Elementos Angular para los Web Components

La industria web vive un momento de esplendor y le crecen los *frameworks* como hierbas primaverales. Pero el estándar HTML no se queda atrás y evoluciona hacia tecnologías potentes y genéricas. **Angular Elements** promueve la reutilización de código en distintos frameworks para que puedas usar tus componentes Angular en otros entornos.

Siendo como es Google una empresa *web first*, se esfuerzan en incorporar y adaptar de la mejor manera los estándares HTML a sus productos. Con el desarrollo de Angular siempre tuvieron la vista puesta en la tecnología de los **Web Components**. Buscando que los usos futuros del código se garantizasen más allá del framework de creación.

Partiendo del código tal como quedó en Internacionalización y puesta en producción. Al finalizar tendrás unos componentes que podrás utilizar fuera de Angular.

> Código asociado a este tutorial en *GitHub*: AcademiaBinaria/angular-boss

1. Componentes independientes del framework

> Hay lácteos que aguantan más que algunos frameworks.

Seguramente este no sea el caso de Angular, ni de otros frameworks de adopción masiva como React, Vue o Svelte. Todos ellos tienen un presente brillante y un futuro garantizado al plazo que la tecnología pueda vislumbrar. Pero más temprano que tarde otra tecnología o paradigma disruptivo los desplazará. O al menos los obligará a cambiar tanto que sean irreconocibles.

Para entonces, y también mientras tanto, nuestro código será cautivo del framework en el que nació. Pero eso cambiará con los **Web Components**.

1.1 Origen y potencial

Los **Web Components** son independientes de los *frameworks*. Esta es la idea clave; se pueden desarrollar con el estándar pelado de JavaScript o con cualquier framework moderno. Pero lo fundamental es que no exigen nada especial para ejecutarse. Esto permite la interoperabilidad y también extiende la vida útil de tus creaciones.

Usos posibles

Partiendo de dichas premisas es fácil entrever todo el potencial y casos dónde aplicarlos. Por ejemplo:

- Librerías de diseño *multiplataforma*.
- Migración paulatina de aplicaciones *legacy*.
- Integración dinámica en grandes soluciones *CMS*.
- Mejoras funcionales en aplicaciones *server side*.

1.2 Estándares y tecnología

Bajo el término Web Components se esconden diversas tecnologías. No todas ellas están al mismo nivel de aceptación, e incluso alguna no ha visto la luz, pero este es un esbozo de lo que tenemos:

- **Shadow DOM:** Manipulación de un árbol en memoria antes de aplicar sus cambios al verdadero.
- **HTML templates:** Fragmentos de HTML que no se utilizan en la carga de la página, pero que se pueden instanciar más adelante.
- **ES Modules:** Inclusión de documentos JS en forma de módulos de manera estándar y ágil.
- **Custom elements:** son etiquetas HTML con funciones encapsuladas, reutilizables y listas para usar en páginas y aplicaciones web.

El estándar:

> Los **Custom Web Elements** sólo requieren *HTML y JavaScript*.

La tecnología:

> *Angular Elements* empaqueta tus componentes como **Custom Web Elements**.

2. Desarrollo y despliegue con Angular

Un componente común de Angular

Partimos de un componente Angular normal y corriente. Un conversor (sí, ya sé que es sólo un miserable multiplicador) de monedas.

`libs\currency\src\lib\converter\converter.component.html`

```
<form>
  <label>Amount to convert: </label>
  <input name="amount"
         [(ngModel)]="amount"
         type="number"
         (change)="convert()" />
  <label>Converted amount: </label>
  <input name="convertedAmount"
         [(ngModel)]="convertedAmount"
         type="number"
         readonly />
</form>
```

`libs\currency\src\lib\converter\converter.component.ts`

```
@Component({
  selector: 'angular-boss-converter',
  templateUrl: './converter.component.html',
  styleUrls: ['./converter.component.css']
```

```
})
export class ConverterComponent implements OnInit {
  @Input() factor = 1.1;
  @Input() amount = 0;
  @Output() converted = new EventEmitter<number>();
  convertedAmount = 0;
  constructor() {}
  ngOnInit() {
    this.convert();
  }
  convert() {
    this.convertedAmount = this.amount * this.factor;
    this.converted.next(this.convertedAmount);
  }
}
```

El componente sigue siendo Angular

Es tan normal y corriente que puedo importar su módulo y usarlo en cualquier aplicación Angular.

`apps\warehouse\src\app\app.module.ts`

```
import { CurrencyModule } from '@angular-boss/currency';

@NgModule({
  imports: [
    CurrencyModule
  ],
})
export class AppModule {}
```

`apps\warehouse\src\app\app.component.html`

```
<angular-boss-converter amount="100"
                        factor="1.5"></angular-boss-converter>
```

2.1 Exponer los componentes

Pero ahora todo va a cambiar. Necesitamos un proyecto de exportación. Nada especial. Yo le pongo aquí el prefijo external para hacer hincapié en su función de exportación.

```
ng g @nrwl/angular:application external-currency
```

Si el componente a exportar ya ha sido probado, y debería, en este proyecto no necesitamos nada más que el módulo. Así que puedes borrar tranquilamente el `AppComponent`, e incluso el `index.html`.

Pongámonos con la exportación, para la cual hacen falta herramientas. Pues adelante con ellas. Incorporamos **Angular Elements**.

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

Ahora ya importamos el componente en el *array* `imports` y lo exportamos, pero no en `exports` ni tampoco en `bootstrap`. Vamos a incluirlo en `entryComponents`.

`apps\external-currency\src\app\app.module.ts`

```
import { ConverterComponent, CurrencyModule } from '@angular-boss/currency';
import { Injector, NgModule } from '@angular/core';
import { createCustomElement } from '@angular/elements';
import { BrowserModule } from '@angular/platform-browser';
import 'zone.js';
@NgModule({
  imports: [BrowserModule, CurrencyModule],
  entryComponents: [ConverterComponent]
})
export class AppModule {
  constructor(private injector: Injector) {}
  ngDoBootstrap() {
    const el = createCustomElement(ConverterComponent, {
      injector: this.injector
    });
    customElements.define('external-currency-converter', el);
  }
}
```

Los `entryComponents` son cargados por Angular de forma imperativa, sin necesidad de incluirlos en un *html*. Menos mal porque este proyecto ni siquiera usa el `index.hml`

ngDoBootstrap

Justamente la falta de componentes en el array `bootstrap` obliga a arrancar la aplicación mediante código. Esto lo hacemos programando en el *hook* `ngDoBootstrap()`, un método que será invocado al inicio y en el meteremos la lógica necesaria para definir el **Web Component**.

Son solamente dos instrucciones que usarás para cada componente que quieras transformar en un elemento.

```
  const el = createCustomElement(ConverterComponent, {
    injector: this.injector
  });
  customElements.define('external-currency-converter', el);
```

En la primera se crea el elemento mediante una función constructora que ofrece `@angular/elements`. Para ello necesita un puntero al componente original y el sistema de inyección de dependencias de Angular. (El cual a su vez reclamamos en el constructor del módulo).

La segunda instrucción ya no es propia de Angular. Simplemente se le asigna al elemento recién creado el *selector tag* por el que será conocido.

> Atención a la importación de `zone.js`.

Es necesaria para poder usar la detección de cambios en aplicaciones no Angular.

2.2 Compilación y despliegue

A partir de aquí se acabó el trabajo de programación. Y toca el más tedioso de operaciones de compilación y distribución.

Afortunadamente podemos aplicar la magia de `@angular/elements` y utilidades como `ngx-build-plus` para compilarlo como un Web Component.

Agregar herramientas de ayuda

Lo primero es instalar una herramienta que mejora el proceso de compilación del CLI.

```
ng add ngx-build-plus --project external-currency
```

Esencialmente usa *webpack* para empaquetar el resultado de la compilación `build` nativa del CLI. Pero no necesitas profundizar en su interior para usarla en el día día.

Polyfills

Con el Internet Explorer hemos topado. Y otros navegadores antiguos, claro. El caso es que para que entiendan este nuevo estándar, tenemos que incluir una serie de arreglos llamados *polyfills*, que rellenan los huecos de incomprensión de los viejos exploradores. Así que para garantizar la compatibilidad en todos los navegadores instalamos todo lo necesario usando `ngx-build-plus`.

```
ng g ngx-build-plus:wc-polyfill --project external-currency
```

Ojo que en algunas versiones se equivoca en la generación de los *assets* y hay que moverlos a mano.

Compilación

Ya solo falta lanzar el comando de compilación `ng build`. Pero para ajustar un poco más el resultado te propongo que deshabilites el nombrado con *hash* y por supuesto que uses algunos parámetros que ha incluido `ngx-build-plus`.

Nombres legibles: `"outputHashing": "none",`

Generación: `ng build --prod --single-bundle --project external-currency`

3. Consumo en HTML

Ya está, el resultado se podrá utilizar en cualquier aplicación HTML. Por ejemplo en una página prácticamente vacía y completamente estándar.

`apps\vanilla\index.html`

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Vanilla Currency</title>
    <base href="/apps/vanilla/" />
    <meta name="viewport"
          content="width=device-width, initial-scale=1" />
    <link rel="icon"
          type="image/x-icon"
          href="favicon.ico" />
  </head>
  <body>
    <h2>Convert money:</h2>
    <external-currency-converter amount="15"></external-currency-converter>
  </body>
</html>
```

¿Que no funciona? Obvio, el navegador no entiende el *tag* `external-currency-converter`. Necesitamos el JavaScript de la compilación anterior

3.1 Copiar

Copia los dos archivos generados en la carpeta de distribución

```
dist\apps\external-currency
```

Y pégalos al lado del html que quieras. Por ejemplo:

```
apps\vanilla\
```

3.2 Importar

Son dos porque hay versión clásica y modular. Importa la que necesites o ambas, y ya está.

```
  <script src="main-es2015.js"
          type="module"></script>
  <script src="main-es5.js"
          nomodule
          defer></script>
```

> Una web HTML pura, mostrando un componente creado en Angular

Ahora ya tienes código creado en Angular pero que puedes integrar en cualquier web. Todo este proceso aún puede resultar tedioso, pero es el futuro. Continúa tu formación avanzada para crear aplicaciones Angular fijándote en el Angular Blueprint creado por la iniciativa Angular.Builders y verás como aprendes a programar grandes aplicaciones con Angular.

> Aprender, programar, disfrutar, repetir. -- *Saludos, Alberto Basalo*