WebComponents

… EN DESARROLLO …

Introducción

WebComponents es el nombre por el que se conoce a un conjunto de características relacionadas con HTML, CSS y Javascript, mediante las cuales se pueden crear elementos mantenibles, reutilizables y encapsulados llamados **componentes**, sin que sea necesario utilizar herramientas externas, librerías o frameworks.

Una de las ventajas de los WebComponents es que son **fáciles de reutilizar**: A menudo, creamos elementos o partes que se repiten en nuestra web. Los WebComponents nos permiten reutilizar ciertas partes e incluirlas mediante una **etiqueta HTML personalizada**. Una forma cómoda y sencilla de reutilizar elementos web.

Además, son **fáciles de mantener**. A medida que escribimos código en una web, la cantidad de líneas de código se hace mayor y cada vez es más complicado con un enfoque global. Necesitamos una forma de poder enfocarnos en una característica concreta, que sólo tenga HTML, CSS y Javascript que influya a dicha característica específica.

Otra de las ventajas que proporcionan los WebComponents es su **encapsulación**. Necesitamos una forma de crear elementos **aislados** de otros. Por varias razones. Evitar cambiar su contenido por error, evitar colisiones de CSS de elementos que se llaman igual, etc. Se trata de traer una característica muy popular y deseada en el mundo de la programación, que facilita la labor de los desarrolladores y hace más fácil la reutilización en aplicaciones muy grandes.

Los elementos o características nativas que forman parte de los **WebComponents** son:

- **Custom Elements** o elementos personalizados son una de las características principales que forman los WebComponents. Con ellos, se nos permite crear nuestras propias etiquetas HTML, pudiendo dotarlas de su propia funcionalidad, marcado HTML o estilo CSS.

- **Templates** o plantillas. Se trata de una etiqueta nativa de HTML que nos permite crear contenido inerte en una página, esto es, contenido HTML que no se procesará por el navegador, y que permanecerá «muerto» hasta que lo clonemos mediante Javascript, y creemos nuevos elementos a partir de él.

- **Shadow DOM**. Probablemente, una de las características más interesantes (y complejas) de WebComponents. Esta característica opcional se basa en crear un DOM (estructura de elementos HTML) particular en un elemento HTML. De esta forma, además del DOM global del documento que normalmente utilizamos, tenemos uno particular. Es una excelente forma de crear estructuras con estilos CSS locales, que sólo repercutan en dicha estructura, y en la que nuestro CSS no afecte al CSS exterior, ni desde el CSS exterior afecta al CSS interior.

- **Módulos ES (ESModules o ESM)** son otra de las características que hace posible que existan los WebComponents. Se trata de un estándar de Javascript, que permite organizar elementos de nuestro código Javascript (constantes, funciones, clases, etc…) en módulos y exportarlos, para ponerlos a disposición de otro archivo Javascript que quiera importarlos.

Etiquetas HTML personalizadas

Los **Custom Elements** son una de las características que forman los WebComponents mediante los cuales podemos crear nuestras propias etiquetas HTML de forma nativa, dotándolos de su propio marcado, funcionalidad y/o estilo.

Nombres de las etiquetas HTML

El estándar de HTML5 define que las etiquetas HTML oficiales deben estar formadas por una sola palabra, mientras que los **Custom Elements** (nuestras propias etiquetas HTML) deben estar formadas de al menos **2 palabras, separadas por al menos un guión y en minúsculas** (pueden tener varios guiones). Si en un futuro se añaden nuevas etiquetas oficiales, no coincidarán con ninguna etiqueta personalizada.

Por tanto, por ejemplo las etiquetas `<app-component>` o `<app-name-component>` son correctas mientras que las etiquetas `<element>`, `<AppComponent>` no son correctas.

⚠️ **NOTA**: una buena práctica es utilizar la primera palabra como *namespace* del componente de forma que así se evitan colisiones con otros elementos con nombre similar.

La forma más básica de crear una etiqueta personalizada HTML es simplemente escribirla. El navegador la tratará como si fuera un elemento `<span>`.

👀 Por defecto se trata de un elemento en línea, por lo que hay que añadirle un *display* diferente y darle dimensiones si no se le añade contenido.

```
<app-element>
  Contenido de nuestro elemento personalizado
</app-element>

<element>
  Contenido de una etiqueta que no cumple el estándar
</element>
```

```
document.querySelector("app-element").constructor.name // => "HTMLElement"
document.querySelector("element").constructor.name // => "HTMLUnknownElement"
```

Cuando la etiqueta personalizada es correcta y se trata efectivamente de un **Custom Element**, el constructor es **HTMLElement**. Cuando la etiqueta no está bien formada, el navegador desconoce de que elemento se trata y por tanto muestra **HTMLUnknownElement**.

Crear un "Custom Element"

Los **WebComponents** se programan a través de una clase Javascript. De esta forma pasan de comportarse como una simple etiqueta `<span>` a tener funcionalidad.

```
// Se utiliza 'PascalCase' para nombrar la clase en relación a la etiqueta
class AppElement extends HTMLElement {
  constructor() {
    // Si se define el constructor (no es necesario) hay que llamar al constructor padre
    super();
    console.log("Inicializado...");
  }
```

```
}

/* Le indicamos al navegador la asociación entre el nombre de la etiqueta y
la clase que implementa la funcionalidad */
customElements.define("app-element", AppElement);
```

La clase JavaScript creada extiende de *HTMLElement*, una clase que es la base de cualquier elemento HTML, por lo que al heredarla nuestra etiqueta personalizada va a tener todas las características que tenga un elemento HTML estándar.

```
// Ahora que existe la clase JavaScript, devuelve el nombre de la clase
document.querySelector("app-element").constructor.name // => "AppElement"
```

Esta clase JavaScript, como norma general se guarda en un fichero '.js' con el nombre del componente y dentro de una carpeta donde guardaremos todos los componentes, como por ejemplo `components\AppElement.js`.

Cargar componentes desde el HTML

Una forma de cargar el componente sería cargar el fichero '.js' del componente en el documento HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="./components/AppElement.js"></script>
</head>
<body>
    <app-element></app-element>
</body>
</html>
```

Cargar componentes desde JavaScript

Sin embargo, podríamos preferir cargar el **WebComponent** desde Javascript en lugar de HTML, ya que puede ser más versátil y flexible en ciertas situaciones. Por ejemplo, si tenemos un gran número de componentes quizás podemos preferir centralizar los import en un fichero central de Javascript, que pueda ser reutilizado, en lugar de tener varias etiquetas `<script>` en varios archivos HTML diferentes.

En este caso tenemos un fichero '.js' donde centralizar todos los imports y que podemos llamar por ejemplo `index.js`:

```
// Fichero 'index.js'
import "./components/AppElement.js";
```

Este fichero lo cargaremos en el documento html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script type="module" src="/js/index.js"></script>
</head>
<body>
    <app-element></app-element>
```

```
  </body>
  </html>
```

⚠ **NOTA**: hay que añadir el `type="module"` **obligatoriamente** o si no veremos por consola un error de tipo `Uncaught SyntaxError: Cannot use import statement outside a module`.

Librerias de WebComponents

1. https://stenciljs.com/
2. https://lit.dev/
3. https://www.fast.design/
4. https://lwc.dev/
5. https://www.solidjs.com/

Enlaces de interés

- https://lenguajejs.com/webcomponents/
- https://developer.mozilla.org/es/docs/Web/API/Web_components
- https://www.webcomponents.org/introduction
- https://open-wc.org/
- https://component.gallery/design-systems
- https://kinsta.com/blog/web-components/
- https://dev.to/steveblue/the-state-of-web-components-in-2022-1ip3
- https://jsfiddle.net/

Licencia