",
"description": "Two test React components",
"author": "Jawblia",
"keywords": ["react", "components", "ui"],
"version": "0.1.0",
"private": false,
"main": "dist/index.js",
"module": "dist/index.js",
"files": [ "dist", "README.md" ],
```

The file is ready for `npm publish`.

4. Use the new package

Check in the CLI and on your npm profile that the package has published. To make sure it's working, open a different project on your local machine, and try to use the package:

```
npm install jawblia
```

In the new project, try to use one of your components by importing it:

> App.js

```
import { Button } from 'jawblia';
import Flex from './layout/Flex';

function App() {

return (
  <Flex middle center column className="page" gap={1.5}>
   <h3>This is my new project</h3>
   <Button label="test" kind="primary"/>
  </Flex>

);

}

export default App;
```

In the browser, we see:



The button imported from the npm package is working

We're able to use any of the props in the original component and change the label, type, and style of the button. The npm package is working.

Coda: Some caveats

There're a few things missed here worth mentioning.

Types:

If you import your new package, you'll notice a warning like this:

```
module "/Users/juliabell/uigarden/node_modules/jawblia/dist/index"

Could not find a declaration file for module 'jawblia'.
'/Users/juliabell/uigarden/node_modules/jawblia/dist/index.js' implicitly has an 'any'
type.
  Try `npm install @types/jawblia` if it exists or add a new declaration (.d.ts) file
containing `declare module 'jawblia';` ts(7016)

No quick fixes available
```

All React npm packages are also premised to be used with Typescript. Declaring types for

scss:

I used scss for the styling on my components, and added a separate folder in the `components` folder called `styles`. This styling all compiles into the App.css sheet in the main src folder. If you don't want the scss to precompile, I found this Stack Overflow answer addressing it.

Babel versions:

If your preset syntax and imported Babel versions aren't aligned you will likely get an error message like:

```
ReferenceError: [BABEL] src/lib/index.js: Unknown option:
/Users/juliabell/skylight/node_modules/babel-preset-react-
app/index.js.overrides. Check out
[<http://babeljs.io/docs/usage/options/>]
(<http://babeljs.io/docs/usage/options/>) for more information about
options.

A common cause of this error is the presence of a configuration
options object without the corresponding preset name. Example:

Invalid: { presets: [{option: value}] } Valid: { presets:
[['presetName', {option: value}]] }
```

or

```
Requires Babel "7.0.0-0" but was loaded with "6.26.3" #8482
```

From looking online it seemed like these errors can be caused by having different versions of babel installed globally vs. locally, having a `.babelrc` or `babel.config.json` file in the parent folder, or using incorrect syntax with the presets. In my case, I was using incorrect syntax in presets and adding scoped Babel versions (`@ babel`) when I had originally installed unscoped Babel versions (`babel`).

Licenses:

References:

Usage Guide · Babel

There are quite a few tools in the Babel toolchain that try to make it easy for you to use Babel whether you're an...

babeljs.io

How To Publish a React Component Library

Now that you've built something useful, share it with the world

betterprogramming.pub

Export React Components as Node Modules to NPM

Pluralsight Guides

www.pluralsight.com

https://itnext.io/how-to-package-your-react-component-for-distribution-via-npm-d32d4bf71b4