Building Custom Elements / Web Components with Angular 6

5-step guide using Angular CLI and Angular Elements

Tomek Sułkowski Follow

May 2, 2018 · 4 min read

With the newest Angular CLI (version 6, released 2018–04–03) and the new addition to Angular family—the **Angular Elements** package it's extremely easy to create **native custom elements**.

If you don't know what custom elements are or what is the connection to Angular, there are several great talks that introduce to these concepts, I highly encourage you to watch one of these.

So now, without further ado, let's see some code!

1. Install Angular CLI 6 and initialize the project

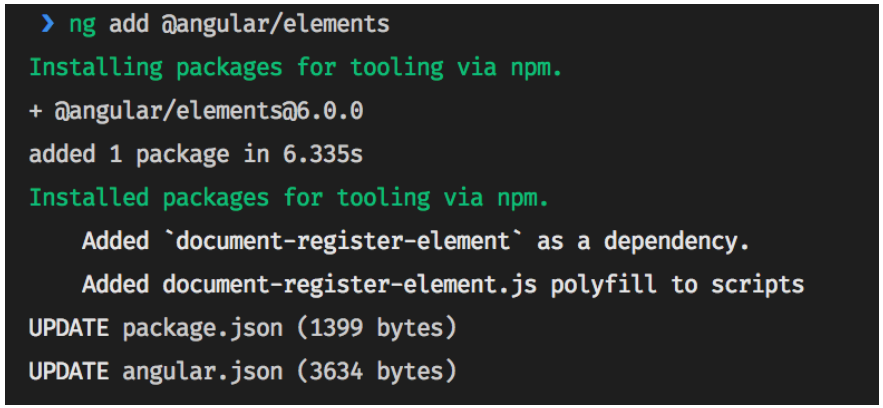
```
npm i -g @angular/cli
ng new elements-demo --prefix custom
```

We're not doing anything special yet, so all the normal `ng new` parameters apply, you could add sass or anything to the mix, but here we'll stop at setting a custom prefix to, well, "*custom*".

2. Add elements & polyfill

In order to have elements functionality available we need the Angular library and a polyfill—with the new CLI it's just a matter of one(!) simple command:

```
ng add @angular/elements
```



Just type a command and a machine is doing all the work—isn't this awesome?!

3. Create a component

Let's create one with *Input* and *Output* to see how they translate to custom elements that are understood by browsers:

```
ng g component button --inline-style --inline-template -v
Native
```

We use `ViewEncapsulation.Native` so that the styles are bundled with the template and the component's class into one file.

After adding some style & template our `button.component.ts` looks like this:

```
@Component({
  selector: 'custom-button',
  template: `<button (click)="handleClick()">{{label}}</button>`,
  styles: [`
    button {
      border: solid 3px;
      padding: 8px 10px;
      background: #bada55;
      font-size: 20px;
    }
  `],
  encapsulation: ViewEncapsulation.Native
})
export class ButtonComponent {
  @Input() label = 'default label';
  @Output() action = new EventEmitter<number>();
  private clicksCt = 0;

  handleClick() {
    this.clicksCt++;
    this.action.emit(this.clicksCt);
  }
}
```

You can copy this code from: https://gist.github.com/sulco/f1e69d0e453bd9c753639454af2bc099

4. Registering component in NgModule

This is the vital part: we use the Angular's `createCustomElement` function to create a class that can be used with browsers' native `customElements.define` functionality.

Angular documentation describes this best:

> `createCustomElement` *Builds a class that encapsulates the functionality of the provided component and uses the configuration information to provide more context to the class.* ***Takes the component factory's inputs and outputs to convert them to the proper custom element API and add hooks to input changes.***

> *The configuration's injector is the initial injector set on the class, and used by default for each created instance.This behavior can be overridden with the static property to affect all newly created instances, or as a constructor argument for one-off creations.*

What is also special about this module is that, since our `ButtonComponent` is not a part of any other component, and is also not a root of an Angular application, we need to specifically tell Angular to compile it: for this we put it on the `entryComponent` list.

We also need to tell Angular to use this module for bootstrapping, hence the `ngDoBootstrap` method.

Our **`app.module.ts`** should now look like this:

```
@NgModule({
  declarations: [ButtonComponent],
  imports: [BrowserModule],
  entryComponents: [ButtonComponent]
})
export class AppModule {
  constructor(private injector: Injector) {
    const customButton = createCustomElement(ButtonComponent, { injector });
    customElements.define('custom-button', customButton);
  }

  ngDoBootstrap() {}
}
```

source: https://github.com/sulco/angular-6-web-components/blob/master/src/app/app.module.ts

5. Build, optimize and run the code

To try our component out we will serve a simple html with `http-server`, so let's add it:

```
npm i -D http-server
```

In order to build we will use a standard `ng build` command, but since it outputs 4 files (`runtime.js` , `scripts.js` , `polyfills.js` and `main.js`) and we'd like to distribute our component as a single js file, we need to turn hashing file names off to know what are the names of files to manually concatenate in a moment. Let's modify the "build" script in `package.json` and add "package" and "serve" entries:

```
"build": "ng build --prod --output-hashing=none",
"package":
    "cat dist/elements-
demo/{runtime,polyfills,scripts,main}.js
     | gzip > elements.js.gz",
"serve": "http-server --gzip"
```

Now the sample `[projectFolder]/index.html` :

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <title>Custom Button Test Page</title>
  <script src="elements.js"></script>
</head>

<body>
  <custom-button label="First Value"></custom-button>

  <script>
    const button = document.querySelector('custom-button');
    button.addEventListener('action', (event) => {
      console.log(`"action" emitted: ${event.detail}`);
    })
    setTimeout(() => button.label = 'Second Value', 3000);
  </script>
</body>

</html>
```

source: https://github.com/sulco/angular-6-web-components/blob/master/index.html

And let's see this in action!

```
npm run build && npm run package
npm run serve
```

Epilogue

So what were the crucial things we did here? In summary we've:

- added elements-relared libraries with `ng add` command
- registered the Angular component as custom element in a module
- combined build artifacts into one file and gzipped them

Not that difficult, right? Personally, **I am extremely excited that this is such an easy process**—it will definitely encourage Angular devs that might have been a conservative in looking into custom elements, to just start playing with the tech, and soon, use it as a standard tool in their tool belts.

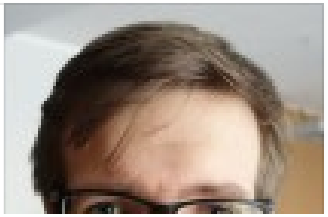
Oh, and btw. the resulting `elements.gz.js` weights **62kB**, which, considering that we have the full power of Angular framework inside it, is a pretty remarkable result!

As usual, you can browse the completed code on Github.

Did you learn something new? If so please:

→ **clap 👏 button** below so more people can see this
→ **follow me** on Twitter (@sulco) so you won't miss future posts!

Tomek Sułkowski (@sulco) | Twitter



The latest Tweets from Tomek Sułkowski (@sulco). #TypeScript · #JavaScript · @Angular trainer & Angular Tricity...

twitter.com