Angular 17 Component Library wired for Web Components!

cory lewis · Follow
7 min read · Dec 18, 2023

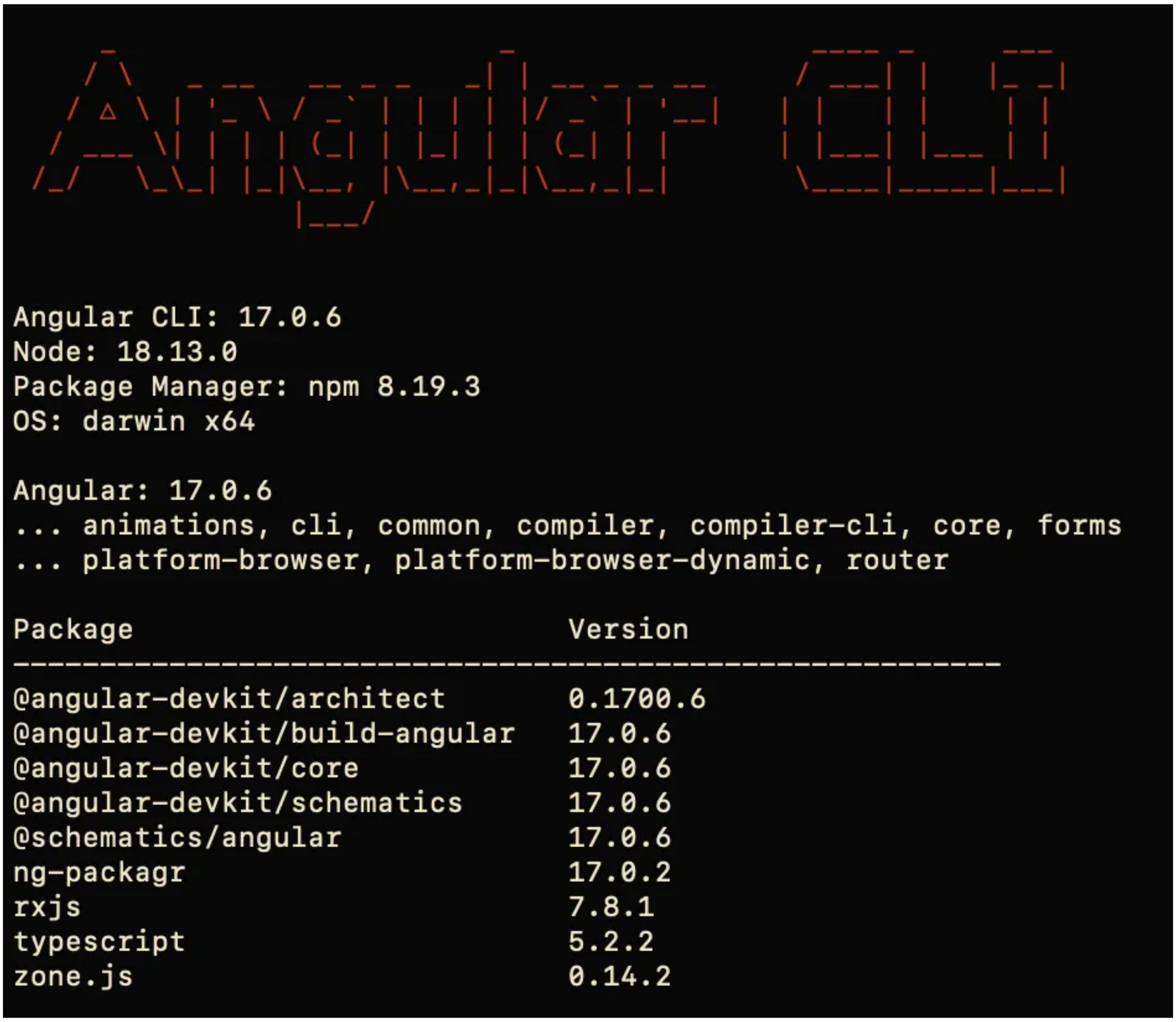
147 5

I found that there are quite a few of resources out on the web for creating an Angular library but not so much up to date walkthroughs explaining how to take your Angular CLI generated library and export one or many of its components out as framework agnostic web component(s).

> *In this article, we will create two libraries: a pure Angular library and a web component library. The web component library will wrap around the Angular library and use web component hooks to expose the components as native HTML elements.*

Important!

This is my current projects environment versions for your reference while stepping through this tutorial



```
     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 17.0.6
Node: 18.13.0
Package Manager: npm 8.19.3
OS: darwin x64

Angular: 17.0.6
... animations, cli, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

Package                         Version
---------------------------------------------------------
@angular-devkit/architect       0.1700.6
@angular-devkit/build-angular   17.0.6
@angular-devkit/core            17.0.6
@angular-devkit/schematics      17.0.6
@schematics/angular             17.0.6
ng-packagr                      17.0.2
rxjs                            7.8.1
typescript                      5.2.2
zone.js                         0.14.2
```

ng version

Getting Started

The first step of this tutorial is to install the Angular CLI globally, which will allow us to use the "ng" commands in the terminal. Then, we will create our initial project using the "ng new" command. In this tutorial, we will work with two libraries: a pure Angular 17 library ("my-library") and a web component library ("my-library-web-components"). The web component library will be framework agnostic and will include the web component and the Angular runtime. If you only want to use the web component library, you can skip publishing the Angular 17 library to npm.

```
# Install the latest Angular CLI globally
npm i -g @angular/cli

# Generate the Angular workspace
ng new my-workspace --no-create-application

# Create your library
ng generate library my-library

# Create your Web Component Library
ng generate application my-library-web-components
```

Add Storybook (optional)

Storybook is a tool that enables developers to create scenarios and playgrounds for their components. It also provides a beautiful interface for collaboration and component reusability among large teams. For a library like this, publishing a playground for your consumers to explore would be a nice feature.

```
npx storybook@latest init
```

> *Important: To scaffold both Angular projects/libraries, you will need to run the storybook init command twice. Each time, you will be prompted to select a project to scaffold. Please choose a different project for each run.*

Add Prettier (optional)

By formatting the code according to your specified schema, Prettier will help you maintain a consistent coding style throughout the code base. The package we are going to install will also add a husky pre-commit hook to the angular project, ensuring that your coding standards are met before each commit.

```
# Run this command in your terminal and follow prompts
ng add @schuchard/prettier

# Run prettier
npm run prettier
```

> *More about the @schuchard/prettier package can be found here*

Add Linting (optional)

We will want to install the eslint schematic to gain access to its toolchain for linting our libraries

```
# Run this command to add the schematic
ng add @angular-eslint/schematics

# Add lint configuration for my-library
ng g @angular-eslint/schematics:add-eslint-to-project my-library

# Add lint configuration for my-library-web-components
ng g @angular-eslint/schematics:add-eslint-to-project my-library-web-compone
```

> *See this github repository for additional information about the schematic and its toolchain*

Building "my-library"

```
ng build my-library --configuration production
```

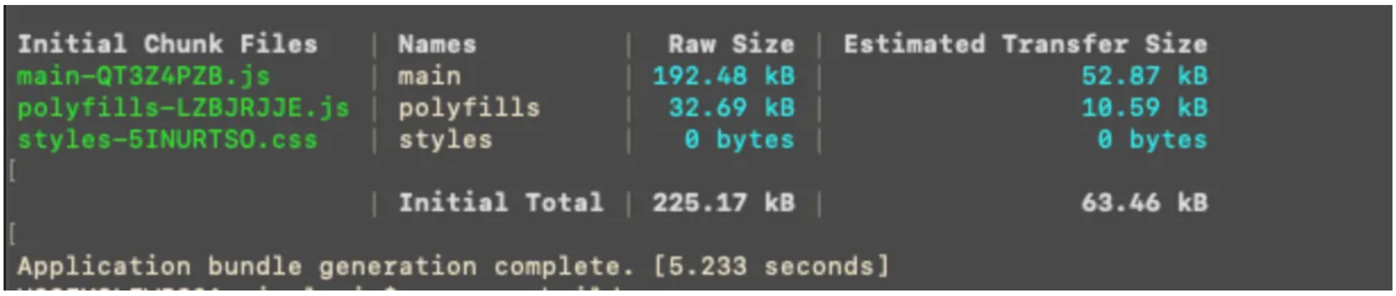
> *You should see a dist folder generated at the root of your application with a folder inside called "my-library". Success your library is now building!*

Building "my-library-web-components"

Now at this point our angular library is building correctly but our templated build process for "my-library-web-components" has some work to be done in order to get things working cleanly.

To start let's see how it currently builds first.

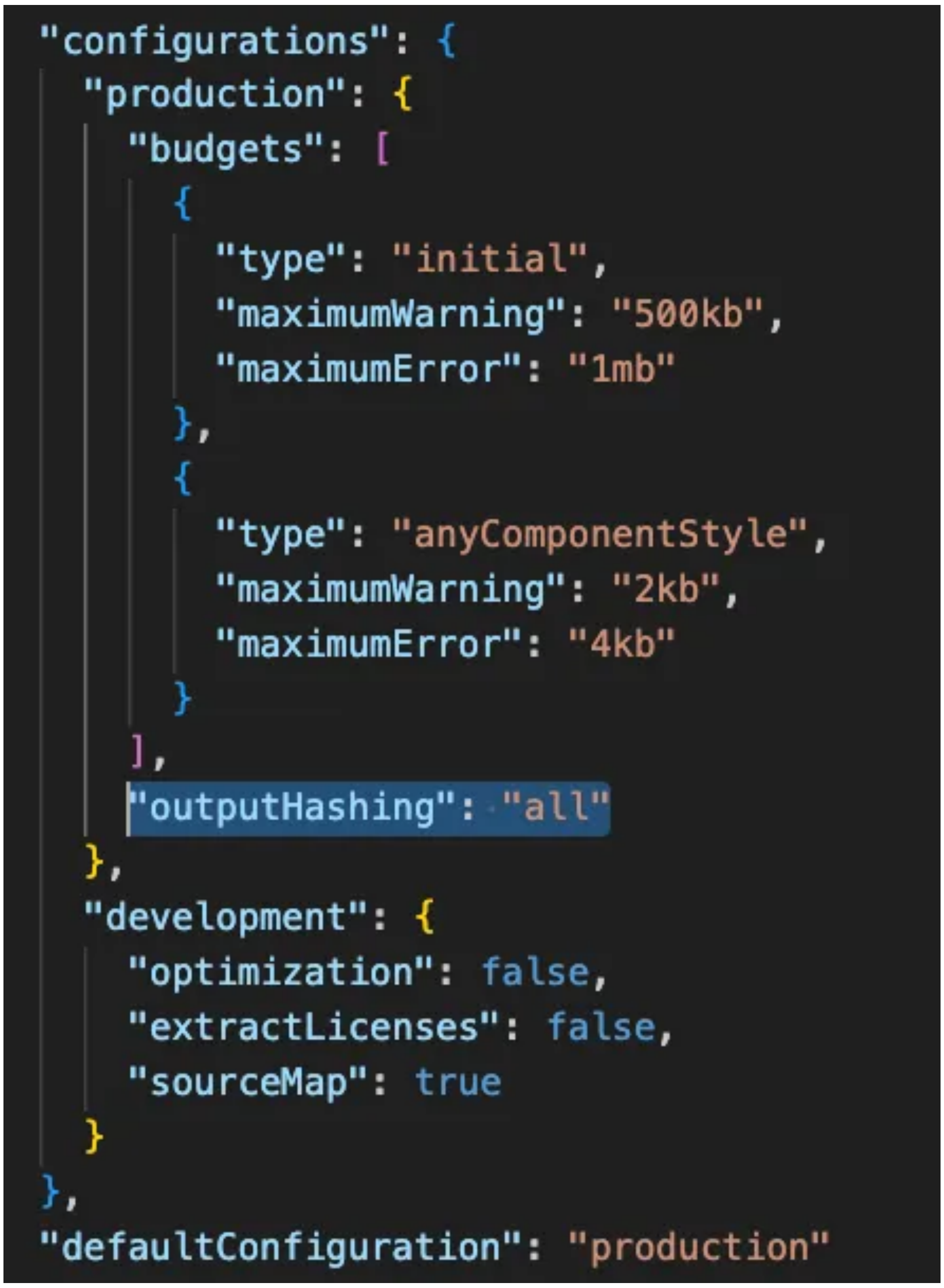
```
# Build the library to see its current output
ng build my-library-web-components --configuration production
```



```
Initial Chunk Files     | Names         |  Raw Size | Estimated Transfer Size
main-QT3Z4PZB.js        | main          | 192.48 kB |                52.87 kB
polyfills-LZBJRJJE.js   | polyfills     |  32.69 kB |                10.59 kB
styles-5INURTSO.css     | styles        |   0 bytes |                 0 bytes
[
                        | Initial Total | 225.17 kB |                63.46 kB
[
Application bundle generation complete. [5.233 seconds]
```

You may notice that each file has a build hash appended to its name. This is normal for a traditional application build, but not for our use case. We want to have predictable file names for our package.json entrypoint. Therefore, we need to disable "outputHashing" in the build configuration for my-library-web-components inside angular.json.

Delete the highlighted line below from angular.json my-library-web-components.build

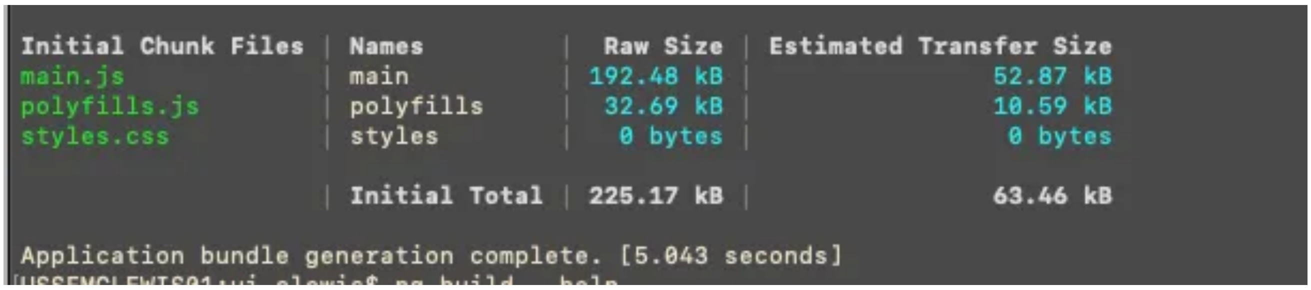


```
"configurations": {
  "production": {
    "budgets": [
      {
        "type": "initial",
        "maximumWarning": "500kb",
        "maximumError": "1mb"
      },
      {
        "type": "anyComponentStyle",
        "maximumWarning": "2kb",
        "maximumError": "4kb"
      }
    ],
    "outputHashing": "all"
  },
  "development": {
    "optimization": false,
    "extractLicenses": false,
    "sourceMap": true
  }
},
"defaultConfiguration": "production"
```

Let’s build again and see if that changed anything

```
ng build my-library-web-components --configuration production
```

Your build output should look similar to this



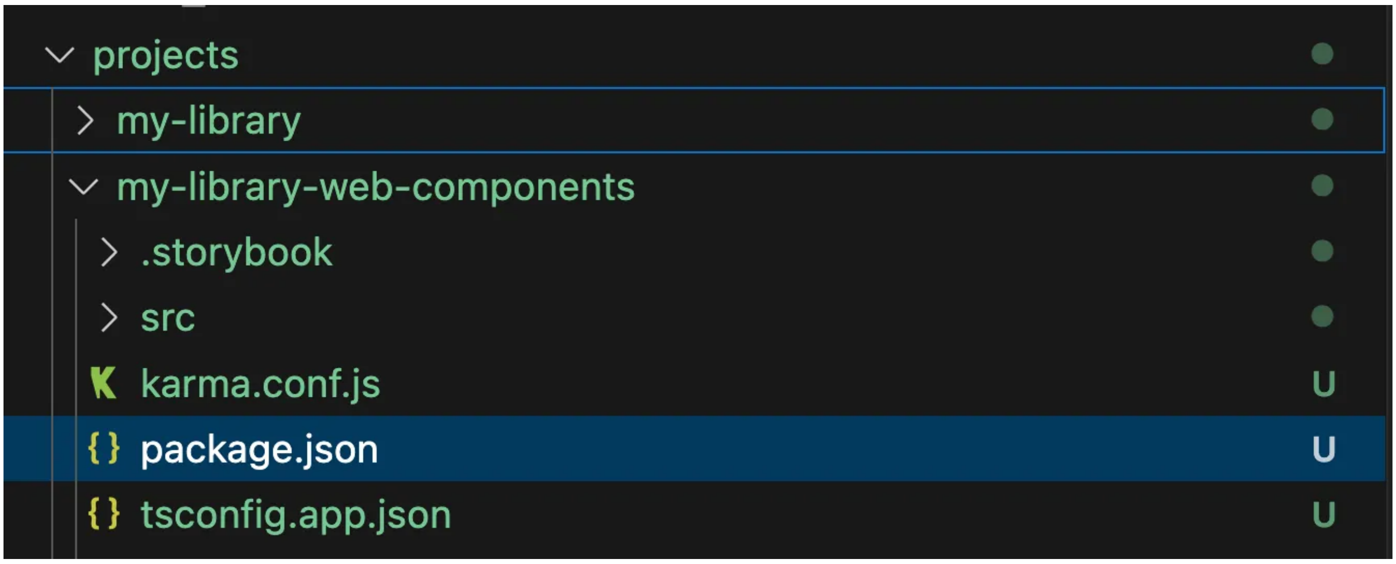
```
Initial Chunk Files | Names         |  Raw Size | Estimated Transfer Size
main.js             | main          | 192.48 kB |                52.87 kB
polyfills.js        | polyfills     |  32.69 kB |                10.59 kB
styles.css          | styles        |   0 bytes |                 0 bytes

                    | Initial Total | 225.17 kB |                63.46 kB

Application bundle generation complete. [5.043 seconds]
```

> *Excellent! By disabling “outputHashing”, we have ensured that the file names remain the same for every build.*

Next we need to create a `package.json` in the root of my-library-web-components



And inside the package.json we will define the following:

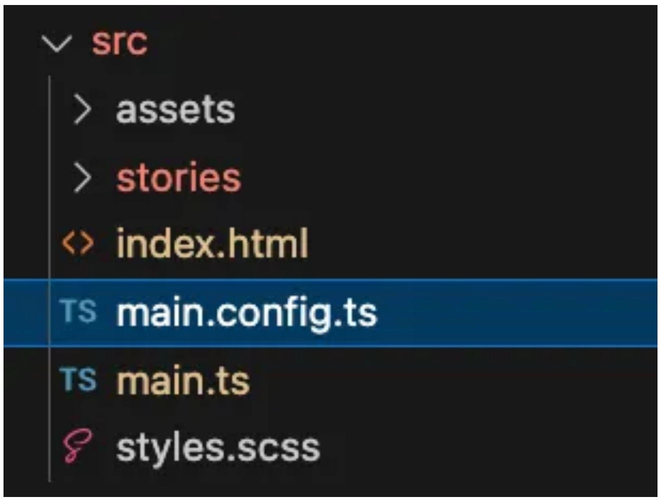
```
{
  "name": "my-library-web-components",
  "version": "0.0.1",
  "main": "bundle.js"
}
```

> *This is very rudimentary but a good starting point for a basic package.json needed to publish to npm*

Setup web component bootstrapping

The ng generate command created some basic files in the projects/my-library-web-components/src/app/* directory. However, we only need the app.config.ts file. Move this file up a directory and rename it to main.config.ts. It should be next to your main.ts file. You can delete the rest of the files in the app/* directory.

Your src directory should look like this when you are done:



Replace ALL content inside **main.config.ts** with:

```
import { ApplicationConfig } from '@angular/core';

export const appConfig: ApplicationConfig = {
    providers: [],
};
```

Replace ALL content inside **main.ts** with:

```
import {createApplication} from '@angular/platform-browser';
import {appConfig} from './main.config';
import {createCustomElement} from '@angular/elements';
import { MyLibraryComponent } from 'my-library';
import { ApplicationRef } from '@angular/core';

(async () => {
  const app: ApplicationRef = await createApplication(appConfig);
```

```
    // Define Web Components
    const myLibraryComponent = createCustomElement(MyLibraryComponent, {inject
    customElements.define('my-library-component', myLibraryComponent);
  })();
```

Modify your **index.html** to use your web component like:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>EtlpWc</title>
        <base href="/" />
        <meta name="viewport" content="width=device-width, initial-scale=1"
    </head>
    <body>
        <my-library-component></my-library-component>
    </body>
</html>
```

> *This is the index.html located in the same directory as your main.ts and main.config.ts*

Smoke Test the web component

We will want to first build the my-library project and afterwards build my-library-web-components because there is a dependency between them.

```
ng build my-library -c production
ng build my-library-web-components -c production
```

Now that everything is built we want to test that it works

```
# To view our distribution build we will need to use a tool like http-server
# install globally with:
npm i http-server -g

# Start the server
http-server ./dist/etlp-wc/browser
```

View your app at http://127.0.0.1:<YOUR_PORT>/index.html

my-library works!

your index.html loading successfully

Create importable bundle file

We have made good progress so far, but we still have one major issue to resolve. If you look at the dist/my-library-web-components/browser/index.html file, you will see that it loads two files: polyfills.js and main.js, using the <script> tag. However, we want to create a library with only one entry point in our package.json file, not two. To fix this, we will generate a third file called bundle.js.

All we need to do is open our `angular.json` at the root of our project and add a script at path `my-library-web-components.architect.build.options.scripts`.

Add this scripts array to your scripts build options:

```
"scripts": [
    {
        "input": "./scripts/postbuild-bundler.js",
        "bundleName": "bundle",
        "inject": false
    }
]
```

Now at the root of our project lets create `scripts` directory with postbuild-bundler.js inside with content:

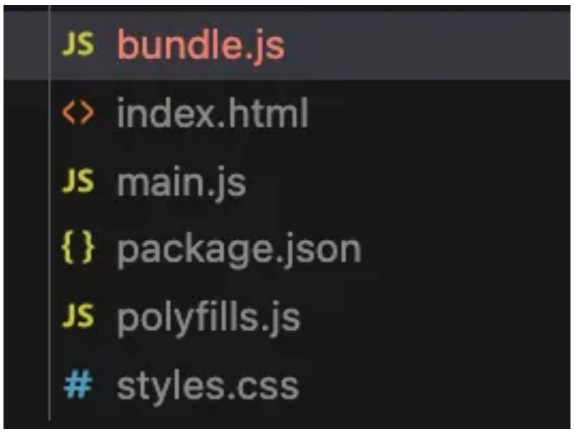
```
async function loadModules() {
    // Import the polyfills and main modules and wait for them to finish
    await import('./polyfills.js');
    await import('./main.js');

    // Do any extra stuff you need for init here.
  }
```

```
loadModules();
```

> *This file will serve as the bundle module loader for the web components library*

Verify everything is setup correctly by building my-library-web-components. `ng build my-library-web-components -c production` . You should see a hierarchy like this in your web component library distribution folder:



```
bundle.js
index.html
main.js
package.json
polyfills.js
styles.css
```

Last Step

In order to publish you will need to change directory into your distribution folder and publish. Below is an example:

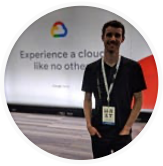
```
# Publish your library
cd dist/my-library
npm version 1.0.0
npm publish

# Publish your web component library
cd dist/my-library-web-components/browser
```

```
  npm version 1.0.0
  npm publish
```

And that's it!

Angular Storybook Terraform Web Development Angular Cli



Written by cory lewis

11 Followers

Engineer | Developer | Tech Enthusiast

Follow

More from cory lewis

cory lewis

OpenSSL in C++

One of the most powerful and opaque libraries for server side c++ is the openssl library. It has many useful tools built into it but fails...

· 4 min read · Aug 26, 2020

8

See all from cory lewis

Recommended from Medium

Melani Sandalika in Stackademic

Exploring the Power of Angular Elements: Building Modular We...

Hello everyone! 😍

4 min read · Jan 30, 2024

31

🪄 OZ 🎩

Angular Signals: Best Practices

In this article, I share my experience of working with Angular Signals after almost ...

4 min read · 6 days ago

261 4

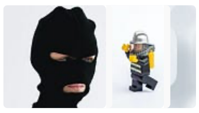
Lists

General Coding Knowledge

20 stories · 1055 saves

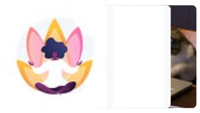
Coding & Development

11 stories · 522 saves



Tech & Tools

16 stories · 189 saves



Stories to Help You Grow as a Software Developer

19 stories · 930 saves

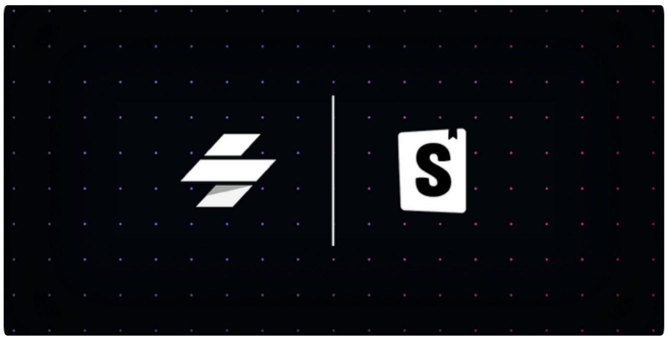
Alain Boudard

Angular library lazy loading

In this article, I wanted to focus on some of the options available with Angular to lazy…

6 min read · Mar 7, 2024

87



リン (linh) in Goalist Blog

Storybook & Stencil comparisons

I. What are they

3 min read · Nov 17, 2023

1

Paul Gschwendtner in Angular Blog

Signal inputs available in developer preview

The new inputs provide developers with new ways that:

5 min read · Feb 22, 2024

606 9

Eniela P. Vela

10 Best Angular Libraries in 2024

Angular is a popular open-source framework for building dynamic web…

4 min read · Jan 26, 2024

358 3

See more recommendations