Web Components, un nuevo estándar para el desarrollo de aplicaciones HTML5

Consideraciones para su implementación

Tesina de grado, julio 2015

Autor

A.P.U. Federico E. González

Director

Lic. Guillermo Feierherd

Universidad Nacional de la Patagonia, San Juan Bosco

Facultad de Ingeniería – Ushuaia

Tierra del Fuego, Argentina

Agradecimientos

Hay dos cosas que -en mi opinión- caracterizan desde siempre al desarrollo web en términos globales: la diversidad tecnológica y la apertura de muchísima gente para compartir su conocimiento. A todas esas personas, a quienes conozco y a quienes no, dedico este trabajo.

En especial a los que con sus consejos, aportes y correcciones hicieron de este texto algo mejor: Leonel Viera, Ezequiel González Rial, Lisandro Delía, Ben Thomas, Eric Bidelman, Addy Osmani, Zeno Rocha, Cristian Lorenzo y Leandro Loiacono.

A la Municipalidad de Ushuaia y a Parques Nacionales por darme acceso a los datos para construir la información del capítulo de análisis.

A Guillermo Feierherd, que siempre ha estado atento. Antes, durante y después.

Y principalmente a mi familia: los que están aquí y los que están allá; por la infinita paciencia, gracias totales.

Índice de contenidos

Introducción

2014 fue un año histórico. La población mundial llegó a 7,3 mil millones de habitantes; y también, en en 2014, se conectaron a internet más de 7,4 mil millones de dispositivos.

2012 también fue clave, ya que supuso un punto de inflexión. En ese año, por primera vez desde su invención, las ventas de PCs fueron inferiores al año anterior. Desde entonces esas ventas no han parado de caer. Estamos sin dudas frente a una nueva era, la de los dispositivos móviles -que seguramente será en breve atropellada por la "Internet de las cosas".

Es indispensable entonces, al desarrollar aplicaciones web, garantizar una correcta funcionalidad frente a cualquier dispositivo, ya sea una PC, notebook, tablet, teléfono, reloj o un par de anteojos[1].

El desarrollo Web evoluciona constantemente. Año tras año, las aplicaciones basadas en Internet ganan terreno sobre las demás. Hoy es común usar un Browser como plataforma para la ejecución de herramientas de trabajo y ocio.

Los lenguajes asociados al desarrollo de aplicaciones Web también evolucionan y empujan a los Browsers a soportar nuevas funciones. El World Wide Web Consortium (W3C) es el organismo internacional encargado de desarrollar las recomendaciones para los estándares abiertos que aseguren un crecimiento ordenado y a largo plazo para la Web. Las principales empresas de tecnología asociadas a Internet como Google, Apple, Microsoft, Mozilla, Opera y Adobe, entre otras, participan activamente en el W3C para la construcción de tales recomendaciones[2].

La velocidad con que evolucionaron las aplicaciones Web no ha sido acompañada por las capacidades de los Browsers actuales.

Así, en desarrollo Web es común el uso de artilugios (polyfills) para simular capacidades no previstas por los Browsers o para garantizar que una función se desempeñe correctamente en distintas plataformas.

[1] Por razones de disponibilidad tecnológica, en esta tesina se decidió acotar la investigación a PCs, notebooks, tablets y teléfonos, que a la fecha representan el 99% de los dispositivos conectados a la Web.

[2] A mayo de 2015 el W3C cuenta con 382 miembros globales, entre los que se encuentran las empresas más importantes del sector <http://www.w3.org/Consortium/Member/List>

Con el tiempo el código de las aplicaciones Web se ha vuelto cada vez más complejo, con "planes de contingencia" para cada situación, llevando a los desarrolladores a perder una parte importante de su tiempo en pruebas para detectar errores en cada una de las plataformas, versiones de Browsers y dispositivos… o simplemente evitando dichas pruebas y entregando software que no siempre funciona como se espera bajo ciertas condiciones no validadas.

Con la versión 5 del estándar que define al principal lenguaje de Internet, HTML, se vislumbra una promesa de cambios y un gran consenso internacional de haber encontrado el camino correcto.

Entre las nuevas recomendaciones que el W3C promueve para la total implementación de HTML5, los Web Components han llamado la atención por ofrecer una solución a problemas conocidos del desarrollo Web: la semántica, encapsulación, extensibilidad, reutilización y validación.

El W3C organiza a los Web Components en cuatro partes: Templates, Shadow DOM, Custom Elements y HTML Imports.

Los **Templates** (plantillas, en castellano) facilitan la generación de bloques de código reutilizables, que una vez definidos, pueden incluirse en distintas situaciones ya sea en tiempo de diseño como en ejecución.

El Modelo de Objetos del Documento (DOM) representa la estructura del HTML y proporciona una interfaz que permite acceder a los contenidos y manipularlos. El **Shadow DOM** facilita la abstracción permitiendo ocultar bloques de código en el HTML y evitando que un componente pueda invadir la declaración de otros.

Los **Custom Elements** (elementos personalizados, en castellano) permiten extender el lenguaje de marcas HTML para crear nuevos elementos reconocibles por el Browser.

Para cargar librerías externas hoy en día, se utilizan etiquetas separadas por cada tecnología, ya sea JavaScript, CSS o Web Fonts, lo que genera una complejidad innecesaria para el manejo de dependencias. **HTML Imports** permite cargar recursos agregándolos a un archivo HTML y luego, sólo incluir ese HTML principal. También reconoce recursos previamente cargados para evitar duplicidad.

Objetivos de esta tesina

En este trabajo se espera establecer una comparación entre cómo se utilizan los lenguajes HTML, CSS y JavaScript para el desarrollo de una aplicación web actual y cómo se podrán utilizar estas tecnologías a partir de la implementación del estándar de Web Components.

Para ello, se proponen los siguientes objetivos secundarios:

Analizar las condiciones de compatibilidad de los principales Browsers de la actualidad, para estimar cómo se espera que evolucione cada uno de cara al estándar.

Generar una serie de recomendaciones que resuman las buenas prácticas para incorporar Web Components en lo inmediato.

Desarrollar cinco Web Components que quedarán publicados mediante licencia Open Source dentro de un repositorio público en Internet.

Requerimientos técnicos y marco legal

Consideraciones técnicas

Como podrá conocerse a lo largo de esta tesina, el estándar de Web Components se encuentra en proceso de desarrollo y muchas de sus funciones aún no están disponibles de forma nativa en la mayoría de los Browsers actuales.

Los ejemplos presentados en el capítulo “Introducción al estándar de Web Components” sólo funcionarán utilizando Google Chrome con versión 36 o superior.

En el capítulo “Librerías para Polyfills y análisis de Browsers” se introduce el concepto de Polyfill y los ejemplos presentados a partir de ahí, deberían poder utilizarse correctamente en cualquier Browser actual.

Repositorio de ejemplos

Toda la información producida en esta tesina (y la tesina en sí misma) ha sido publicada en el repositorio GitHub http://fedegonzal.github.io/Web-Components/ desde donde pueden descargarse tanto el código de todos los ejemplos, como el texto editable de este trabajo.

Además se incluyeron links a todos los ejemplos presentados a lo largo del trabajo, para testearlos on-line, sin que sea necesario descargarlos.

Consideraciones legales

Este trabajo tiene fines pedagógicos y académicos, toda la información expuesta tanto en el documento como en los códigos de ejemplo puede ser compartida, adaptada y utilizada -incluso comercialmente por terceros- siempre que se respeten las demás licencias que dependen de las librerías vinculadas.

El código está licenciado bajo BSD[3] y la documentación bajo Creative Commons Reconocimiento 4.0 Internacional[4].

Quien requiera utilizar en su totalidad o en parte a esta tesina, debe citar a su autor e indicar los cambios realizados -en caso de haberlos hecho- mencionando la siguiente información:

Autor: Federico González Brizzio
Email: fedegonzal@gmail.com
GitHub: http://fedegonzal.github.io/Web-Components/

[3] OPEN SOURCE INITIATIVE. *The BSD 3-Clause License* <http://opensource.org/licenses/BSD-3-Clause> [Consulta: 10 Junio 2015]

[4] CREATIVE COMMONS. *Creative Commons License 4.0* <https://creativecommons.org/licenses/by/4.0/> [Consulta: 10 Junio 2015]

Metodología de trabajo

Se desarrollan tablas comparativas analizando las distintas capacidades de los Browsers (y sus diferentes versiones) a fin de sistematizar qué funciones de la tecnología Web Components están disponibles en la actualidad y qué Polyfills (alternativas de código) pueden utilizarse en Browsers anteriores o sin soporte, según cada función.

De esta manera se presentan una serie de recomendaciones prácticas, para que los desarrolladores de aplicaciones Web puedan incorporar en sus Workflows (flujos/mecanismos de trabajo) a fin de simplificar las tareas y minimizar los errores para distintos ambientes en tiempo de ejecución.

También se codificaron y documentaron cinco Web Components para ejemplificar los distintos problemas que puedan surgir al implementar esta nueva tecnología, al momento de desarrollar aplicaciones Web basadas en componentes.

Herramientas utilizadas

Documentación y codificación

Google Drive fue de gran utilidad para la edición y revisión, ya que permite múltiples usuarios sobre un mismo documento sin necesidad de enviar actualizaciones por email con copias y versiones confusas.

La edición de los detalles finales se realizó con Microsoft Word, para tener un mayor control sobre la numeración de páginas y el índice.

La portada se diseñó con Microsoft Powerpoint y se exportó a PDF.

Finalmente se utilizó la aplicación web Unir PDF, disponible en http://smallpdf.com/es/unir-pdf con la que se unieron los PDF de portada y contenido, para generar la versión impresa definitiva.

Free Formater es una web http://www.freeformatter.com/ en la que puede pegarse un código HTML (también CSS y JS) y obtener su versión de sintaxis resaltada para incorporar a la edición de un documento. Fue de gran utilidad para facilitar la lectura de los ejemplos de código en la tesina.

Los bocetos del último ejemplo fueron realizados a mano alzada, pero podría haberse utilizado Pencil, una herramienta gratuita y multiplataforma ideal para ese propósito http://pencil.evolus.vn/

La edición de imágenes en general no requirió más que capturar pantallas, pero en algunos casos se utilizó Adobe Fireworks para pequeñas ediciones y recortes visuales.

Tanto el código de los ejemplos, como el documento de la tesina en sí mismo, se publicaron en GitHub https://github.com/ una plataforma que provee repositorios gratuitos para proyectos opensource y es utilizada por 10 millones de desarrolladores alrededor del mundo. GIT también es un protocolo para versionado de código, incluido entre los beneficios de GitHub.

Los códigos HTML, JavaScript y CSS se editaron indistintamente con Sublime Text, un software gratuito multiplataforma http://www.sublimetext.com/ y con Brackets http://brackets.io/ también multiplataforma pero además opensource.

StackOverflow http://stackoverflow.com/ es un sitio web donde se puede preguntar públicamente sobre temas técnicos y una importante comunidad global responde, bajo un estricto sistema semiautomático de curadurías. Fue de gran importancia para resolver dudas y problemas puntuales.

Ejecución y testeo de los ejemplos

NPM https://www.npmjs.com/ es un manejador de paquetes que incluye una extensa oferta de herramientas gratuitas de gran ayuda para desarrolladores. Entre los paquetes utilizados se puede mencionar:

- Browser Sync y Life Reload monitorean cambios en el código fuente y actualizan automáticamente los Browsers que tengan esos archivos abiertos.
- Grunt es una herramienta de automatización que se describe a continuación.
- http-server inicia un servidor básico ideal para desarrollo FrontEnd.
- json-server es un servidor ficticio de APIs REST/JSON para desarrollo local.
- Minifier comprime código CSS y JavaScript y se describe a continuación.

Otra herramienta clave para testeo y depuración es el Browser en sí mismo, con su consola para desarrolladores. Se utilizó Google Chrome 43, Mozilla Firefox 35 y Microsoft Internet Explorer 11.

Sauce Labs https://saucelabs.com/ brinda servicios de testing automatizado y tiene productos gratuitos para proyectos opensource. Utiliza Selenium y Q-Unit, dos reconocidas herramientas

de testeo. Provee máquinas virtuales y acceso a más de 500 Browsers / Sistemas Opertavios para chequear la aplicación.

Sistematización y distribución

Grunt http://gruntjs.com/ permite la automatización de tareas (task runner) con el objetivo de sistematizar actividades típicas previas a la puesta en producción de proyectos.

Vulcanize https://github.com/polymer/vulcanize concatena dependencias con HTML Imports facilitando preparar Web Components para su distribución.

Minifier es un paquete de NPM que puede utilizarse con Grunt y permite minimizar código CSS y JS para evitar consumos innecesarios de red en tiempo de ejecución.

Bower http://bower.io/ es otro manejador de paquetes muy usado por desarrolladores de aplicaciones web, que facilita la descarga de librerías y sus dependencias.

Resumen de capítulos

En el capítulo de **antecedentes** se expone sobre cómo los lenguajes de programación evolucionaron hacia el uso de componentes y por qué el HTML ha encontrado obstáculos en ese camino.

En la **Introducción al estándar de Web Components** se presenta la situación actual de este nuevo conjunto de herramientas promovidas por el W3C y se explica en detalle cada uno de sus pilares, concretamente: Templates, Custom Elements, Shadow DOM e Imports. También se incluye un ejemplo técnico de cómo implementar Web Components bajo circunstancias ideales: esto es, si todos los Browsers soportaran e interpretaran de igual manera el estándar.

Dicho ejemplo se utiliza reiteradas veces a lo largo del trabajo, haciéndolo evolucionar a medida que se presentan herramientas o para ejemplificar problemas puntuales.

El capítulo **estado del arte** introduce los desafíos actuales que implica la implementación del estándar para desarrollar aplicaciones web y presenta una **comparativa** de las capacidades que tienen los distintos Browsers para interpretar dicho estándar, exponiendo las limitaciones y problemas actuales.

En el capítulo **librerías para Polyfills y análisis de Browsers** se explica el concepto de Polyfill, se presentan herramientas para facilitar su aplicación y se realiza un análisis detallado para Argentina y Ushuaia, con recomendaciones para implementar Web Components en el corto, mediano y largo plazo.

El capítulo **guía básica para el desarrollo de Web Components** presenta dos ejemplos completos donde paso a paso se profundiza en el desarrollo de Web Components con las librerías de Polyfills presentadas en el capítulo anterior.

En el capítulo **ejemplos para el desarrollo de componentes** se presentan cinco Web Components desarrollados y testeados siguiendo las recomendaciones mencionadas y que han sido publicados mediante licencia Open Source en un repositorio público en Internet.

El capítulo **conclusiones y recomendaciones** resume la experiencia alcanzada con el trabajo y se sistematizan una serie de buenas prácticas para incorporar Web Components en lo inmediato.

Antecedentes

Ya en 1968, durante la conferencia "Software Engineering, Report on a conference sponsored by the NATO Science Committee" que se llevó a cabo en Garmisch, Alemania, Douglas McIlroy hablaba de la programación orientada a componentes "Creo que la industria del software es débil y que uno de los aspectos de esta debilidad es la ausencia de una subindustria de componentes" y también hablaba de los *component's markets* (mercados de componentes) "El comprador de un componente elegirá el que se ajuste a sus necesidades exactas. Se consultará a un catálogo que ofrece rutinas en diversos grados de precisión, robustez y rendimiento espacio-tiempo [...] En pocas palabras, debe ser capaz de considerar a los componentes como cajas negras."[5]

Diversos lenguajes ya consolidados como C++, Delphi, Java, .NET y muchos otros, hacen un uso intensivo y basan su programación en el uso de componentes.

Características esperables de los componentes son la abstracción, el empaquetado, la modularidad y el acoplamiento, la extensibilidad y por supuesto, la semántica y la reusabilidad.

El desarrollo web se encuentra en un estado de transición y ha estado inmóvil durante algún tiempo. Los Browsers fueron diseñados originalmente para ver documentos, pero lentamente se han ido moviendo para convertirse en plataformas para correr aplicaciones, transformando radicalmente el paisaje conocido alrededor del software. Nunca fue tan fácil poner en producción una nueva versión de una aplicación: un desarrollador pone en línea su nuevo código y los usuarios finales ya lo tienen a disposición. (Overson & Strimpel, 2015)

HTML, CSS y JavaScript son los pilares para cualquier aplicación o sitio web. Todos los Browsers actuales interpretan y reconocen esos tres lenguajes, es natural entonces que cualquier análisis de la actualidad en desarrollo de aplicaciones web comience por aquí.

Se trata de tres lenguajes sencillos, simples y extremadamente poderosos.

HTML es un lenguaje de marcas e interpretado, que puede describir semánticamente la estructura de una página web haciendo foco en sus contenidos -ya sean textos, imágenes, videos, hipervínculos, etc. Fue inventado en 1990 por Tim Berners-Lee, hoy reconocido mundialmente como el padre de la WWW.

[5] MCILROY, DOUGLAS (1968). Discurso completo: *Mass Produced Software Components* <http://www.cs.dartmouth.edu/~doug/components.txt> [Consulta: 10 Junio 2015]

CSS es un lenguaje orientado a la presentación y su propósito es definir “cómo se verá” un contenido escrito con marcas HTML al momento de representarlo en cualquier dispositivo, como puede ser una computadora, teléfono, impresora, reloj, etc. CSS fue desarrollado en 1996 por Håkon Wium Lie y Bert Bos para el W3C, que por esos años recién daba sus primeros pasos de la mano de Tim Berners-Lee.

HTML y CSS actualmente son mantenidos por el consorcio internacional W3C.

“JavaScript es uno de los lenguajes de programación más populares del mundo, y a la vez, el menos comprendido.”[6] Fue creado en 1995 por Brendan Eich en Netscape, lanzado bajo el nombre de LiveScript y renombrado a JavaScript para aprovechar las oportunidades de marketing que Java imponía por esos días, aunque ambos en realidad tengan muy poco en común. Se trata de un lenguaje con tipos de datos dinámicos y orientado a objetos, su entorno de ejecución más común es el Browser, aunque puede encontrarselo embebido en dispositivos de hardware y, recientemente, como lenguaje en servidores NodeJS y más.

Una característica común de estos tres lenguajes es que se ejecutan en un Browser y si bien los lenguajes han ido evolucionando de manera sostenida, dependen de que los navegadores integren las nuevas capacidades en sus próximas versiones (porque son los Browsers quienes interpretan a los lenguajes) y si los usuarios finales no actualizan su navegador, no dispondrán de tales mejoras.

Sumado a esto, las empresas que desarrollan navegadores lanzan sus productos con distintos grados de implementación de las tecnologías asociadas. A la vez, cada usuario puede elegir qué Browser (y qué versión) utilizar, con cuál sistema operativo y con qué tipo de dispositivo, lo que redunda en un importante ecosistema de diferencias e inconsistencias.

[6] CROCKFORD, DOUGLAS (2001). *JavaScript: The World's Most Misunderstood Programming Language* <http://javascript.crockford.com/javascript.html> [Consulta: 10 Junio 2015]

Introducción al estándar de Web Components

“No creo estar exagerando al decir que la especificación de Web Components propone el cambio más radical desde la creación del HTML, hace ya 20 años.”[7]

Así es, en la web se esperan nuevos vientos y con ellos, nuevas oportunidades.

Al ser HTML un lenguaje de marcas, debería ser fácil de extender. Así, si existe la marca `<img>` que refiere a una imagen, sería deseable pensar que un desarrollador pueda generar sus propias marcas, por ejemplo `<google-map>` para referir a un mapa interactivo con el servicio de Google.

HTML ha sido pensado extensible, pero los Browsers no pueden interpretar marcas que no conocen. Los Web Components son la respuesta que el W3C ha desarrollado en la versión 5 de HTML para resolver este problema.

Los Web Components surgieron en 2011 como una propuesta de Google y con el tiempo se han abierto camino en el W3C, quien comenzó a trabajar en una especificación formal. Después de acuerdos y desacuerdos, los desarrolladores de Browsers han comenzando a adoptar la nueva tecnología. Ya se habla de un estándar, si bien formalmente aún no lo es.

En pocas palabras: los Web Components permiten establecer marcas, funciones y estilos para crear nuevos elementos HTML. Algo muy importante es que esos nuevos elementos son totalmente encapsulados, de manera que todo su HTML y CSS no pueda colisionar con otros elementos (nativos o nuevos) gracias a que sólo conocen el ámbito en donde fueron definidos. Por lo tanto, los estilos definidos siempre se comportan como se espera y el HTML está protegido frente a código JavaScript externo al componente. (Dodson, 2014)

Uno de los principales problemas en el desarrollo de componentes HTML es que los elementos utilizados para programar tales componentes son parte del mismo DOM (Document Object Model) y, por lo tanto, susceptibles de generar conflictos entre sí, ya sea en JavaScript como en CSS.

Se hace mención por ejemplo, a los conflictos de herencia las como reglas CSS, que aplicadas a un elemento superior afectan a los inferiores o incluso comprometen formatos y comportamientos en cualquier parte del árbol del HTML: el DOM.

[7] GASSTON, PETER (2013). *The Modern Web*. No Starch Press, Cap. 11, Pág. 192

Otro de los problemas está asociado al conflicto de nombres, donde la misma clase o ID es utilizada, sin saberlo, en diferentes partes del código, haciendo que reglas declaradas intencionalmente para cierto elemento, afecten sin quererlo a otros y los perjudiquen.

La mejor manera de evitar estos conflictos es separar el componente del resto del DOM, previniendo cualquier herencia o solapamiento en la definición con otros. Esta técnica es conocida como encapsulación y es fundamental en la programación orientada a objetos.

Los Web Components llevan la encapsulación hacia el DOM del HTML, permitiendo crear elementos que aparecen cuando se genera la página y no en el DOM en sí mismo. Así, cada componente tiene su propio DOM.

Los Web Components concretan y formalizan la posibilidad de crear widgets que puedan ser reutilizados en diferentes situaciones, sin tener que preocuparse de los conflictos que pudieran generarse con CSS o JavaScript, ya que se trata de espacios de dominio diferentes. (Gasston, 2013)

Lenguajes como Java, PHP, Ruby, .NET y muchas tecnologías más asociadas al desarrollo de aplicaciones Web hacen uso de componentes -cada una a su manera- y permiten definir marcas para simplificar la codificación, pero todas tienen el mismo problema: esas marcas son propias del lenguaje o de alguna herramienta agregada y no son parte de un estándar que permita reutilizar los componentes más allá de la tecnología con la que se esté trabajando.

Los Web Components proponen una visión distinta porque son interoperables, esto quiere decir que no dependen de tecnologías particulares para funcionar: son parte de un estándar que funciona en todos los Browsers: el HTML.

Los Web Components se resuelven durante la ejecución en el navegador del cliente (desde el punto de vista cliente-servidor) y no durante la ejecución en el servidor.

Un ejemplo simple para comprender los Web Components es pensar en la etiqueta `<img>` que está presente desde los inicios del HTML.

La etiqueta `<img>` permite incorporar una imagen en un documento HTML, esto no quiere decir que la imagen en sí misma forme parte del HTML, sino que simplemente está referenciada en el código y el Browser sabe interpretar dicha marca y mostrar, por ejemplo, una foto.

En 2005 ya existía YouTube, sin embargo, HTML no preveía ninguna marca `<video>` que facilitase la incorporación de ese tipo de contenidos en una página web. Los creadores de

YouTube resolvieron su problema creando un componente con tecnología Flash, que permitía reproducir videos dentro de un sitio web, siempre que el usuario final tuviera instalado el *Plugin* correspondiente.

Diez años después, con el avance de los teléfonos y tablets capaces de navegar en internet, la tecnología Flash quedó fuera de batalla y en HTML5 se incorporó la marca `<video>` que le permitió a YouTube aprovechar la siguiente era y todo el potencial de los nuevos dispositivos para seguir captando usuarios.

La vida antes de los Web Components

A continuación se presenta un ejemplo clásico, que se irá extendiendo y mejorando a lo largo de esta tesina: un reproductor de diapositivas.

Supóngase cinco fotos y se quiere mostrar primero una, luego la siguiente y así hasta la última, haciendo alguna animación o transición mientras van pasando.

Sería ideal si HTML tuviera una marca `<slider>` que permitiese indicarle cuáles son las fotos que se quiere mostrar, por ejemplo:

```
<slider>
    <img src="foto1.jpg">
    <img src="foto2.jpg">
    <img src="foto3.jpg">
    <img src="foto4.jpg">
    <img src="foto5.jpg">
</slider>
```

La marca `<slider>` no existe en HTML y en eso radican los problemas previos a los Web Components. Para construir el slider, sería normal entonces hacerlo escribiendo código HTML y CSS, podría quedar así:[8]

[8] Dodson, Rob (2013). *A Guide to Web Components* <http://css-tricks.com/modular-future-web-components/> [Consulta: 10 Junio 2015]

Código HTML

```
<div id="slider">
    <input checked="" type="radio" name="slider" id="slide1" selected="false">
    <input type="radio" name="slider" id="slide2" selected="false">
    <input type="radio" name="slider" id="slide3" selected="false">
    <input type="radio" name="slider" id="slide4" selected="false">
    <div id="slides">
        <div id="overflow">
            <div class="inner">
                <img src="images/rock.jpg">
                <img src="images/grooves.jpg">
                <img src="images/arch.jpg">
                <img src="images/sunset.jpg">
            </div>
        </div>
    </div>
    <label for="slide1"></label>
    <label for="slide2"></label>
    <label for="slide3"></label>
    <label for="slide4"></label>
</div>
```

Código CSS

```
* {
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
-ms-box-sizing: border-box;
box-sizing: border-box;
}
#slider {
max-width: 600px;
text-align: center;
margin: 0 auto;
}
#overflow {
width: 100%;
overflow: hidden;
}
#slides .inner {
width: 400%;
}
```

```
#slides .inner {
-webkit-transform: translateZ(0);
-moz-transform: translateZ(0);
-o-transform: translateZ(0);
-ms-transform: translateZ(0);
transform: translateZ(0);
-webkit-transition: all 800ms cubic-bezier(0.770, 0.000, 0.175, 1.000);
-moz-transition: all 800ms cubic-bezier(0.770, 0.000, 0.175, 1.000);
-o-transition: all 800ms cubic-bezier(0.770, 0.000, 0.175, 1.000);
-ms-transition: all 800ms cubic-bezier(0.770, 0.000, 0.175, 1.000);
transition: all 800ms cubic-bezier(0.770, 0.000, 0.175, 1.000);
-webkit-transition-timing-function: cubic-bezier(0.770, 0.000, 0.175, 1.000);
-moz-transition-timing-function: cubic-bezier(0.770, 0.000, 0.175, 1.000);
-o-transition-timing-function: cubic-bezier(0.770, 0.000, 0.175, 1.000);
-ms-transition-timing-function: cubic-bezier(0.770, 0.000, 0.175, 1.000);
transition-timing-function: cubic-bezier(0.770, 0.000, 0.175, 1.000);
}
#slides img {
width: 25%;
float: left;
}
#slide1:checked ~ #slides .inner {
margin-left: 0;
}
#slide2:checked ~ #slides .inner {
margin-left: -100%;
}
#slide3:checked ~ #slides .inner {
margin-left: -200%;
}
#slide4:checked ~ #slides .inner {
margin-left: -300%;
}
input[type="radio"] {
display: none;
}
label {
background: #CCC;
display: inline-block;
cursor: pointer;
width: 10px;
height: 10px;
border-radius: 5px;
}
```

```
#slide1:checked ~ label[for="slide1"],
#slide2:checked ~ label[for="slide2"],
#slide3:checked ~ label[for="slide3"],
#slide4:checked ~ label[for="slide4"] {
background: #333;
}
```

Como puede observarse, y gracias a las nuevas funciones de CSS3, no ha sido necesario escribir código JavaScript para los efectos de transición y animaciones. El ejemplo funcional puede verse en http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/slider/sin-web-component.html

Ahora bien ¿se puede reutilizar este código en distintos proyectos, creando una librería y haciendo uso de ella cuando se la necesite? No directamente.

Lo que sí se podría hacer es crear un archivo .css y volver a utilizarlo. Pero con la parte HTML no sería lo mismo, ya que el lenguaje HTML no prevé la importación de librerías ni bloques de código. Al menos hasta que los Web Components entren en escena.

Sin utilizar Web Components, lo normal es copiar código HTML de alguna "librería", documentación o ejemplo previo y pegarlo en la aplicación, saturando la página con plugins de terceros, utilizando JavaScript para sistematizar alguna parte de la implementación.

En un escenario ideal, el lenguaje HTML debería ser expresivo, suficiente para crear complejas interfaces de usuario y también extensivo. Esto es, que los desarrolladores puedan rellenar cualquier faltante, con sus propias etiquetas.

Hoy es posible gracias al nuevo conjunto de tecnologías llamadas Web Components.

Web Components ya conocidos

Antes de entrar en detalles se presentará un pequeño ejemplo más, ésta vez para demostrar que ciertos Web Components ya están presentes en los Browsers de forma nativa.

Como se ha visto antes, tan sólo escribiendo `<video src="archivo.mp4"></video>` en el código HTML de una página, el Browser mostrará un completo reproductor de video con capacidades para iniciar y pausar, también incluirá un control de volumen, línea de tiempo y más. Todo esto sólo por haber escrito `<video>`

Los creadores de Browsers (Chrome, Firefox, Safari, Explorer, etc.) necesitan una forma de garantizar que las etiquetas que ellos implementaron se visualicen siempre correctamente, sin que algún código HTML, CSS o JavaScript escrito por un desarrollador e incorporado en esa misma página, pueda interferir en su comportamiento o representación.

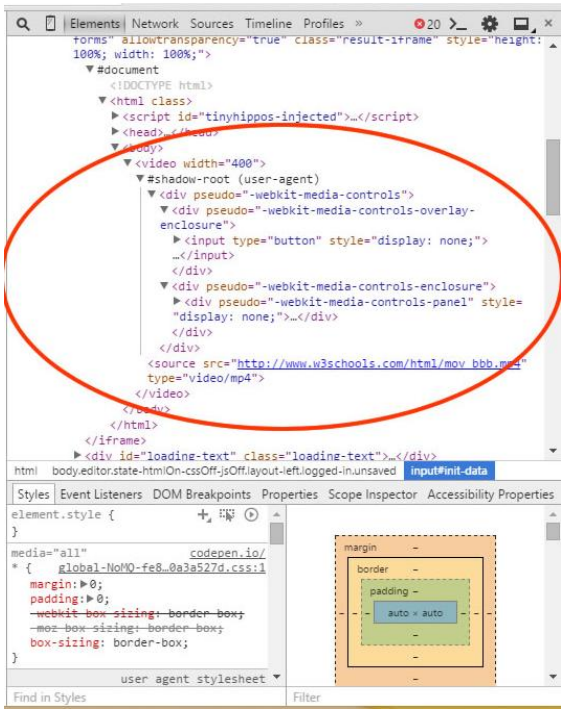
Para resguardarse de eso, en el Browser existe un “pasaje secreto” donde se esconde el código real de la etiqueta `<video>` impidiendo que otras etiquetas o cualquier otro código la afecte: el Shadow DOM.

El Shadow DOM es parte fundamental de los Web Components y permite ocultar ciertos elementos de una página en un DOM especial, evitando que queden expuestos al resto del código.

Haciendo uso de Google Chrome puede visualizarse el código HTML interpretado por el Browser, habilitando la opción de “herramientas para desarrolladores”.

Si en la sección “configuración” (dentro de las herramientas para desarrolladores) se activa el item “Show user agent Shadow DOM”, el Browser mostrará los detalles de la implementación de las etiquetas ocultas, que no son visibles de manera predeterminada.

En la imagen de la izquierda se muestra el código que puede accederse mediante las herramientas para desarrolladores, donde los detalles de la etiqueta `<video>` no quedan expuestos. A la derecha puede verse el detalle completo, habiendo sido activada la opción “Show user agent Shadow DOM”.



En el ejemplo anterior se mostró cómo los detalles de implementación de la etiqueta `<video>` son protegidos y aislados del resto del código, gracias a la utilización de Shadow DOM.

A continuación se explica el estándar de Web Components y, como podrá observarse, la encapsulación por medio de Shadow DOM tendrá un rol fundamental para la construcción segura de nuevos componentes, ya no nativos, sino creados por un desarrollador.

Los Web Components y el estándar

Muchas tecnologías nacen como la aplicación concreta de un producto, luego se popularizan y sus detalles de implementación se dan a conocer públicamente para que distintas empresas y organismos puedan aplicarla. Cuando esto ocurre a nivel global, se habla de un “estándar de facto”.

Hablar de Web Components, técnicamente significa referirse a una especificación propuesta por el W3C que todavía no es soportado por la mayoría de los Browsers y, por lo tanto, debe utilizarse con cuidado para proyectos que requieran funcionar correctamente bajo cualquier situación. No es un estándar aún formalizado -aunque va en vías de serlo- pero ya se habla de Web Components como tal.

El modelo de Web Components involucra un set de tecnologías que facilitan su implementación y que pueden resumirse en: Templates, Custom Elements, Shadow DOM e Imports.

En esta sección se describen las cuatro tecnologías y al finalizar se da un ejemplo con el desarrollo de un primer Web Component respetando el estándar, lo que significa que sólo funcionará correctamente en los Browsers que lo soporten a pleno.

En el próximo capítulo se vuelve sobre el mismo ejemplo, pero con el objeto de prepararlo para funcionar en circunstancias más amplias y reales, acompañando su evolución para generar una “guía para el desarrollo de Web Components”.

Templates

Probablemente la manera más simple de introducir los conceptos detrás de Web Components sea explicando los Templates (plantillas, en castellano). La idea de programar con bloques de código reutilizables ha estado presente desde los inicios del desarrollo web, pero nunca existió la posibilidad de hacerlo con una implementación nativa en HTML.

Puede pensarse en un Web Component como una plantilla dentro del DOM donde los contenidos son analizados por el Browser, pero no son interpretados hasta que el Template se instancie en una copia funcional. Un Template es declarado con el elemento *template* de HTML y cualquiera de sus elementos hijos, formarán parte de ese Template.

Un ejemplo sencillo

Se supondrá que se pretende hacer una aplicación para buscar información en una base de datos sobre especies animales en áreas protegidas de Argentina y mostrar los resultados en pantalla, por ejemplo:

Nombre científico: *Gastrotheca gracilis*
Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog
Autor: Laurent, 1969
Origen: Autóctono

Nombre científico: *Campephilus magellanicus*
Nombres locales: Carpintero gigante, carpintero patagónico, Magellanic Woodpecker
Autor: King, 1828

Origen: Autóctono

Nombre científico: *Tapirus terrestris*
Nombres locales: Tapir, Anta
Autor: Linnaeus, 1758
Origen: Autóctono

Se trata de un claro ejemplo candidato para un Web Component al que se llamará `<especie-item>` y que podrá repetirse tantas veces como resultados devuelva la búsqueda.

Suponga que su código HTML sea:

```
<article>
    <h2>Nombre científico: <em>Gastrotheca gracilis</em></h2>
    <p>Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog</p>
    <p>Autor: Laurent, 1969</p>
    <p>Origen: Autóctono</p>
</article>
```

Para generar un Template, sólo es necesario englobarlo con la etiqueta `<template>`, así:

```
<template id="especieItemTemplate">
    <article>
        <h2>Nombre científico: <em>Gastrotheca gracilis</em></h2>
        <p>Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog</p>
        <p>Autor: Laurent, 1969</p>
        <p>Origen: Autóctono</p>
    </article>
</template>
```

Es importante destacar que al englobar el HTML con la etiqueta `<template>` todo lo escrito ahí no se verá en la página resultante.

Según el W3C "los Templates contienen marcas que serán usadas luego. El contenido de un elemento Template es analizado por el Browser, pero es inerte: los scripts no son procesados, las imágenes no son descargadas, etc. El elemento Template tiene una propiedad llamada content que retiene el contenido del Template en un fragmento del documento. Cuando el desarrollador necesite utilizar ese contenido, podrá mover o copiar ese nodo del DOM según su necesidad."[9]

[9] W3C. *Introduction to Web Components. Working Draft 6 June 2013* <http://www.w3.org/TR/2013/WD-components-intro-20130606/#template-section>

Así, para crear una instancia del Template en cuestión, es necesario escribir lo siguiente con JavaScript:

```
// Obtenemos el Template
var tpl = document.getElementById('especieItemTemplate');

// Creamos una copia de su contenido
var copia = tpl.content.cloneNode(true);

// Agregamos el contenido dentro de un elemento existente en la página
var resultados = document.getElementById('resultados');
resultados.appendChild(copia);
```

El ejemplo funcionando puede verse en http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/nativo/ejemplo-template.html

Como puede observarse, cada instancia del Template que se ha generado tiene copia exacta de su contenido. Entonces, si bien para el ejemplo ha servido, en la práctica se requiere que el template permita reemplazar variables con los contenidos definitivos, para poder mostrar (en este ejemplo) distintas especies en los resultados de búsqueda y no siempre la misma.

Custom Elements

El primer paso para la creación de un Web Component es poder crear nuevas etiquetas HTML y que el Browser entienda cómo representarlas. En el ejemplo anterior se mostró cómo generar Templates, a continuación se extenderá ese concepto para crear elementos reutilizables, con nombre propio.

Según el W3C un Custom Element es "un objeto-plataforma cuya interfaz es definida por su autor" y su objetivo es "proveer una herramienta para que los desarrolladores web puedan construir sus propios elementos del DOM totalmente funcionales".[10]

Para continuar con el caso de los resultados de búsquedas de especies, se creará el Custom Element `<especie-item>` que representará una instancia del Template definido recientemente.

[Consulta: 10 Junio 2015]
[10] W3C. *Custom Elements. Working draft, 21 June 2015* <http://w3c.github.io/webcomponents/spec/custom/> [Consulta: 21 junio 2015]

```
<main id="resultados">
    <especie-item></especie-item>
    <especie-item></especie-item>
</main>
<template id="especieItemTemplate">
    <article>
        <h2>Nombre científico: <em>Gastrotheca gracilis</em></h2>
        <p>Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog</p>
        <p>Autor: Laurent, 1969</p>
        <p>Origen: Autóctono</p>
    </article>
</template>
```

A simple vista puede apreciarse que el (nuevo) elemento `<especie-item>` se utiliza dos veces, mientras que se tiene una sola definición en el Template llamado `especieItemTemplate`.

Ahora bien ¿cómo supo el Browser que `<especie-item>` es una instancia del Template en cuestión? A la definición anterior faltó agregarle el código JavaScript que vincula las partes, creando el Custom Element:

```
var especiePrototype = Object.create(HTMLElement.prototype, {
    createdCallback: {
        value: function() {
            var tpl = document.querySelector('#especieItemTemplate');
            var copia = document.importNode(tpl.content, true);
            this.createShadowRoot().appendChild(copia);
        }
    }
});
document.registerElement('especie-item', {prototype: especiePrototype});
```

El ejemplo funcionando puede verse en http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/nativo/ejemplo-custom-element.html

Para resumir: una vez definido un Custom Element, se puede utilizar n-veces en el código HTML haciendo un llamado a `<especie-item>`. Para que el Browser lo reconozca y sepa cómo representarlo, se ha utilizado el elemento Template, vinculándolo con JavaScript gracias al método `document.registerElement`.

Con lo hecho hasta aquí se obtiene un gran avance para poder crear Web Components, pero todavía faltan herramientas clave que permitan formalizar su definición, generalizar sus contenidos, aislarlo y empaquetarlo.

Shadow DOM

En el ejemplo anterior sobre la etiqueta `<video>` se explicó cómo los Browsers actuales pueden representar marcas complejas (incluso un completo reproductor de video) sin que éstas queden expuestas al resto del código HTML, CSS o JavaScript de la página, protegiendo la definición de la etiqueta y garantizando su integridad visual y funcional.

Shadow DOM de alguna manera es similar a lo que muchas veces se busca con la etiqueta <iframe> pero en este último caso se trata genuinamente de páginas dentro de otras, mientras que los Web Components a través de Shadow DOM, lo que proponen es "componentes dentro de páginas". Pequeñas diferencias, distintos conceptos.

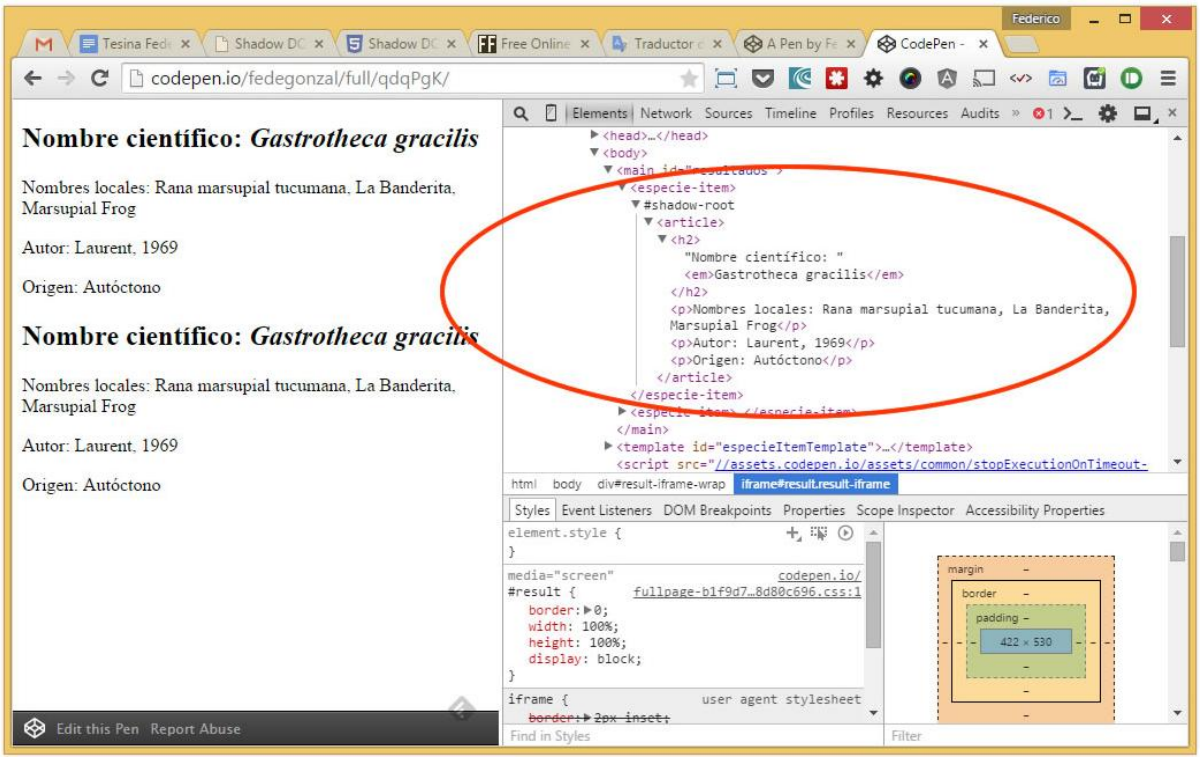
Dicho de otro modo, es un método para combinar múltiples DOM en un sólo árbol de herencias y regular cómo esos árboles interactúan entre sí y con la página, permitiendo un mejor manejo del DOM.[11]

En el reciente ejemplo de resultados de búsqueda para especies, se ha utilizado Shadow DOM antes de registrar el Custom Element, como puede observarse aquí abajo marcado con amarillo:

```
var especieProtoype = Object.create(HTMLElement.prototype, {
    createdCallback: {
        value: function() {
            var tpl = document.querySelector('#especieItemTemplate');
            var copia = document.importNode(tpl.content, true);
            this.createShadowRoot().appendChild(copia);
        }
    }
});
document.registerElement('especie-item', {prototype: especieProtoype});
```

[11] W3C. *Shadow DOM. Working draft. 19 Jun 2015* <http://w3c.github.io/webcomponents/spec/shadow/> [Consulta: 21 Junio 2015]

La imagen a continuación muestra el DOM del ejemplo con el elemento `<especie-item>` utilizando las herramientas para desarrolladores de Google Chrome y habiendo habilitado la opción “show user agent Shadow DOM” para acceder a los detalles de su representación.



Imports

Se ha enunciado que las principales características esperables de los componentes son la abstracción, el empaquetado, la modularidad y el acoplamiento, la extensibilidad y por supuesto, la semántica y la reusabilidad.

Gracias al estándar de Web Components se ha comprobado que disponer de Templates, Shadow DOM y Custom Elements permite cumplir prácticamente todas estas premisas, a excepción del empaquetado.

Los Imports son parte de la especificación de Web Components y proveen una manera de incluir documentos HTML dentro de otros y también CSS, JavaScript o cualquier otra cosa que pueda aceptar una página HTML[12].

[12] Bidelman, Eric (2013). *HTML Imports* <http://www.html5rocks.com/en/tutorials/webcomponents/imports/> [Consulta: 10 Junio 2015]

Evidentemente la función más importante de Imports es facilitar el empaquetado de librerías, facilitando su distribución y utilización, a partir de que ya no sea necesario copiar código o vincular distintos recursos para implementar un componente dentro de una aplicación web.

Pero eso no es todo, Imports también agrega inteligencia al Browser para que reconozca dos recursos idénticos y no los incluya dos veces. En otras palabras, si un Web Component requiere cierto recurso externo y otro componente precisa del mismo, ese recurso se cargará sólo una vez sin importar cuántas veces se lo haya mencionado.

Un ejemplo sencillo. Bootstrap es el Framework más popular del mundo para el maquetado de aplicaciones y sitios web, facilitando una serie de “componentes” que incluyen código HTML, CSS y JavaScript.

A continuación se muestra un código básico para generar una página con soporte para Bootstrap y según se detalla en http://getbootstrap.com/getting-started/ se requieren al menos tres librerías para utilizar el mencionado Framework.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <script src="js/jquery/1.11.2/jquery.min.js"></script>
        <script src="js/bootstrap.min.js"></script>
    </head>
    <body>
        <h1>Hello, world!</h1>
    </body>
</html>
```

Así, por ejemplo, mediante el uso de Imports esas tres líneas podrían “empaquetarse” en un sólo archivo llamado bootstrap.dependences.html reduciendo la página a:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="import" href="libs/bootstrap.dependences.html">
    </head>
    <body>
        <h1>Hello, world!</h1>
    </body>
</html>
```

A continuación se presenta un ejemplo más complejo continuando la lista de especies.

Un primer Web Component

En el presente capítulo se ha hecho una introducción al estándar de Web Components y se abordaron sus principales herramientas: Templates, Shadow DOM, Imports y Custom Elements. En este momento ya es posible crear un primer Web Component según el estándar.

Se recordará el objetivo: crear un Web Component para mostrar una lista de resultados, producto de una búsqueda de especies animales de Argentina.

Cada ítem resultante de la búsqueda se mostrará de la siguiente forma:

Nombre científico: *Campephilus magellanicus*
Nombres locales: carpintero gigante, carpintero patagónico
Autor: King, 1828
Origen: Autóctono

Y su código HTML será:

```
<article>
    <h2>Nombre científico: <em>Campephilus magellanicus</em></h2>
    <p>Nombres locales: carpintero gigante, carpintero patagónico</p>
    <p>Autor: King, 1828</p>
    <p>Origen: Autóctono</p>
</article>
```

Es importante destacar que esta primera versión de nuestro Web Component tiene información estática. Por lo tanto, cada instancia mostrará exactamente el mismo resultado. Suponga que se han creado dos archivos: `index.html` y `especie.item.html` donde este último es el Web Component, mientras que `index` lo utiliza y crea instancias.

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="import" href="libs/especie.item.html">
    </head>
    <body>
        <especie-item></especie-item>
    </body>
</html>
```

especie.item.html

```
<template>
    <article>
        <h2>Nombre científico: <em>Gastrotheca gracilis</em></h2>
        <p>Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog</p>
        <p>Autor: Laurent, 1969</p>
        <p>Origen: Autóctono</p>
    </article>
</template>
<script>
    (function() {
       var importDoc = document.currentScript.ownerDocument;

       var proto = Object.create(HTMLElement.prototype);
       proto.createdCallback = function() {
              var template = importDoc.querySelector('template');
              var clone = document.importNode(template.content, true);
              var root = this.createShadowRoot();
              root.appendChild(clone);
        };

       document.registerElement('especie-item', {prototype: proto});
    })();
</script>
```

En el código puede verse resaltado un nuevo concepto que no ha sido mencionado antes y que refiere al documento que llama al Web Component, y que se indica como `ownerDocument`. Ese fragmento de código es esencial para crear instancias porque permite vincular al archivo `index.html` con el Web Component `especie.item.html` siendo que éste último no conoce la existencia del primero.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/nativo/ejemplo-estatico.html

De lo estático a lo dinámico

El ejemplo anterior ha sido de interés para explicar conceptualmente el desarrollo básico de un Web Component, pero carece de utilidad ya que todas sus instancias tendrán el mismo

contenido. A continuación se presenta un ejemplo más acabado, haciendo uso de parámetros para otorgarle dinamismo.

Previamente se ha instanciado el Web Component escribiendo el código HTML `<especie-item></especie-item>` lo que generó en pantalla la información básica de una especie, según el simulacro de una búsqueda en una base de datos.

Ahora es de interés incorporar una serie de parámetros a esa llamada, de manera que el Web Component genere contenidos distintos, según la información recibida. Se utilizará:

```
<especie-item spName="" localNames="" author="" year="" origin=""></especie-item>
```

Donde spName será el parámetro para el nombre científico de la especie; localNames para los nombres locales; author referirá al nombre de su autor[13]; year indicará el año de registro; origin a su procedencia (autóctona o exótica).

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="import" href="libs/especie.item.html">
    </head>
    <body>
        <especie-item spName="Gastrotheca gracilis"
            localNames="Rana marsupial tucumana, La Banderita, Marsupial Frog"
            author="Laurent" year="1969" origin="Autóctono">
        </especie-item>

        <especie-item spName="Campephilus magellanicus"
            localNames="carpintero gigante, carpintero patagónico"
            author="King" year="1828" origin="Autóctono">
        </especie-item>
    </body>
</html>
```

[13] En taxonomía biológica las especies son registradas mundialmente según las normas del ICZN (International Code of Zoological Nomenclature). Se llama “autor” a la persona que documentó la especie por primera vez y el “año” refiere a la fecha en que fue registrada.

especie.item.html

```
<template>
  <article>
    <h2>Nombre científico: <em id="spName">Scientific Name</em></h2>
    <p>Nombres locales: <span id="localNames">Local names</span></p>
    <p>Autor: <span id="author">Author</span>, <span id="year">Year</span></p>
    <p>Origen: <span id="origin">Native</span></p>
  </article>
</template>

<script>
  (function() {
    var importDoc = document.currentScript.ownerDocument;
    var proto = Object.create(HTMLElement.prototype);
    proto.createdCallback = function() {
      var template = importDoc.querySelector('template');

      // Aplicamos los parámetros recibidos
      template.content.querySelector('#spName').textContent = this.getAttribute('spName');
      template.content.querySelector('#localNames').textContent = this.getAttribute('localNames');
      template.content.querySelector('#author').textContent = this.getAttribute('author');
      template.content.querySelector('#year').textContent = this.getAttribute('year');
      template.content.querySelector('#origin').textContent = this.getAttribute('origin');

      var clone = document.importNode(template.content, true);
      var root = this.createShadowRoot();
      root.appendChild(clone);
    };
    document.registerElement('especie-item', {
      prototype: proto
    });
  })();
</script>
```

Con muy poco esfuerzo se ha generalizado un primer Web Component para aceptar parámetros y representarlos en las distintas instancias que se fueran creando.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/nativo/ejemplo-dinamico.html

En el código anterior se han marcado nuevos conceptos. En el espacio del Template puede verse una etiqueta `span` seguida de un atributo `id,` que permite identificarla fácilmente para

establecer su contenido[14]. Por otra parte con `this.getAttribute('localNames')` se accedió al contenido del parámetro recibido y se asignó ese contenido al `span` a través de `template.content.querySelector('#localNames').textContent`

Próximos pasos

En este capítulo se presentó y resumió el estándar de Web Components y se estableció un ejemplo sencillo con el que se alcanzaron los principales conceptos: Templates, Shadow DOM, Imports y Custom Elements. Por último se ahondó en el pasaje de parámetros, favoreciendo la creación de instancias con diferentes contenidos.

Para simplificar la introducción y hacer foco en lo fundamental se han omitido algunas características de importancia, como por ejemplo: la capacidad de asignar estilos al Web Component, definir nuevos componentes a partir de otros existentes o diversas cuestiones que hacen a la seguridad.

Otra de las omisiones en esta sección tuvo que ver con la compatibilidad y el soporte que los Browsers en la actualidad tienen para el estándar de Web Components, algo fundamental a considerar para desarrollar aplicaciones que funcionen correctamente.

De eso se trata el siguiente capítulo.

[14] Se utilizó la etiqueta `<span>` pero podría haberse invocado cualquier marca válida de HTML que sirviese para construir esa parte del código.

Estado del arte

Los Web Components representan un gran avance para el desarrollo de sitios y aplicaciones web, es verdad. Pero también es verdad que el camino para su completa implementación es largo y eso en gran medida depende de que los Browsers adopten el estándar y los usuarios utilicen Browsers actualizados, que hayan adoptado esas capacidades.

Dicho esto, es de esperar que una aplicación web que utilice Web Components como los definidos en el capítulo anterior, no funcione correctamente si el Browser donde se está ejecutando no soporta de pleno el estándar.

En este capítulo se introducen los desafíos que implica la implementación del estándar para desarrollar aplicaciones web y se presenta una comparativa de las capacidades que tienen los distintos Browsers para interpretar dicho estándar, exponiendo sus limitaciones y problemas actuales.

Comparativa de capacidades para los principales Browsers

El sitio web "Can I use?"[15] provee tablas actualizadas que resumen la aceptación de las tecnologías web como HTML, CSS y JavaScript en los Browsers más utilizados, indicando su grado de implementación y compatibilidad para las versiones previas, actual y futuras de cada Browser, tanto en dispositivos de escritorio como móviles.

[15] "Can I use?" es mantenido y actualizado permanentemente por Alexis Deveria <http://caniuse.com/> [Consulta: 10 Junio 2015]

Imports

A continuación se resume la aceptación de los principales Browsers para el método Imports de HTML5. Como puede observarse sólo Google Chrome y Opera están dando soporte a esta característica, mientras que empresas como Microsoft, Mozilla y Apple no consideran su implementación en el corto plazo.

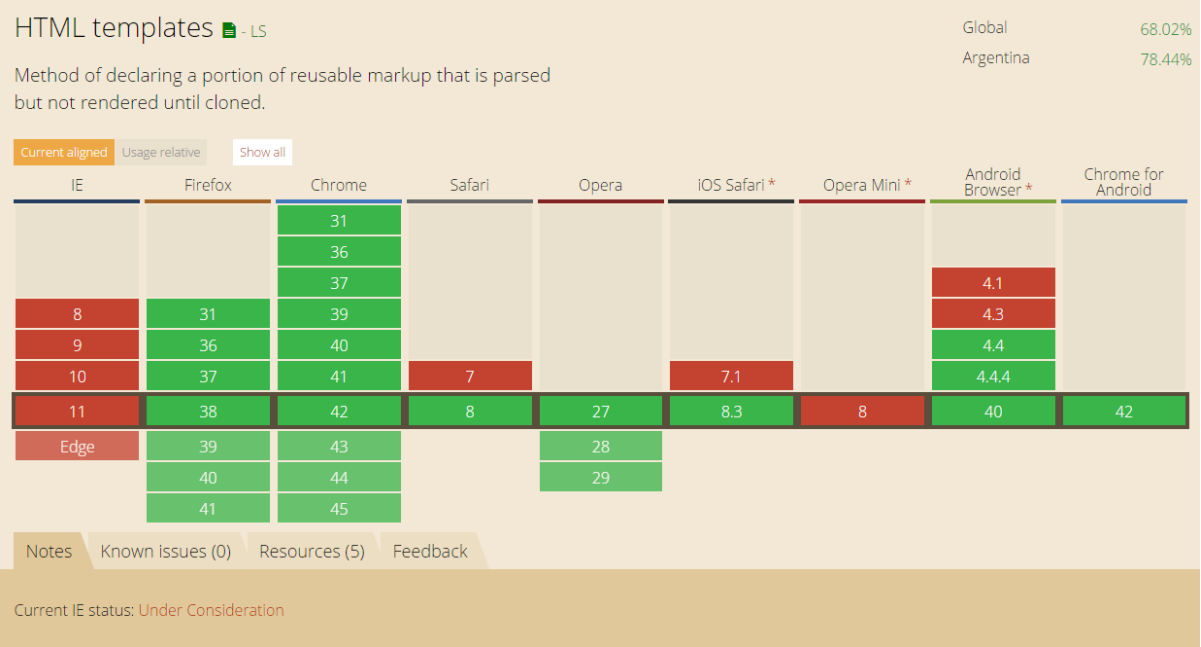
HTML Imports http://caniuse.com/#feat=imports

Fecha: 17 mayo 2015

Templates

El mecanismo de Templates corre con algo más de suerte, prácticamente todas las principales empresas desarrolladoras de Browsers lo han implementado. Sólo Microsoft aún no lo ha hecho, aunque lo considera entre sus próximas capacidades a incorporar.

Cabe destacar que Microsoft está recibiendo una fuerte presión por la rápida implementación del estándar en su conjunto. En https://status.modern.ie/?sort=votes miembros de su comunidad de desarrolladores pueden votar y priorizar las herramientas que la empresa debe incluir en las próximas versiones de su Browser. Las cuatro tecnologías de Web Components han recibido la mayor cantidad de votos, destacándose por sobre todas las demás.



HTML Template http://caniuse.com/#feat=template
Fecha: 17 mayo 2015

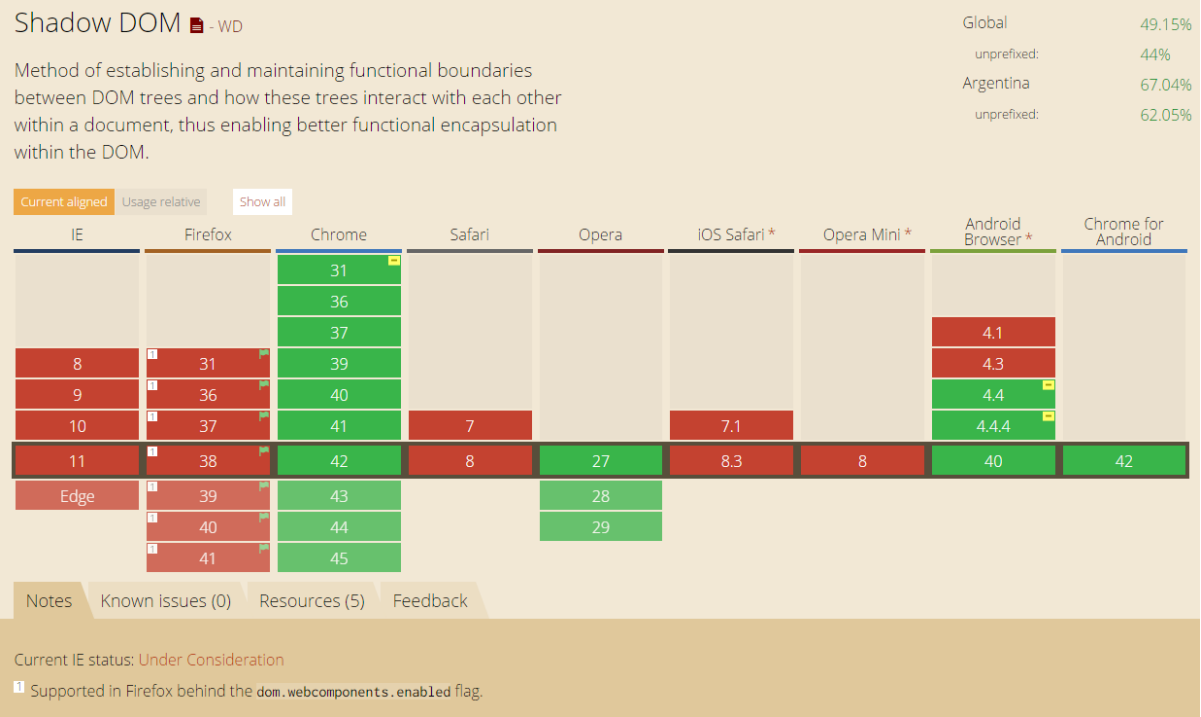
Custom Elements

Al igual que Imports, los Custom Elements tienen gran camino por recorrer hacia su implementación en los principales Browsers. Así, la construcción de etiquetas personalizadas utilizando el estándar de Web Components probablemente deba esperar algunos años para su total disponibilidad.

Custom Elements http://caniuse.com/#feat=custom-elements
Fecha: 17 mayo 2015

Shadow DOM

Vuelve a repetirse la misma condición y sólo Google Chrome y Opera son los Browsers que soportan de pleno los Shadow DOM.



Shadow DOM http://caniuse.com/#feat=shadowdom

Fecha: 17 mayo 2015

Capacidades por Browser y versión

En función de la información resumida en los párrafos anteriores provista por “Can I use?”, se ha creado una tabla sintética que facilita identificar rápidamente las capacidades aceptadas según cada Browser, indicando desde qué versión es soportada cada una. Por ejemplo, Mozilla Firefox soporta Templates desde su versión 31 y se lo mencionará como 31+, para indicar que esa versión y todas las siguientes permiten utilizar Templates de forma nativa.

Browser	Versión actual	Imports	Templates	Custom Elements	Shadow DOM
Internet Explorer	11	No	No	No	No
Firefox	38	No	31+	No	No
Chrome	42	36+	31+	36+	31+
Safari	8	No	8+	No	No
Opera	27	27+	27+	27+	27+
IOS Safari	8.3	No	8.3+	No	No
Opera Mini	8	No	No	No	No
Android Browser	40	40+	4.4+	40+	4.4+
Chrome for Mobile	42	42+	42+	42+	42+

Fecha de revisión: 17 mayo 2015

Análisis de Browsers y versiones en la región

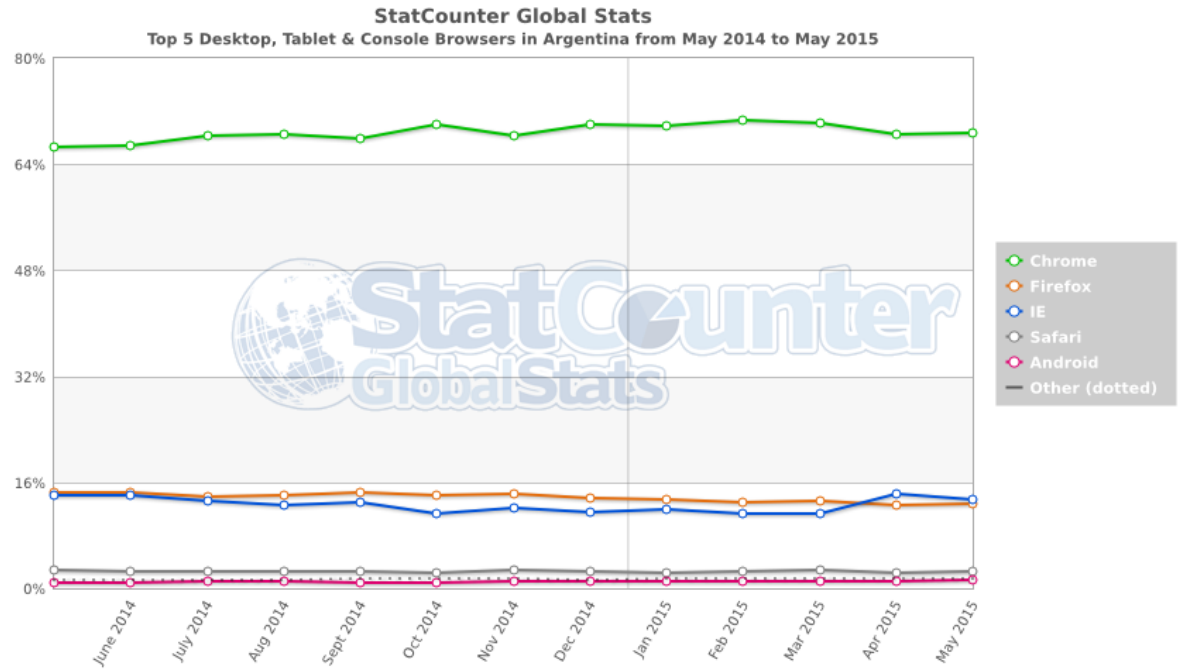
Existen cientos de Browsers, miles de versiones y decenas de sistemas operativos, por lo que analizar todas las opciones sería imposible. Las estadísticas públicas globales coinciden en que los principales Browsers hoy en día son: Chrome, Firefox, Safari, Internet Explorer y Opera.

De ellos, Chrome tiene entre el 50 a 70%, Firefox, Internet Explorer y Safari entre 10 y 20% y Opera oscila entre 2 y 5% del total de usuarios.

En Argentina la cifra varía un poco en relación a Safari debido a que Apple no tiene representación física de ventas y también por diversas restricciones nacionales de importación, pero en general las condiciones son similares.

Más allá de los números puntuales, lo que se puede observar fácilmente es que los principales Browsers del mercado son: Chrome, Firefox, Internet Explorer, Safari y Opera.

A ellos se acota este estudio, agregando el Browser predeterminado del sistema operativo para móviles, Android.



Nota del autor: Si bien existen estudios globales o regionales como los mencionados, que dan una idea acabada de los Browsers, dispositivos y sistemas operativos más utilizados hoy en día, el nivel de detalle de mi interés requiere disponer de información local para tomar decisiones en función de datos que me permitan establecer criterios puntuales para la implementación de Web Components en el ámbito de interés. Así, antes de utilizar estudios globales, he decidido establecer el análisis a partir de información confiable de organismos a los que tengo acceso y que por su tamaño y distribución geográfica, son representativos de la realidad en Argentina y en particular en Ushuaia, Tierra del Fuego, donde resido.

La tabla resume la información de cada organismo analizado y el porcentaje de sesiones reportado por Google Analytics para el mes de Mayo de 2015.

Organismo	Chrome Desktop	Firefox	Internet Explorer	Safari	Android Browser	Opera	Mercado
Parques Nacionales	72,38%	11,35%	7,25%	4,06%	2,96%	0,60%	98,60%
Municip. Ushuaia	72,77%	11,06%	7,36%	4,01%	2,82%	0,60%	98,62%
Promedio	72,58%	11,21%	7,31%	4,04%	2,89%	0,60%	98,61%

Como puede observarse los Browsers analizados representan en promedio casi el 99% del mercado en cuestión, por lo tanto se supondrá que una herramienta de Polyfill compatible con este mercado será más que suficiente para desarrollar aplicaciones que incorporen Web Components.

Sobre los organismos involucrados en el estudio

Parques Nacionales: se trata el sitio web oficial de la Administración de Parques Nacionales que puede ser accedido en www.parquesnacionales.gob.ar y que por su condición federal es representativo del país en general.

Municipalidad de Ushuaia: el sitio web del municipio representa en buena medida la realidad en el sur del país y en particular en Tierra del Fuego. Accesible desde www.ushuaia.gob.ar

Las dos instituciones cuentan con suficientes accesos mensuales como para ser representativos de la mayoría de la población en sus respectivos ámbitos geográficos.

En la tabla presentada arriba, pudo observarse que los porcentajes de acceso por Browser son similares en ambos casos.

Desafíos y limitaciones actuales

Se ha visto que existe consenso general ubicando a los Web Components como una puerta con nuevas oportunidades para los profesionales del desarrollo de aplicaciones web y para la claridad semántica de sus productos, entre muchos otros aspectos y ventajas. El estándar está suficientemente maduro y las empresas fabricantes de Browsers se han comprometido a colaborar para su rápida implementación.

Pero la transición es lenta y siempre existe el riesgo de que un usuario no tenga el Browser *correcto* para que nuestra aplicación se comporte como se espera.

¿Es posible entonces para comenzar a implementar Web Components en nuestros proyectos reales? ¿Correrán correctamente esos productos en los Browsers que utilicen los usuarios?

El 17 de julio de 2014, TJ VanToll escribió en el blog de Telerik -una importante empresa proveedora de componentes de desarrollo- y el título de su nota decía "Por qué los Web Components no están listos para producción… todavía"[16].

Veinte días más tarde, el 7 de agosto, volvió a escribir pero esta vez decía "Por qué los Web Components están listos para producción"[17].

Se podrá responder entonces a estas preguntas con un "no", aunque como se verá en el próximo capítulo, la respuesta es "sí".

[16] VANTOLL, TJ (2014). *Why Web Components Aren't Ready for Production... Yet* <http://developer.telerik.com/featured/web-components-arent-ready-production-yet/> [Consulta: 10 Junio 2015]

[17] VANTOLL, TJ (2014). *Why Web Components Are Ready For Production* <http://developer.telerik.com/featured/web-components-ready-production/> [Consulta: 10 Junio 2015]

Librerías para Polyfills y análisis de Browsers

Se han visto las principales características del estándar de Web Components y las ventajas que supone su utilización; también se pudo comprobar que la mayoría de los Browsers en la actualidad no soportan dicho estándar.

En este capítulo se explica el concepto de Polyfill y se presentan herramientas para facilitar la aplicación de Web Components *hoy*, sin tener que esperar *unos años* hasta que todos los Browsers soporten estas nuevas capacidades y los usuarios actualicen sus Browsers.

El estándar se compone de cuatro tecnologías: Imports, Templates, Custom Elements y Shadow DOM. Surge una pregunta ¿es necesario utilizar todas esas tecnologías para crear un Web Component? No siempre.

Tal vez la principal tecnología del estándar para acercarse a un Web Component sean los Custom Elements. Con ella es posible definir nuevas etiquetas HTML y el Browser sabrá interpretarlas.

¿Que se perdería? Utilizando sólo Custom Elements no se podrían establecer dependencias gracias a los Imports, tampoco se podría organizar el código prolijamente mediante Templates, ni estaría aislado del DOM principal por medio de Shadow DOM.

¿Qué ventajas supone utilizar *sólo* Custom Elements?

El código HTML de las aplicaciones sería más expresivo desde el punto de vista semántico, implementar un mismo comportamiento en distintas páginas sería rápido y sencillo. Pero tal vez lo más importante: se podrá ir mejorando la definición (codificación) del Web Component a medida que exista mayor soporte en los Browsers, porque esa codificación estará fuera de su implementación en el HTML.

¿Es posible utilizar Custom Elements en Browsers que no soportan esta tecnología? Sí, con el uso de Polyfills.

Introducción a los Polyfills

En 2009, mientras trabajaba en su popular libro “Introducing HTML5”, Remy Sharp concibió el término Polyfill para describir una práctica que se iba incrementando en el ámbito del desarrollo de aplicaciones web. Sharp definía un Polyfill como “una pieza de código o plugin

que provee la tecnología que un desarrollador espera de un Browser en forma nativa y que éste no la ofrece"[18].

Como muchas cosas en el mundo de la tecnología, la práctica de utilizar Polyfills es tan antigua como se pueda imaginar. Si bien el término en sí mismo es reciente, hace mucho tiempo que los Browsers presentan implementaciones inconsistentes de las características y estándares del desarrollo web. Siempre se han utilizado Polyfills de una u otra forma. (Satrom, 2014)

Polyfills disponibles para el estándar de Web Components

A la fecha existen dos librerías de Polyfills: Web Components JS y Document Register Element.

Web Components JS ofrece Polyfills para todas las tecnologías implicadas en el estándar, ha sido desarrollado por el equipo de de Polymer, un proyecto de Google, y cuenta con el apoyo activo de Firefox y Opera. El único aspecto negativo es su compatibilidad, ya que sólo garantiza su funcionamiento para las últimas 3 versiones de cada Browser, lo que también se conoce como "Green Browsers".

Document Register Element ha sido desarrollado individualmente por Andrea Giammarchi y **sólo ofrece Polyfill para Custom Elements**, no incluye las demás tecnologías del estándar. A cambio, su compatibilidad con los Browsers del mercado es envidiable.

Dicho esto, vale la pena hacer un análisis más detallado de compatibilidad del estándar, ya no sólo con cada Browser, sino con cada versión en particular y evaluar así, un plan de acción para el desarrollo o implementación de Web Components en lo inmediato.

Polyfill	Chrome Desktop	Firefox	Internet Explorer	Safari	Chrome Android	Safari IOS	Opera	Android Browser
Doc. Register Element	4+	5+	9+	4+	2.1+	4+	11+	2.2.3+
Web Comp. JS	31+	31+	10+	7+	40+	7+	27+	4.4+
Nativo	36+	--	--	--	42+	--	27+	40+

La tabla resume los requerimientos por versión de Browser para cada Polyfill.

[18] Sharp, Remy (2010). *What is a Polyfill?* <https://remysharp.com/2010/10/08/what-is-a-polyfill> [Consulta: 10 Junio 2015]

En el capítulo anterior se introdujo un breve análisis con las estadísticas de navegación para los sitios web de Parques Nacionales y la Municipalidad de Ushuaia. Se conocieron los principales Browsers del mercado y sus promedios por cantidad de sesiones para mayo de 2015.

Es de particular interés en este momento conocer qué penetración de mercado tienen las distintas versiones de los principales Browsers, para establecer la “cuota de mercado” (a la que se llamará cuota de análisis) y con ese dato estimar el público potencial que tendrá cada librería de Polyfill. Luego será fácil averiguar, por su inversa, el público al que no podrá llegarse utilizando cada librería.

La herramienta Google Analytics ofrece información detallada para conocer con qué versión de cada Browser navegan los usuarios, pero el nivel de detalle de la numeración de las versiones, hace que la información sea tan específica que se pierde la visión general de lo que interesa conocer en este momento.

Por ejemplo en el sitio web de la Municipalidad de Ushuaia, sólo con Google Chrome, durante el mes de mayo se han utilizado 177 versiones distintas de ese Browser. Esto es así debido a que Google lanza actualizaciones periódicas y numera las versiones con el formato XX.X.XXXX.XX, por ejemplo 43.0.2357.81 y 43.0.2311.111 representan distintas actualizaciones de la versión 43.

A su vez, las librerías de Polyfills indican su grado de compatibilidad redondeando el número de versión a XX.

Google Analytics permite establecer filtros personalizados para sobreescribir aspectos de la información generada. Definiendo una expresión regular para redefinir la versión de cada Browser agrupando versiones inferiores, se pudo conocer la cantidad de sesiones con el grado de detalle apropiado para este estudio. Por ejemplo, si la versión 42.5.2241.53 tenía 845 sesiones en mayo y la versión 42.3.3276.23 tenía 1056 sesiones, el filtro permitió sumar ambas y saber que las versiones agrupadas en 42.X obtuvieron 1901 sesiones.

Luego, si el requerimiento mínimo de una librería de Polyfill era, por ejemplo, de 31+ para un Browser en particular, sumando las versiones 31.X + 32.X + …. pudo obtenerse la cantidad total de sesiones que utilizaron versiones soportadas para esa librería.

A partir de la cantidad de sesiones por cada versión soportada para cada Browser, se calculó la “cuota de mercado que representa el soporte de cada Polyfill por Browser”.

Conociendo ese número y sabiendo la cantidad total de sesiones que un sitio web obtuvo para todos los Browsers durante el período analizado, se calculó un porcentaje que representa la "la sumatoria de las cuotas de mercado de las versiones soportadas para cada Polyfill y cada Browser". A ese valor se lo llamó "cuota de sesiones".

Ahora bien, al inicio del análisis se mencionó que el estudio se acotaría a los principales Browsers, que representan el 99% de las sesiones.

Con eso en mente, se calculó otro dato al que se llamará "cuota de análisis" y que representa la cuota de sesiones, pero no para el 100% de accesos, sino para ese 99% de los principales Browsers. Esta información es importante porque permite analizar Browsers conocidos globalmente para los que se cuenta con información concreta.

A continuación se resumen los resultados del estudio para la Municipalidad de Ushuaia durante el mes de mayo de 2015.

Polyfill / mercado	Chrome Desktop	Firefox	IE	Safari	Chrome Android	Safari IOS	Opera	Android Browser	∑ cuota sesiones	∑ cuota análisis
Doc. Register Element	35,64%	16,74%	5,72%	0,58%	23,31%	2,34%	0,39%	9,16%	94%	99%
Web Comp. JS	35,22%	12,46%	4,61%	0,54%	16,34%	2,12%	0,32%	8,95%	81%	85%
Nativo	34,64%	0,00%	0,00%	0,00%	15,03%	0,00%	0,32%	8,95%	59%	62%

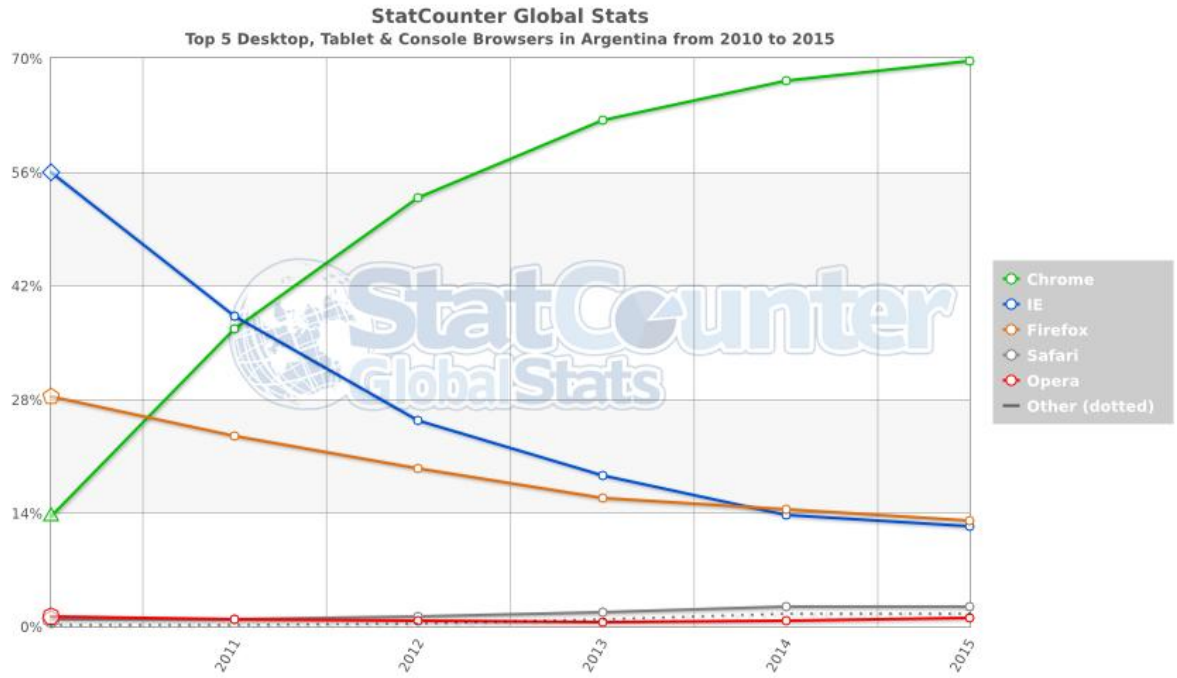
Referencias para la tabla: Las columnas con Browsers muestran la "cuota de mercado que representa el soporte de cada Polyfill por Browser". La columna "∑ cuota de sesiones" muestra la sumatoria de las cuotas de mercado de los Browsers soportados por cada Polyfill, respecto al total de accesos del sitio web. La columna "∑ cuota de análisis" es similar a la anterior pero respecto al total de accesos al sitio web sólo considerando los Browsers analizados (osea, el 99%).

Análisis de tendencia para el mercado de Browsers

Internet Explorer ha sido considerado El Rey del mercado de Browsers desde finales de los 90 hasta 2012, cuando fue superado por Google Chrome. Entre 2002 y 2003, Internet Explorer tuvo alrededor del 95% de los usuarios en el mundo.

En Argentina la tendencia ha sido la misma, incluso más intensa a favor de Chrome en términos globales, debido a que el mercado asiático -liderado por China e India- adoptó en los

últimos años el Browser UC, que se especializa en dispositivos móviles y es casi desconocido en el mundo occidental.



La gráfica evidencia la evolución de Google Chrome en Argentina entre el período 2010 - 2015 y la estrepitosa caída de Internet Explorer.

Microsoft lanzará en julio de 2015[19] su sistema operativo Windows 10 y en conjunto, un nuevo Browser llamado Edge -del que no hay mucha información técnica y no ha sido incluido en este estudio, excepto en las tablas de “Can I use?” presentadas en el Estado del arte.

Considerando que tanto Opera como Safari se han mantenido constantes en los últimos cinco años debajo del 3%, que Internet Explorer y Mozilla Firefox siguen decayendo -aunque a un ritmo menor- y Google Chrome continúa con su mercado ascendente, es de esperar que en los próximos dos años se mantengan esas constantes: Chrome por encima del 70%, Firefox y Explorer/Edge alrededor del 10% y el porcentaje restante dividido entre los demás Browsers.

Conclusiones sobre la elección de librerías de Polyfill

Como ha podido observarse y ya conociendo el análisis de Browsers con precisión, es sencillo interpretar que la librería Document Register Element se resuelve correctamente en el 99% de los accesos analizados, mientras que Web Components JS lo hace para el 85% y sólo el 62% tuvieron soporte nativo.

[19] Esta tesina se está finalizando de editar a finales de junio de 2015, un mes antes del lanzamiento de Edge.

También se ha comentado que la librería Document Register Element sólo ofrece Polyfill para Custom Elements, dejando sin resolución las demás tecnologías del estándar.

Por otra parte, teniendo en cuenta que las personas seguirán actualizando sus Browsers y que Web Components JS cuenta con el 85% del soporte, es fácil suponer que en el corto plazo superará el 90% e inevitablemente, en el mediano plazo, se ubicará muy cerca del 99% que hoy sostiene Document Register Element.

Por lo tanto, podría concluirse que:

Para implementar Web Components en lo inmediato, se recomienda utilizar la librería Document Register Element y limitarse sólo a la tecnología Custom Elements.

Para implementar Web Components en proyectos de mediano o largo plazo que se fueran a implementar a partir de 2016, se recomienda utilizar la librería Web Components JS.

Surge la pregunta ¿Qué sentido tendría implementar Web Components con Document Register Element en lo inmediato si a mediano plazo sería mejor utilizar Web Components JS?

La incorporación de Custom Elements puede entenderse como un paso intermedio para la implementación de Web Components en toda su dimensión. Utilizar Custom Elements permite desarrollar aplicaciones más expresivas, fáciles de mantener, uniformes y -tal vez lo más importante- acorta drásticamente los tiempos de desarrollo.

Es más, la librería Document Register Element es compatible con Web Components JS, por lo tanto el cambio hacia esta última se podrá realizar como un proceso natural, sin riesgos.

Algunas consideraciones sobre Shadow DOM

Cuando una aplicación web incluye componentes de terceros usando scripts remotos, el usuario ejecutará ese código en el contexto de la aplicación. Esto no sólo expone la funcionalidad del código de la aplicación web, sino que también le otorga, al script remoto, acceso completo al contexto cliente (client-side) incluyendo los contenidos de la página, datos locales y funciones protegidas por origen (same-origin). Esta ausencia de aislamiento de código puede acarrear importantes problemas si se incluye código que no se comporte correctamente.

Consecuentemente, al incluir potenciales scripts remotos, el desarrollador de una aplicación web acepta ese riesgo, tanto para la integridad funcional de la aplicación como para los datos en el lado del cliente. Los Web Components protegidos con Shadow DOM, pueden mantener seguros a los datos privados de ataques oportunistas, ocultando información estática en el DOM y aislando elementos interactivos sensibles dentro del Web Component. (De Ryck *et al.*, 2015)

Hablar de riesgos y ataques oportunistas puede sonar a películas de espionaje, pero dejando de lado los ataques y poniendo la mirada sobre los riesgos, algo muy común al incluir plugins o scripts de terceros, es el solapamiento en las definiciones de CSS y sus consecuentes innumerables problemas que causan en el producto esperado.

Dicho de otro modo, según explica Dominic Cooney en su artículo de HTML5 Rocks[20], "Existe un problema fundamental que hace que los plugins sean riesgosos de usar: un widget dentro del árbol del DOM no es encapsulado en relación al resto de la página. Esta falta de encapsulación permite que el CSS del documento pueda ser accidentalmente invadido, el código JavaScript accidentalmente modificado, los IDs de los elementos podrían ser sobreescritos, y así sucesivamente".

A continuación se ofrece un ejemplo sencillo:

Se dispone de una página web en la que se define un botón `<button>` y se le aplica un estilo CSS simple, luego se incorpora un plugin de terceros que también define sus botones… y lo que ocurre es que alguno de los dos sale lesionado.

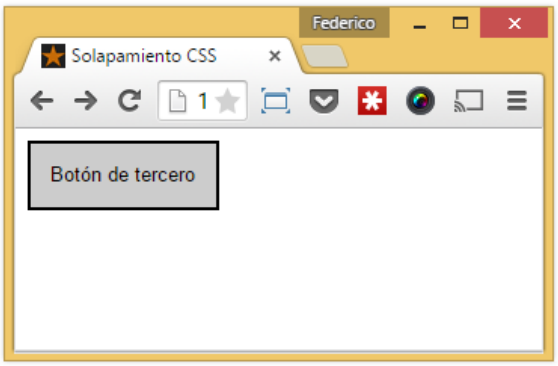
[20] COONEY, DOMINIC (2013). *Shadow DOM* <http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/> [Consulta: 10 Junio 2015]

Este tipo de problemas es muy común en el desarrollo web debido a que el lenguaje CSS no dispone de herramientas para aislar la manera en que serán presentados distintos elementos, y si bien existen técnicas conocidas -como el uso de clases con prefijos propios- lo cierto es que siempre queda latente una posible colisión de definiciones en la hoja de estilos. No sólo eso, el orden en que esas definiciones CSS sean incorporadas en la aplicación web también será crucial.

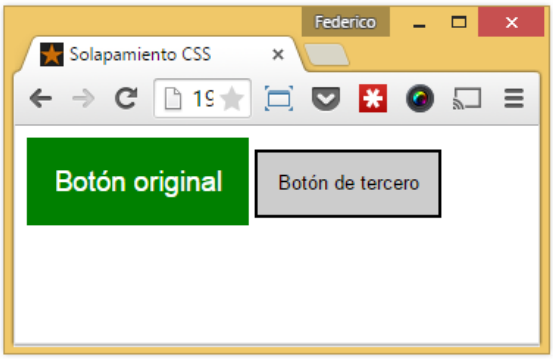
Las imágenes muestran cuatro situaciones típicas:

- La aplicación funcionando sin interferencias.
- El ejemplo que interesa incorporar -también sin interferencias.
- El resultado esperado.
- El resultado obtenido, que está lejos de ser el esperado.

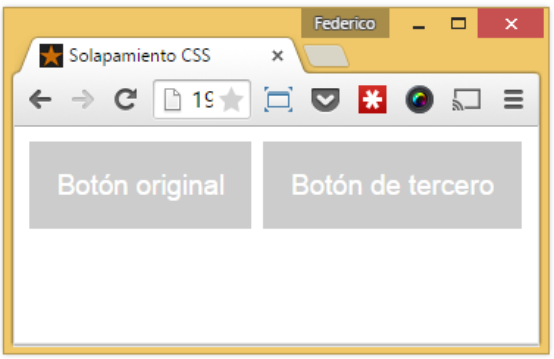
Botón propio funcionando solo



Ejemplo ofrecido por el plugin en su web



Resultado esperado



Resultado obtenido

Se trata de un ejemplo sencillo, pero que a cualquiera que desarrolle aplicaciones web, seguramente le ha traído ingratos recuerdos de luchas pasadas.

Distintos DOM, nuevas oportunidades

Shadow DOM ofrece la posibilidad de crear “páginas dentro de otras”, con cierto grado de comunicación pero con un mayor control de cómo se establece esa comunicación. De alguna

manera es similar a lo que muchas veces se busca con la etiqueta <iframe> pero en este último caso se trata genuinamente de páginas dentro de otras, mientras que los Web Components a través de Shadow DOM, lo que proponen es “componentes dentro de páginas”. Pequeñas diferencias, distintos conceptos.

La creación de un bloque con Shadow DOM es de lo más simple, sólo debe seleccionarse un nodo cualquiera de la página y llevarlo a “las sombras”, por ejemplo:

```
var node = document.querySelector('unSelectorCualquiera');
var shadow = node.createShadowRoot();
```

En general el objetivo pasa por utilizar esta tecnología para la creación de Web Components que queden aislados de la página que los contenga y no para aislar bloques de una misma página -como lo haría el ejemplo anterior.

A continuación se presenta un ejemplo más completo, con la declaración de un plugin al que se llamará `<my-image>` y que tendrá como propósito mostrar una foto y un párrafo de información asociada. Al repetir el componente dentro de la página contenedora se generará una galería de imágenes. Lo importante del ejemplo es que la página con la galería de imágenes tendrá declarados estilos para los elementos `<p>` y dentro de la definición del Web Component también se declararán estilos para `<p>` evitando conflictos gracias al uso de Shadow DOM.

Definición del Web Component // `my-image.html`

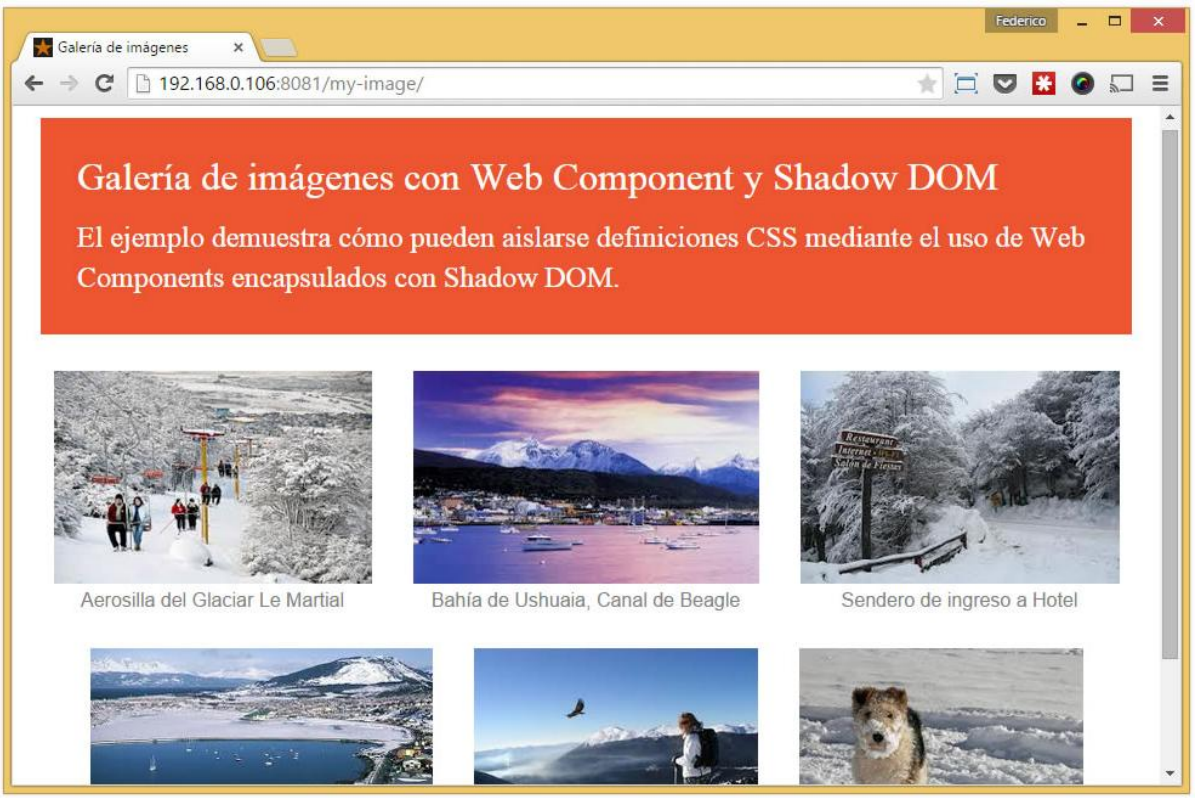
```
<template>
  <style>
        figure { display: block; margin: 15px; }
        p {
          font-family: arial, helvetica, sans-serif;
          font-size: 12pt;
          color: gray;
          line-height: 1.2em;
          margin: 0;
        }
  </style>

  <figure>
    <img param="url" src="http://dummyimage.com/300/cccccc/fff" height="175">
    <p param="caption"></p>
  </figure>
</template>
```

```
<script>
  (function() {
    var importDoc = document.currentScript.ownerDocument;
    var proto = Object.create(HTMLElement.prototype);
    proto.createdCallback = function() {
      var template = importDoc.querySelector('template');

      // Aplicamos los parámetros recibidos
      template.content.querySelector('[param=url]').src = this.getAttribute('url');
      template.content.querySelector('[param=caption]').textContent = this.getAttribute('caption');

      var clone = document.importNode(template.content, true);
      var root = this.createShadowRoot();
      root.appendChild(clone);
    };
    document.registerElement('my-image', {
      prototype: proto
    });
  })();
</script>
```

Definición de la galería de imágenes // `galeria.html`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Galería de imágenes</title>
    <link rel="import" href="my-image.html">

    <style>
      my-image { display: inline-block; }
      p {
        font-family: "Times New Roman", Georgia, Serif;
        font-size: 18pt;
        color: white;
        line-height: 1.4em;
        margin: 15px;
      }

      header { margin: 15px; padding: 15px; background-color: #EC552F; }
      section { text-align: center; }
      h1 { color: white; font-weight: normal; margin: 15px; }
    </style>
  </head>
```

```
  <body>

    <header>
      <h1>Galería de imágenes con Web Component y Shadow DOM</h1>
      <p>El ejemplo demuestra cómo pueden aislarse definiciones CSS
         mediante el uso de Web Components encapsulados con Shadow DOM.
      </p>
    </header>

    <section>
      <my-image url="images/imagen01.jpg" caption="Aerosilla del Glaciar Le Martial"></my-image>
      <my-image url="images/imagen02.jpg" caption="Bahía de Ushuaia, Canal de Beagle"></my-image>
      <my-image url="images/imagen03.jpg" caption="Sendero de ingreso a Hotel"></my-image>
      <my-image url="images/imagen04.jpg" caption="Bahía Encerrada congelada"></my-image>
      <my-image url="images/imagen05.jpg" caption="Cumbre en el Cerro Castor"></my-image>
      <my-image url="images/imagen06.jpg" caption="Postales de Ushuaia"></my-image>
    </section>

  </body>
</html>
```



La captura de pantalla muestra claramente cómo las etiquetas `<p>` han sido aisladas gracias a la tecnología de Shadow DOM y, pese a que la declaración ambigua que se estableció en la sección de estilos CSS para `p { }` tanto en el archivo `galeria.html` como en el archivo `my-`

`image.html` no hubo ninguna colisión y ambas definiciones pudieron convivir sin inconvenientes.

En el código expuesto en las dos páginas anteriores se ha marcado con amarillo las definiciones de CSS que hubieran traído problemas si no se habría usado Shadow DOM. También puede apreciarse que en el archivo `galeria.html` se definieron estilos para la nueva etiqueta my-image haciendo que las distintas instancias del Web Component se incorporen una al lado de la otra, con la finalidad de crear visualmente la galería de imágenes.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/my-image/galeria.html

Aislar y encapsular más allá del CSS

Sólo a modo de comentario, ya que en este capítulo no se abordarán Web Components de mayor complejidad, vale la pena mencionar que todo lo dicho respecto a la protección del código CSS para que no se invadan declaraciones también se aplica al lenguaje JavaScript.

Esto quiere decir que todo el código de JavaScript escrito dentro del Web Component no podrá invadir el contexto de su contenedor, protegiendo así ya no sólo los estilos, sino también aspectos funcionales de la aplicación web donde se lo utilice.

Guía básica para el desarrollo de Web Components

En los capítulos anteriores se abordó el estándar de Web Components, se desarrollaron ejemplos de componentes básicos y se expusieron sus limitaciones para la puesta en producción en la actualidad. También se presentaron librerías de Polyfill y se analizaron los principales Browsers del mercado. Con toda esa información se dieron recomendaciones sobre cómo actuar en el corto, mediano y largo plazo.

En este capítulo se desarrollan ejemplos prácticos que explican técnicamente cómo implementar Web Components en la actualidad, haciendo uso de las librerías mencionadas: Document Register Element y Web Components JS.

Al inicio de esta tesina, en el capítulo “Introducción al estándar de Web Components”, se planteó un ejemplo sencillo donde se requería hacer una aplicación para buscar información en una base de datos sobre especies animales en áreas protegidas de Argentina y mostrar los resultados en pantalla, por ejemplo:

Nombre científico: *Gastrotheca gracilis*
Nombres locales: Rana marsupial tucumana, La Banderita, Marsupial Frog
Autor: Laurent, 1969
Origen: Autóctono

Nombre científico: *Campephilus magellanicus*
Nombres locales: Carpintero gigante, carpintero patagónico, Magellanic Woodpecker
Autor: King, 1828
Origen: Autóctono

Durante ese primer ensayo de Web Component se creó la etiqueta `<especie-item>` y con la ayuda de algunos parámetros, se pudieron construir cada uno de los resultados de esa búsqueda mediante Web Components. Por ejemplo:

```
<especie-item spName="Gastrotheca gracilis" localNames="Rana marsupial tucumana, La Banderita, Marsupial Frog" author="Laurent" year="1969" origin="Autóctono"></especie-item>
```

También se vio que ese ejemplo sólo funciona en Browsers que soporten de forma nativa el estándar de Web Components. Eso cambiará a continuación con el uso de Polyfills.

Ejemplo básico de Custom Elements con Polyfills

En esta sección se desarrollará un Web Component acotado sólo a la tecnología de Custom Elements y se lo construirá y comparará utilizando las dos librerías ya mencionadas.

Custom Elements con la librería Document Register Element

Como se ha comentado anteriormente, Document Register Element sólo tiene soporte para Custom Elements, dejando de lado las demás tecnologías como HTML Imports, Shadow DOM y Templates.

También es cierto que HTML Imports y Templates son fáciles de emular bajo circunstancias simples y con algunas restricciones, como se verá a continuación.

La librería Document Register Element puede descargarse desde https://github.com/WebReflection/document-register-element y en http://webreflection.blogspot.com.ar/2014/07/ pueden leerse los fundamentos por los cuales su autor, Andrea Giammarchi, decidió desarrollar dicho Polyfill.

El código requerido para la creación de un Custom Element con Document Register Element es exactamente igual a como se lo crea con el estándar:

```
document.registerElement('especie-item', {
    prototype: Object.create(HTMLElement.prototype, {
        createdCallback: {
            value: function() {
                 // Aquí irá el código con la creación del elemento
            }
        }
    })
});
```

Para incorporar el elemento creado en una página HTML se deben hacer dos cosas:

- Cargar las librerías con el Polyfill y el componente creado.
- Utilizar el componente a través de su etiqueta.

```
<script src="libs/document-register-element.js"></script>
<script src="libs/especie-item.js"></script>
<body>
    <especie-item spName="Gastrotheca gracilis" localNames="Rana marsupial tucumana, La
Banderita, Marsupial Frog" author="Laurent" year="1969" origin="Autóctono"></especie-item>
</body>
```

Si se presta atención puede observarse que la librería del Web Component creado es un archivo de JavaScript llamado `especie-item.js` y no un archivo HTML como se esperaría hacer con el estándar, esto es justamente porque con Document Register Element no se dispone de HTML Imports, una manera de emularlo es escribiendo el componente dentro de un archivo js para luego incluirlo en la página web con `script` en vez de `link`.

Al hacerlo de este modo hay que considerar que `script` no verifica superposición de librerías como sí lo hace HTML imports con `link`, entonces debe tenerse precaución de no cargar dos veces la misma librería, porque podrían producirse inconsistencias.

También se había mencionado que no se disponía de Templates, a continuación se muestra el ejemplo completo con la definición del Web Component, emulando Templates con una expresión regular.

```
document.registerElement('especie-item', {
  prototype: Object.create(
    HTMLElement.prototype, {
    createdCallback: { value: function() {

      // sin emulateTemplate() el código del componente debería
      // haberse escrito en una sóla línea de string, lo que sería poco legible
      var template = emulateTemplate( function() {
      /*

      <article>
        <h2>Nombre científico: <em param="spName">Scientific Name</em></h2>
        <p>Nombres locales: <span param="localNames">Local names</span></p>
        <p>Autor: <span param="author">Author</span>, <span param="year">Year</span></p>
        <p>Origen: <span param="origin">Native</span></p>
      </article>
```

```
      */
      });

      this.innerHTML = template;

      // Aquí se reciben y procesan los parámetros recibidos
      this.querySelector("[param=spName]").innerHTML = this.getAttribute("spName")
        || this.querySelector("[param=spName]").innerHTML;

      this.querySelector("[param=localNames]").innerHTML = this.getAttribute("localNames")
        || this.querySelector("[param=localNames]").innerHTML;

      this.querySelector("[param=author]").innerHTML = this.getAttribute("author")
        || this.querySelector("[param=author]").innerHTML;

      this.querySelector("[param=year]").innerHTML = this.getAttribute("year")
        || this.querySelector("[param=year]").innerHTML;

      this.querySelector("[param=origin]").innerHTML = this.getAttribute("origin")
        || this.querySelector("[param=origin]").innerHTML;

    }}, // createdCallback

    attributeChangedCallback: { value: function(name, previousValue, value) {
      console.log(name);
      console.log(previousValue);
      console.log(value);
    }} // attributeChangedCallback

  }) // prototype
}); // registerElement

// Esta función, escrita por Nate Ferrero, recibe un string
// con saltos de línea y lo devuelve en una sola línea
function emulateTemplate(f) {
  return f.toString().match(/\/\*\s*([\s\S]*?)\s*\*\//m)[1];
};
```

Muchas de las funciones disponibles en la librería Document Register Element no han sido abordadas y son de gran utilidad al momento de crear Web Components complejos. El método `attributeChangedCallback()` permite activar una escucha permanente en cada instancia del

componente y al detectar cambios en sus atributos, lanza una acción. Este mecanismo es de interés por ejemplo para la sincronización entre componentes.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/document-register-element/ejemplo-dinamico.html

En el próximo capítulo se abordan algunas de estas funciones avanzadas, mientras que otras escapan a esta tesina principalmente por razones de longitud.

Custom Elements con la librería Web Components JS

A continuación se presenta el mismo ejemplo, pero esta vez se lo desarrollará con la librería de Polyfill Web Components JS.

Ya se ha mencionado que Web Components JS es un proyecto liderado por Google y cuenta con el apoyo activo de Firefox y Opera. En http://webcomponents.org/ se presenta la documentación y su código puede ser descargado desde https://github.com/WebComponents/webcomponentsjs

El concepto de Polyfill establece que la manera en que una librería es utilizada debería ser igual a estándar, ya demostró con Document Register Element y a continuación se podrá observar lo mismo con Web Components JS -la forma en que se usa es exactamente igual a como se codifica siguiendo el estándar.

Por lo tanto, si en el ejemplo anterior se cambia el llamado de una librería por otra, todo estará listo para seguir funcionando y no habrá que modificar absolutamente nada en la definición del componente ni en la forma en que se usa.

```
<script src="libs/document-register-element.js"></script>
<script src="libs/webcomponents-lite.js"></script>
```

Todo el resto del código seguirá igual.

Ahora bien, a diferencia de Document Register Element, Web Components JS incluye HTML Imports y Templates.

A continuación se expone el ejemplo con los cambios necesarios para incorporar estas dos nuevas tecnologías, no disponibles en la librería anterior.

Para descargar la librería de Web Components JS y sus dependencias la mejor manera es hacerlo con Bower, un manejador de paquetes y dependencias que se encargará de descargar todo lo necesario. Más información sobre Bower en http://bower.io/

```
bower install --save webcomponentsjs
```

Al ejecutar el comando anterior se descargará la librería con todas sus dependencias dentro de la carpeta `bower_components/webcomponentsjs/` luego sólo será necesario referenciar a `webcomponents.js` como se muestra a continuación.

Web Components JS ofrece dos opciones para utilizar su librería, la versión completa y la versión "lite" que no incluye soporte para Shadow DOM y, a cambio, ocupa sólo 37Kb contra los 114Kb de la original.

El uso del Web Component seguirá siendo similar, sólo cambia la forma en que se lo carga, esta vez, utilizando HTML Import.

```
<script src="bower_components/webcomponentsjs/webcomponents-lite.js"></script>
<link rel="import" href="my_components/especie-item.html">
<body>
    <especie-item spName="Gastrotheca gracilis" localNames="Rana marsupial tucumana, La Banderita, Marsupial Frog" author="Laurent" year="1969" origin="Autóctono"></especie-item>
</body>
```

Puede observarse que ahora la definición del componente hace referencia al archivo `especie-item.html` a continuación se detalla su código.

Sin necesidad de hacer ningún otro cambio respecto al ejemplo anterior, esta nueva versión ya funciona correctamente, pero para mejorar el ejemplo se hará uso de Template - aprovechando que la librería lo soporta- y no emulado como se utilizó hasta ahora:

```
<template>
  <article>
    <h2>Nombre científico: <em param="spName">Scientific Name</em></h2>
    <p>Nombres locales: <span param="localNames">Local names</span></p>
    <p>Autor: <span param="author">Author</span>, <span param="year">Year</span></p>
    <p>Origen: <span param="origin">Native</span></p>
  </article>
</template>

<script>
```

```
// Obtenemos el DOM de la página que está utilizando el Web Component
var ownerDocument = document._currentScript ?
    document._currentScript.ownerDocument : document.currentScript.ownerDocument;

document.registerElement('especie-item', {
  prototype: Object.create(
    HTMLElement.prototype, {
    createdCallback: { value: function() {

      var template = ownerDocument.querySelector('template');
      this.appendChild(template.content.cloneNode(true));

      // Aquí se reciben y procesan los parámetros recibidos
      // Igual al ejemplo anterior

    }} // createdCallback

  }) // prototype
}); // registerElement
</script>
```

Como pudo verse, ambos ejemplos son muy similares. El mayor cambio en el último se da al utilizar Template con el Polyfill. El código queda más legible y se colabora con la separación de lenguajes: dentro de la etiqueta <template> existe sólo HTML y dentro de la etiqueta <script> sólo se ha escrito JavaScript.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/capitulos-preliminares/especie-item/web-component-js/ejemplo-dinamico.html

En amarillo se han marcado dos fragmentos de código que vale la pena mencionar: el primero hace referencia a la necesidad de identificar quién es la página que está utilizando al Web Component que se ha definido, el segundo inserta en esa página el resultado de la instancia creada.

Ejemplo con Polyfill Web Component JS funcionando en Google Chrome 43.0.2357.81

Ejemplo con Polyfill Web Component JS funcionando en Internet Explorer 11

Ejemplo con Polyfill Web Component JS funcionando en Mozilla Firefox 35.0

En resumidas cuentas: la gran diferencia de estos dos ejemplos comparados con el mismo, presentado en el capítulo “estado del arte”, es que los últimos funcionan en la gran mayoría de los Browsers -tal como se analizó en “Librerías para Polyfills y análisis de Browsers”- mientras que el ejemplo sin Polyfills sólo funciona en Browsers con soporte nativo, osea, el 60% de los usuarios hoy en día.

Ejemplos para el desarrollo de componentes

A través de los distintos capítulos de este trabajo se presentaron las principales características para comprender el desarrollo de Web Components, ya sea en forma nativa o incorporando Polyfills para alcanzar un mayor grado de compatibilidad con los Browsers.

Si bien se dieron ejemplos para ilustrar la tecnología, trató de mantenerse la tesina sobre una línea conceptual que introdujera al lector en las capacidades y oportunidades que brinda el estándar, más que en los detalles de su implementación técnica.

También se llevó adelante un análisis estadístico de la situación actual en Argentina y Ushuaia respecto a la penetración de mercado de los Browsers y sus distintas versiones, relacionando esto con el soporte de cada librería de Polyfill.

Este capítulo se presenta a través de ejemplos prácticos que sirven de referencia para desarrollar componentes propios, incorporando lo aprendido y agregando nuevas capacidades.

Se han desarrollado cinco ejemplos, con el objetivo de cubrir las principales situaciones que puedan encontrarse al momento de desarrollar componentes propios.

Ninguno de los casos desarrollados tiene como propósito estar listo para producción, son ejemplos explicativos que tienen como finalidad ahondar en las características principales a tener en cuenta para el desarrollo de Web Components. Se trata de ejemplos con un fin claramente pedagógico.

Concretamente se espera demostrar:

1. Cómo recibir parámetros (similar a los ya vistos).
2. Cómo incluir elementos complejos: nativos, propios o de terceros.
3. Cómo iterar datos y generar instancias de componentes.
4. Cómo extender un elemento nativo.
5. Cómo articular datos entre componentes.

En esta etapa de la tesina es de interés demostrar ejemplos de casos reales. Al trabajar en situaciones aplicables a proyectos que pudieran ir a producción, es necesario contar con herramientas preparadas para tal fin.

Sobre la elección de herramientas

Como se ha visto, el estándar no está preparado aún para proyectos en producción y de las dos librerías disponibles, sólo una brinda soporte para Shadow DOM. Por lo tanto es razonable pensar que los ejemplos se desarrollen con el Polyfill de Web Components JS.

Por otra parte, al desarrollar aplicaciones reales también es común utilizar herramientas que, en una capa de nivel superior, ayuden a evitar errores, trabajar más rápido y ordenado. A la fecha existen tres proyectos conocidos, que haciendo uso de Web Components JS agregan una capa de abstracción para facilitar la programación de componentes propios: X-Tags, Bosonic y Polymer.

En capítulos anteriores se mencionó que la librería de Polyfill, Web Components JS, es desarrollada y mantenida por Google en colaboración con Mozilla.

Bosonic[21] es un proyecto individual que no cuenta con el apoyo explícito de empresas del sector y no participa en la construcción de librerías de base para Polyfill, sólo las implementa. Su documentación es sencilla de interpretar -aunque un poco acotada- y provee 15 Web Components prefabricados listos para utilizar.

X-Tags[22] es desarrollado por Mozilla y sus miembros colaboran con Google en la evolución del Polyfill. Su documentación es limitada y muchos de sus ejemplos no funcionan correctamente. Incluye una librería con 12 Web Components prefabricados.

Polymer[23] es la herramienta que más ha madurado y la única que se encuentra en versión de producción. Es desarrollada en Google por el mismo equipo que programa el Polyfill y cuenta con una pujante comunidad mundial de desarrolladores. Ofrece 83 componentes oficiales agrupados en 7 áreas de interés, donde día a día se incluyen nuevos elementos.

En el sitio web http://webcomponents.org/ se resume información sobre el estándar y sus librerías asociadas, incluyendo artículos comparativos sobre las herramientas recién mencionadas.

En https://customelements.io/ y en http://component.kitchen/ se publican elementos desarrollados por individuos que quieren compartir sus Web Components, a la fecha hay casi 700 componentes listos para usar. Los contenidos no son chequeados ni curados, simplemente se publica automáticamente lo que cada autor envíe, por lo tanto pueden encontrarse distintos grados de calidad según quién lo haya desarrollado.

[21] Web oficial de Bosonic <http://bosonic.github.io/index.html>

[22] Web oficial de X-Tags <http://x-tags.org/index>

[23] Web oficial de Polymer <https://www.polymer-project.org/>

No es objetivo de esta tesina analizar o comparar las tres herramientas mencionadas, pero frente a la necesidad de elegir una en particular, se optó por Polymer en función de las ventajas resumidas en párrafos anteriores.

Comentarios sobre Polymer

El conjunto de herramientas que forman Polymer están diseñadas para que los desarrolladores puedan generar componentes reutilizables para la web, rápidamente y fácil.

Provee una sintaxis declarativa que simplifica la definición de elementos personalizados, incorpora características como las ya mencionadas plantillas (templates) pero agregándoles capacidades extra como la vinculación de datos en dos sentidos (two-way data binding), la observación de propiedades (property observers), las vistas asistidas por modelos MDV (model driven views), disparadores y escuchadores de eventos y más.

Los principales detalles de las capacidades y características particulares de Polymer se explican junto a cada uno de los ejemplos que se desarrollan a continuación.

Ejemplo 1. Cómo recibir parámetros

Como se ha hecho a lo largo de toda la tesina, en este primer ejemplo se define nuevamente el elemento `<especie-item>` pero esta vez haciendo uso de la librería Polymer, que como se mencionó recientemente hace uso del Polyfill Web Compoents JS.

El ejemplo no requiere de mucha explicación porque ya es bien conocido, a continuación se expone el código del Web Component `especie-item.html`

```
<link rel="import" href="../polymer/polymer.html">
<dom-module id="especie-item">
	<template>
		<article>
			<h2>Nombre científico: <em><span>{{spname}}</span></em></h2>
			<p>Nombres locales: <span>{{localnames}}</span></p>
			<p>Autor: <span>{{author}}</span>, <span>{{year}}</span></p>
			<p>Origen: <span>{{origin}}</span></p>
		</article>
	</template>
</dom-module>
```

```
<script>
Polymer({
    is: "especie-item",
    properties: {
      spname: { type: String, value: "Nombre científico" },
      localnames: { type: String, value: "Nombres locales" },
      author: { type: String, value: "Autor" },
      year: { type: String, value: "Año" },
      origin: { type: String, value: "Origen" }
    }
});
</script>
```

A continuación se describen algunas características propias de Polymer que no se han explicado anteriormente y que sirven también para los próximos ejemplos.

En este capítulo no se espera documentar Polymer, ya que no es un objetivo de esta tesina. Sólo se explicará lo indispensable para que el lector pueda comprender los ejemplos y en caso de presentar interés, acceder a su documentación[24] en Internet o a partir del material citado en la bibliografía.

Un componente de Polymer está formado por tres partes:

1. Dependencias
2. Template
3. Script

Dependencias

En la primera línea se utiliza HTML Imports para incluir Polymer, todos los Web Components desarrollados en estos ejemplos harán lo mismo, ya que requieren de esa librería para funcionar. El hecho de los componentes carguen Polymer no significa que la librería se cargue repetidas veces, porque una de las características de HTML Imports es su capacidad de identificar si un recurso ya fue utilizado y no volver a cargarlo innecesariamente.

Template

[24] Polymer Developer Guide. Feature overview - versión 1.0 <https://www.polymer-project.org/1.0/docs/devguide/feature-overview.html> [Consulta: 10 Junio 2015]

La siguiente línea hace referencia a `<dom-module>` que se utiliza para agrupar las definiciones de estilos `<style>` y la plantilla `<template>` del componente a crear. En el ejemplo no se han utilizado estilos.

Dentro del `<template>` puede observarse la etiqueta `<span>{{author}}</span>` donde la variable "author", que se identifica al estar encerrada entre llaves, será reemplazada por el valor recibido al crear cada instancia de `<especie-item>`. Más abajo, en la última línea de código resaltada en amarillo, puede observarse cómo se reciben los parámetros -que estarán vinculados a las variables del template- y también cómo se puede predeterminar un valor.

Script

Por último, al inicio del script se utiliza `Polymer({ ... })` para crear el componente, asignándole un nombre "`especie-item`" y luego definiendo sus propiedades, que serán los parámetros que podrá recibir en cada una de las instancias que se creen de él.

Finalmente puede utilizarse el Web Component definido como ya se ha visto anteriormente:

```
<!DOCTYPE html>
<html>
<head>
  <!-- Polyfill Web Components -->
  <script src="components/webcomponentsjs/webcomponents.min.js"></script>

  <!-- Web Compontent: especie-item -->
  <link rel="import" href="bower_components/especie-item/especie-item.html">
</head>
<body>

  <especie-item spname="Gastrothe cagracilis"
    localnames="Rana marsupial tucumana, La Banderita, Marsupial Frog"
    author="Laurent" year="1969" origin="Autóctono">
  </especie-item>

  <especie-item spname="Campephilus magellanicus"
    localnames="carpintero gigante, carpintero patagónico"
    author="King" year="1828" origin="Autóctono">
  </especie-item>

</body>
</html>
```

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/como-recibir-parametros.html

Ejemplo 2. Cómo incluir elementos complejos

En esta sección se hará una ligera modificación al ejemplo anterior, suponiendo que es de interés dar mayor flexibilidad a la información contenida en la ficha de especie.

En vez de pasar todo el contenido en parámetros, se dejará la libertad de incorporar otros elementos dentro de la ficha; también se agregó un parámetro en el que se indicará la URL con una foto.

Por ejemplo:

```
<especie-item spname="Campephilus magellanicus" url="fotos/campephilus-magellanicus.jpg">
  <p>Carpintero gigante, Carpintero patagónico</p>
  <p>King, 1828</p>
  <p>Autóctono</p>
</especie-item>
```

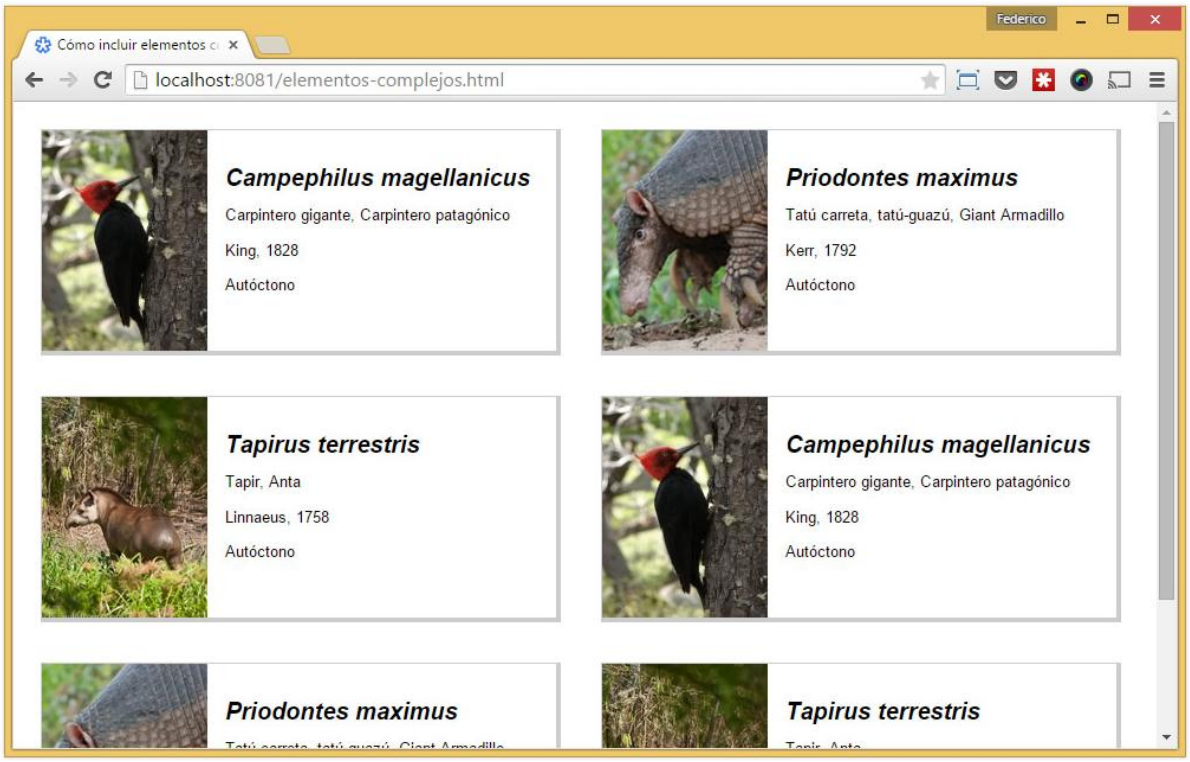
En la imagen se muestra un ejemplo del resultado obtenido habiendo definido una serie de estilos dentro de la definición del Web Component.

El código del Web Component es muy similar al anterior, de hecho más simple porque no incluye tantos parámetros:

```
<template>
  <article>
    <img class="picture" src="{{url}}">
    <div class="abstract">
      <h2><em><span>{{spname}}</span></em></h2>
      <content></content>
    </div>
  </article>
</template>
```

```
<script>
Polymer({
    is: "especie-item-complejo",
    properties: {
      spname: { type: String, value: "Nombre científico" },
      url: { type: String }
    }
});
</script>
```

Aparece un nuevo elemento llamado `<content>` que se establece como un "punto de inserción". Se declara en la definición de la plantilla y de estar presente, automáticamente recibirá todo lo que se haya incorporado dentro de las marcas `<especie-item-complejo> ... </especie-item-complejo>` para cada instancia creada.



Con muy poco esfuerzo y ayudándose de algunos estilos CSS, es fácil armar una lista de galería de especies generando distintas instancias del Web Component definido.

El ejemplo de acaba de describirse tuvo por objetivo demostrar de manera sencilla cómo se pueden definir un componente que pueda recibir otros en su interior. El mecanismo es el mismo, tanto para soportar componentes nativos como `<p>` o cualquier otro tipo de elemento definido por terceros.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/elementos-complejos.html

Ejemplo 3. Cómo iterar datos y generar instancias de componentes

En el ejercicio anterior se generó un componente y se lo iteró manualmente generando una galería de especies. A continuación se muestra un ejemplo similar, pero esta vez la información se obtendrá a partir de una fuente de datos externa en formato JSON. Este es un típico caso de provisión de información mediante APIs REST.

JavaScript es un lenguaje muy rico respecto a sus características de programación orientada a objetos. El estándar JSON (JavaScript Object Notation; y en castellano notación de objetos de JavaScript) es utilizado en prácticamente todas las aplicaciones web modernas, tanto para comunicación como para persistencia de datos. La sintaxis de la notación de objetos de JavaScript es simple, flexible y concisa; mucho más compacta y flexible que XML, al que ha reemplazado (Elliott, 2014).

```
<link rel="import" href="../polymer/polymer.html">
<link rel="import" href="../especie-item/especie-item-complejo.html">
<link rel="import" href="../iron-ajax/iron-ajax.html">

<dom-module id="especie-lista">
        <template>
            <iron-ajax url="{{src}}" last-response="{{data}}" auto></iron-ajax>

            <template is="dom-repeat" items="{{data}}">
                    <especie-item-complejo spname="{{item.spname}}" url="{{item.image}}">
                        <p>{{item.localnames}}</p>
                        <p><span>{{item.author}}</span>, <span>{{item.year}}</span></p>
                        <p>{{item.origin}}</p>
                    </especie-item-complejo>
            </template>
        </template>
</dom-module>

<script>
Polymer({
    is: "especie-lista",
    properties: {
      src: { type: String }
    }
```

```
});
</script>
```

El código del Web Component introduce nuevos conceptos: utiliza el elemento `<iron-ajax>`[25] de Polymer, que se encarga de obtener los datos solicitados vía AJAX[26] y una vez recibidos se los entrega al template por medio de `{{data}}`. También puede observarse que se ha reutilizado el componente del ejemplo anterior `<especie-item-complejo>` al que se le pasó la información iterada por `data.item`

Para utilizar este nuevo componente, sólo ha sido necesario escribir:

```
<especie-lista src="data/species.json"></especie-lista>
```

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/extender-elemento.html

Ejemplo 4. Cómo extender un elemento nativo

Hasta aquí se ha visto cómo crear componentes básicos, enviarles parámetros, incluir componentes dentro de otros y recibir datos de fuentes externas para iterar y construir listas. En este nuevo ejemplo se avanza en un caso típico de la especialización: la herencia.

La herencia facilita la creación de objetos a partir de otros ya existentes, esto implica que un elemento obtendrá el comportamiento, con sus métodos y atributos, de a quien esté especializando.

Polymer sólo soporta extensión para elementos de HTML nativo, aunque se prevé que esta capacidad sea incorporada para componentes propios o de terceros en el futuro cercano[27].

En este ejemplo se demuestra cómo crear un componente a partir del elemento nativo `<button>` al que se llamará `<button-extended>` y tendrá como particularidad la posibilidad de

[25] Polymer Catalog `<iron-ajax>` <https://elements.polymer-project.org/elements/iron-ajax> [Consulta: 10 Junio 2015]

[26] AJAX es un mecanismo de JavaScript que permite obtener información de un recurso externo. El elemento `<iron-ajax>` facilita ese proceso abstrayendo al desarrollador en los detalles de implementación del llamado AJAX.

[27] POLYMER (2015). *Extend native HTML elements* <https://www.polymer-project.org/1.0/docs/devguide/registering-elements.html#type-extension> [Consulta: 10 Junio 2015]

recibir un ícono acompañando al texto del botón. También se aprovechará la definición del nuevo componente para establecer una serie de estilos CSS que unifiquen este tipo de botones.

```
<dom-module id="button-extended">
        <style>
                :host { // Definición de estilos que se aplicarán al botón }
                :host img { // Estilos que se aplicarán a la imagen con el ícono }
        </style>
        <template>
                <img src="{{icon}}"/>
                <content></content>
        </template>
</dom-module>

<script>
Polymer({
        is: "button-extended",
        extends: 'button',
        properties: { ... }
});
</script>
```

Aparecen dos nuevos conceptos, el de `:host` y la herencia en sí misma, con `extends: 'button'`

El pseudo elemento `:host` se utiliza dentro de la hoja de estilos del componente para referirse al componente en sí, en tiempo de ejecución. Todo lo que se defina ahí afectará toda la estructura del componente.

La propiedad `extends` se utiliza para realizar la herencia y sólo está disponible para elementos nativos, como se mencionó más arriba.

La imagen muestra el componente en ejecución. Con gris el botón con su atributo `disabled` activo y en anaranjado cuando recibe el puntero del mouse.

Vale la pena destacar nuevamente que todos los atributos, métodos y eventos asociados al botón nativo (como `type="submit"` o `disabled`) han sido automáticamente heredados en la nueva definición.

Para utilizar el componente y crear el botón “navegar” que se ve en la imágen debe escribirse:

```
<button is="button-extended" icon="icons/ship.png" type="submit">Navegar</button>
```

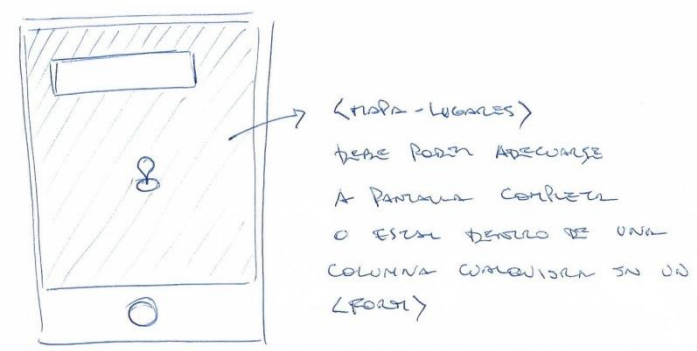
Nótese la importancia del atributo `is="button-extended"` que -concretamente- es el que indica que se trata de una especialización de `<button>`

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/iterar-datos.html

Ejemplo 5. Cómo articular datos entre componentes

Por su complejidad, los ejemplos anteriores no requirieron de mayores explicaciones para comprender lo que se buscaba como producto final. Tampoco se ahondó en detalles cualitativos para considerarlos Web Components listos para compartir y utilizar en producción, sino que se hizo hincapié en los aspectos pedagógicos de interés para la tesina.

En este caso el producto que se pretende lograr tiene cierta complejidad en comparación con los ya mencionados, y aunque tampoco se trata de un Web Component que se vaya a compartir en términos de producción, sí se harán ciertos avances en lo que respecta a su explicación y versatilidad.



Se han utilizado técnicas de Sketch (bocetos a mano alzada) para representar visualmente el producto esperado. Según el caso, este proceso podría detallarse en wireframes, mockups y prototipos.

Para despejar dudas sobre las diferencias y conceptos de estos cuatro estadíos del diseño, un vistazo rápido al artículo “Diferenciando un Sketch, Mockup, Wireframe y Prototipo”[28] puede ser de utilidad.

La comunicación entre componentes es una técnica de gran utilidad para generar aplicaciones interactivas, que sean sensibles a los cambios de información que responden a eventos generados por el usuario u otros derivados de diversas situaciones del sistema.

[28] OTHER WISE ONLINE (2013). *Diferenciando un Sketch, Mockup, Wireframe y Prototipo* <http://www.otherwiseonline.net/diferencias-entre-sketch-mockup-wireframe-prototipo/> [Consulta: 10 Junio 2015]

En el ámbito del desarrollo web, son bien conocidos los eventos que se disparan según la evolución de la aplicación y que pueden capturarse para aplicar acciones concretas en ese momento.

Por ejemplo, un elemento `<button>` tiene asociados eventos como `onclick`, que dispara cuando el usuario hace *click* con el mouse sobre el botón. Otro ejemplo es el componente nativo `<input>` que tiene asociado el evento `onblur`, que se ejecuta cuando el usuario abandona ese elemento y se posiciona en otro.

```
<input onblur="setLatLng()" type="text">
```

Cuando el usuario abandone el elemento `<input>` se ejecutará la función `setLatLng()` previamente definida.

El evento `onclick` se dispara gracias a una acción explícita del usuario -cuando hace *click* sobre el elemento en cuestión. Otro tipo de eventos son los que se activan cuando ocurren ciertos cambios a nivel de sistema:

```
document.addEventListener("DOMContentLoaded", function(event) {
    // acciones a ejecutar al activarse el evento
});
```

El código anterior crea un “escuchador de eventos” que reacciona cuando el contenido de la página haya finalizado de cargar.

Los componentes nativos tienen definidos sus eventos de forma nativa y también pueden agregárseles nuevos eventos definidos especialmente, de manera similar al ejemplo dado. Por otra parte, al crear componentes propios, es común crear y asignarles eventos.

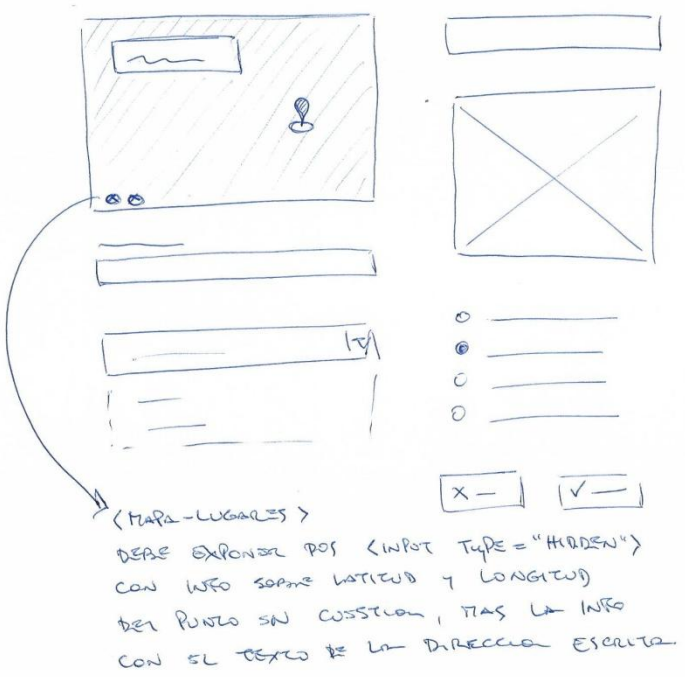
Existen distintas estrategias de comunicación entre componentes, Polymer propone cuatro:

- Vinculación automática de datos (data binding)
- Observadores de cambios (changed watchers)
- Eventos personalizados (custom events)
- Vinculación manual con eventos nativos (using an element’s API)

Eric Bidelman, uno de los creadores de Polymer, ha escrito un interesante resumen con ejemplos de las cuatro estrategias mencionadas, que puede ser accedido en https://www.polymer-project.org/0.5/articles/communication.html

En este último ejemplo de la tesina se trabajará con dos enfoques diferentes para resolver la comunicación entre componentes.

En el primer caso se ha creado un Web Component al que se llamará `<mapa-lugares>` y que tiene en su interior dos componentes: un `<input>` nativo y un mapa que utiliza la API de Google gracias al componente `<google-map>` de Polymer. Cuando el usuario modifique el domicilio en el `<input>`, el mapa cambiará la posición del marcador ubicándose en la nueva dirección postal. Se trata de un claro ejemplo de "vinculación de datos y comunicación dentro de un componente".



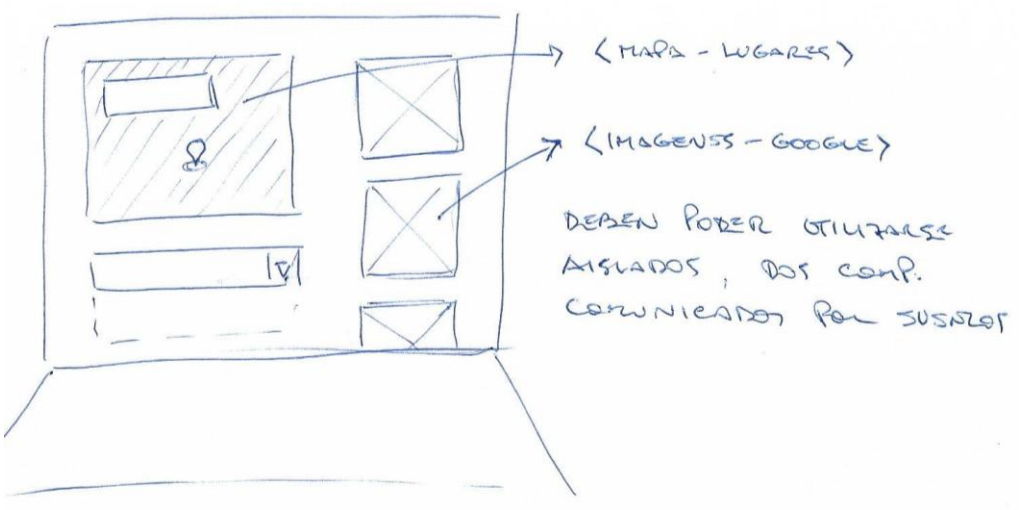
En el boceto se muestra un ejemplo de cómo podría utilizarse <mapa-lugares> para establecer la ubicación geográfica de un alojamiento hotelero en el marco de un formulario de carga con la información del hotel.

Podrían darse dos casos:

- Que alcance con escribir el domicilio para que el mapa ubique el marcador.

- Que el domicilio no sea reconocido (es común en cabañas alejadas de la ciudad) y entonces el usuario debería poder mover el marcador manualmente para ubicarlo correctamente, sin afectar lo que se haya escrito en el domicilio.

El segundo caso va más allá y "dialoga" con otros elementos externos, a lo que se llamará "vinculación de datos y comunicación entre distintos componentes". Así, se ha creado un componente (independiente del anterior) llamado `<imagenes-google>` que estará "escuchando" cambios en `<mapa-lugares>` y mostrará imágenes relativas al lugar indicado.



Vinculación de datos y comunicación dentro de un componente

A continuación se ejemplifica una de las maneras en que un Web Component puede hacer dialogar a sus elementos interiores -esto es, a los elementos por los que está formado.

La ventaja de este tipo de situaciones radica en que todo se agrupa en un solo componente y que es muy sencillo entonces incorporarlo en una aplicación y tener garantizado su correcto funcionamiento, ya que no se requiere de ningún código en particular más allá de la inclusión del elemento en el HTML.

Para incluir toda la capacidad de <mapa-lugares> en una aplicación web, sólo basta escribir:

```
<mapa-lugares latitude="-54.8019121" longitude="-68.302951"></mapa-lugares>
```

Incluso los parámetros latitude y longitude son opcionales.

El código Polymer de `<mapa-lugares>` es muy simple:

```
<template>
  <div class="component-container">
    <input id="domicilio" on-keypress="setLatLng" placeholder="Escribí un domicilio, ciudad o lugar">

    <google-map id="map" latitude="{{latitude}}" longitude="{{longitude}}" disable-default-ui>
      <google-map-marker id="marker" latitude="{{latitude}}" longitude="{{longitude}}"
        on-google-map-marker-mouseup="markerMoved" draggable="true" mouse-events="true">
      </google-map-marker>
    </google-map>

    <iron-signals on-iron-signal-new-location="newLocation">
```

```
  </div>
</template>
```

En amarillo se destacan dos secciones importantes del código.

La primera es de utilidad para iniciar la búsqueda de una locación y se activa con `on-keypress` que hace referencia al evento que se dispara cada vez que el usuario toca una tecla sobre el cuadro de texto `<input>` domicilio. Como HTML no cuenta con un evento on-press-enter la única manera de saber si la persona presiona enter, será capturando cada tecla.

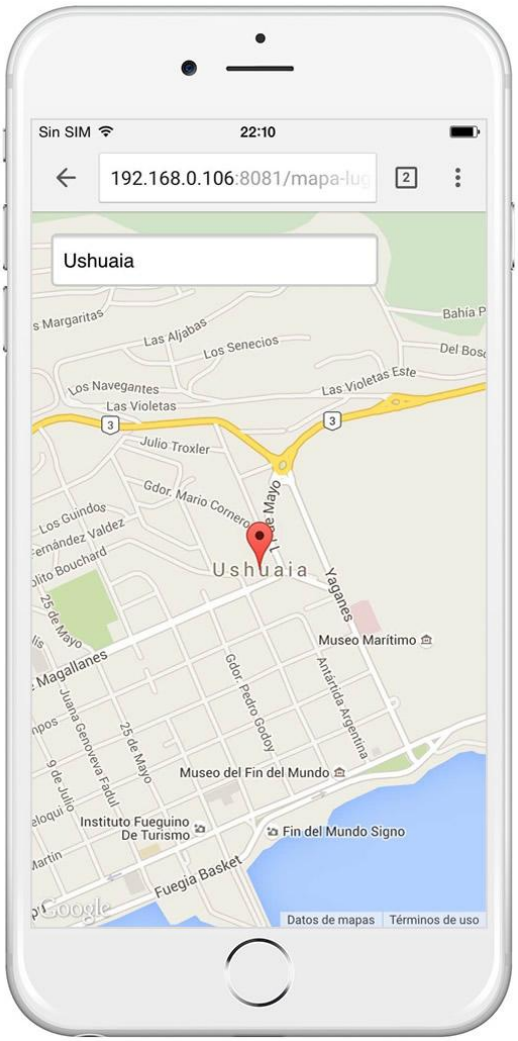
La segunda sección en amarillo utiliza un Web Component llamado `<iron-signals>` que llamará al método `newLocation()` cada vez que se dispare un evento del tipo "`new-location`" que se explica a continuación.

```
Polymer({
  is: "mapa-lugares",

  properties: {
     latitude: { type: Number },
     longitude: { type: Number }
  },

  setLatLng: function(event) {

    if(event.keyCode == 13) { // keyCode == 13 representa la tecla <enter>
      var me = this;
      var theAddress = this.$.domicilio.value;
      var geocoder = new google.maps.Geocoder();
      geocoder.geocode( { 'address': theAddress }, function(results, status) {
        if (status == google.maps.GeocoderStatus.OK) {
          var newLocation = {
            location: results[0].geometry.location,
            place: theAddress
          };
          me.fire('iron-signal', { name: "new-location", data: newLocation });
        }
      });
    }
  },

  // Se actualiza el mapa y el marcador con la nueva posición geográfica
  newLocation: function(event, detail, sender) {
    var map = this.$.map.map;
    var marker = this.$.marker.marker;
    map.setCenter(detail.location);
    marker.setPosition(detail.location);
```

```
  },

  // Cuando se mueve el marcador, se actualiza el mapa y la posición geográfica
  markerMoved: function(event) {
    var detail = { "location":  new google.maps.LatLng(this.latitude, this.longitude) };
    this.newLocation(event, detail, this.$.marker.marker);
  }
});
```

En la primera sección de código resaltado se puede observar cómo se utiliza la API de Google Maps para convertir un domicilio postal en una posición geográfica del tipo latitud y longitud.

Más adelante se indica el momento en que se lanza un evento del tipo `iron-signal` asociándole la nueva información geográfica y que será interceptado por el elemento `<iron-signals>` definido en `<template>` que finalmente ejecutará el método `newLocation()`.



En la imagen puede verse `<mapa-lugares>` mostrando la ciudad de Ushuaia.

El elemento creado está preparado para adaptarse a cualquier dispositivo, ya sea una computadora o un teléfono móvil y permite ser modificado en ancho y alto para adecuarse a distintas situaciones, no sólo ocupando toda la pantalla, sino también conviviendo con otros componentes.

Un caso de utilidad para <mapa-lugares> es guardar la ubicación geográfica de un domicilio postal garantizando que -aunque ese domicilio no sea reconocido por Google- la ubicación en el mapa sea la correcta, ya que siempre será posible manipular el marcador rojo para establecer una coordenada particular.

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/mapa-lugares.html

Vinculación de datos y comunicación entre distintos componentes

En este último ejemplo se expone una situación diferente: dados dos componentes individuales, se busca hacerlos dialogar a partir de la comunicación con eventos. A diferencia del caso anterior, se trata de componentes individuales, no de dos componentes agrupados dentro de otro.

El código HTML para instanciar los componentes sigue la línea de todo lo visto en los ejemplos anteriores.

```
<mapa-lugares></mapa-lugares>
<imagenes-google></imagenes-google>
```

No es necesario hacer nada más, los componentes sabrán escucharse y reaccionar a los eventos que se generen.

Una de las ventajas más importantes que este enfoque tiene, al tratarse de dos componentes individuales, es que pueden utilizarse en distintas ubicaciones de la página en cuestión separados por otros bloques de código.

Así, otra opción válida para utilizar ambos componentes podría ser:

```
<h1>Buscador de fotos y lugares</h1>
<mapa-lugares></mapa-lugares>
<h2>Las imágenes que se muestran están vinculadas al lugar que elegiste en el mapa</h2>
<imagenes-google></imagenes-google>
```

Por supuesto, la organización podría ser mucho más compleja y todo seguirá funcionando correctamente. Los componentes no requieren de una disposición en particular para dialogar.

La imagen muestra los Web Components `<imagenes-google>` y `<mapa-lugares>` en ejecución, sincronizados bajo la búsqueda de la ciudad de Ushuaia.

A continuación se detalla el código Polymer del Web Component `<imagenes-google>`

```
<template>
<iron-signals on-iron-signal-new-location="newLocation">
<iron-ajax id="images" last-response="{{data}}"
  url="https://ajax.googleapis.com/ajax/services/search/images">
</iron-ajax>
<div class="component-container">
    <template is="dom-repeat" items="{{data.responseData.results}}">
      <paper-material>
        <img src="{{item.url}}" alt="{{item.contentNoFormatting}}">
        <p>{{item.contentNoFormatting}}</p>
      </paper-material>
    </template>
</div>
</template>
```

Es fácil ver si similitud con todos los ejemplos anteriores. Sólo es necesario destacar el elemento `<iron-signals>` que, como en el caso previo, escuchará al evento "`new-location`" generado por `<mapa-lugares>` y aunque `<mapa-lugares>` sea un componente distinto e independiente, tiene capacidad de emitir eventos públicos que pueden ser escuchados por otros componentes, como es el caso de `<imagenes-google>`.

El código que sigue es también parte de la definición de `<imagenes-google>` y nuevamente se repite un esquema similar a los ya conocidos. El método `newLocation()` recibirá la información emitida por el evento "`new-location`" y a través del parámetro detail.place obtendrá el texto con el nombre de la localidad a buscar. Luego simplemente se acciona el llamado a la API de Google Images con `this.$.images.generateRequest();` para actualizar la lista de imágenes.

```
Polymer({
    is: "imagenes-google",

    properties: {
       location: { type: String }
    },

    newLocation: function(event, detail, sender) {
      var params = {
        "q": detail.place,
        "v": "1.0",
        "key": "AIzaSyA0fI_USpH-X0ev2tk-heuLGKBlruthp04",
        "rsz": 8,
        "imgsz": "medium"
      };

      this.$.images.params = params;
      this.$.images.generateRequest();
    }
});
```

Ejemplo funcionando: http://rawgit.com/fedegonzal/Web-Components/master/para-compartir/mapa-lugares-imagenes.html

Conclusiones y recomendaciones

La estandarización y aceptación de las tecnologías vinculadas a los Web Components son un excelente ejemplo de cuán difícil es para las compañías que desarrollan Browsers incorporar nuevas características. Existen grandes intereses corporativos para imponer estándares (en realidad, para estandarizar propuestas) y -en ese sentido- desde la presentación en 2011 hasta la fecha, Google ha hecho un esfuerzo destacable para acercar las partes con su propuesta.

Desde 2014 los Browsers más importantes ya han comenzado a soportar Web Components y todo indica que con el correr del tiempo, la recomendación del W3C terminará siendo un estándar global.

Así, como se ha podido conocer en este trabajo, la posibilidad de incorporar Web Components a un proyecto en producción depende en gran medida de las capacidades que tengan los Browsers que los usuarios utilicen –y claramente, no es un objetivo obligar a las personas a usar un navegador en particular.

Luego del estudio realizado en los capítulos “Estado del arte” y “Librerías para Polyfills y análisis de Browsers”, se ha llegado a la conclusión de que hoy en día ya es posible trabajar con Web Components, siempre que se utilicen Polyfills para facilitar la compatibilidad con los diferentes navegadores.

En los capítulos “Guía básica para el desarrollo de Web Components” y “Ejemplos para el desarrollo de componentes” se presentó la librería Polymer, que con el impulso de Google, se ha consolidado como la herramienta de Polyfill más avanzada y la única apta para producción.

En este último capítulo de la tesina se documentan una serie de buenas prácticas para considerar al momento de desarrollar y utilizar Web Components, muchas de estas recomendaciones se encuentran publicadas en el documento http://webcomponents.org/articles/web-components-best-practices/ que es mantenido comunitariamente, otras recomendaciones tienen que ver exclusivamente con la experiencia alcanzada durante el desarrollo de este trabajo.

Al desarrollar componentes

Nombres de etiquetas: la especificación de Custom Elements establece que los componentes deben crearse asignando al menos un guión en su nombre, el objetivo de esto es generar un marco para identificar conjuntos de componentes. Por ejemplo: `<google-map>` , `<google-calendar>` , `<bootstrap-navigation>` , `<flex-column>` , etc. Para no solapar nombres con otros desarrolladores, se recomienda usar prefijos con más de 3 caracteres.

Imitar a los elementos nativos: los componentes creados deben poder sentirse y usarse como cualquier componente nativo.

Usar eventos para intercambiar información: es la mejor forma de exponer cambios o datos para que otros componentes puedan obtenerlos y conocerlos. Al menos que hayan sido diseñados para trabajar exclusivamente juntos, dos componentes no deberían depender el uno del otro.

Incluir las dependencias: es imprescindible incluir todas las dependencias que el componente requiera, no importa que sean redundantes con otros, ya se ha visto que un mismo `<link rel="import" ...>` no será cargado dos veces.

No crear componentes similares: si la única diferencia entre dos componentes tiene que ver con un aspecto en la estructura visual de la misma información, debe considerarse consolidarlos en un solo elementos y crear dos templates, para alternar entre ellos con algún parámetro o configuración. Alternativamente, un elemento puede ser extendido para especializar su funcionamiento.

Armonizar la API del componente: los atributos forman parte de la API declarativa, los métodos y eventos son imperativos. Debe mantenerse una analogía entre ambas partes tanto como sea posible para que un desarrollador pueda utilizar cualquiera de los dos métodos, según su conveniencia.

No confiarse del contexto: si un componente es útil, probablemente vaya a ser usado por muchas personas, en contextos nunca antes esperados y en relación con componentes desconocidos. Debe mantenerse lo más encapsulado posible y a la vez flexible. De ser posible, evitar depender de frameworks y otras estructuras.

Accesibilidad: a las aplicaciones interactivas se las llama en inglés “Rich Applications” y al desarrollar componentes, necesariamente se los utilizará en este tipo de productos. La sigla ARIA hace referencia a las “Accessible Rich Internet Applications” y la W3C ha definido una

serie de recomendaciones para utilizar el prefijo aria-* como un atributo de HTML para comunicar funciones, estados y propiedades semánticas a las tecnologías de asistencia para la accesibilidad. Si se quiere desarrollar Web Components accesibles, es indispensable incorporar ARIA. En el capítulo 6 del libro “Mobile HTML5” citado en la bibliografía puede leerse más información y ejemplos. También en http://w3c.github.io/aria-in-html/#html5na

Performance: considerar que un componente poco eficiente puede entorpecer el normal funcionamiento de toda la aplicación, sobre todo al ser accedida con dispositivos pequeños como teléfonos. Para mayor información sobre performance, puede consultarse el libro “Pro HTML5 performance” citado en la bibliografía, otra buena lectura es http://www.smashingmagazine.com/2012/11/05/writing-fast-memory-efficient-javascript/

JavaScript no siempre es la respuesta: muchas estructuras y técnicas de estilos que se utilizaban hasta hace poco tiempo (como los bordes redondeados o las animaciones) hoy son fácilmente aplicables con CSS.

Adaptables: los componentes deben diseñarse con criterios de “responsive design”, adaptándose a su entorno. No es sólo cuestión de organización visual, sino también de adaptar funciones. Por ejemplo, el evento click no puede dejar de funcionar si el usuario tiene una tablet en su mano.

Mockups y Wireframes: siempre es bueno tener una idea clara de cómo se verá el componente y qué sucederá en diferentes situaciones. Nunca está demás hacer dibujos a mano alzada o utilizar herramientas digitales para bocetar esas situaciones.

Pruebas automatizadas: el desarrollo de un Web Component involucra grandes retos. Distintos Browsers, diferentes dispositivos y sistemas operativos, parámetros obligatorios y opcionales, componentes desconocidos que puedan influenciarlo y muchas cosas más. No se trata sólo de que su código “funcione”, sino de que funcione siempre, en cualquier situación. El testeo manual no alcanza, las herramientas de automatización como WCT de Polymer, servicios como Sauce Labs, Mocha y Selenium son indispensables.

Al compartir componentes

La documentación es clave: debe tenerse presente que al crear componentes su desarrollador es quien conoce qué parámetros necesita y cómo se comporta. El tiempo y el olvido nunca están a favor de quienes postergan la documentación. Enumerar y explicar sus propiedades, métodos y eventos.

Explicar los atributos: los parámetros que pueda recibir un componente siempre pueden ser explicados con ejemplos, una manera práctica de complementar la documentación. Si un componente es diseñado para trabajar con otro, es importante demostrarlo con un ejemplo.

Si algo falla, el silencio es oro: nuevamente, los componentes propios deben actuar como los nativos. Por lo tanto, si algo falla es importante no interrumpir el funcionamiento del resto de la aplicación. Por supuesto, cuanto más rigurosos sean los controles y las definiciones de tipos de datos al momento de desarrollar el componente, menores serán las chances de que haya problemas.

Automatización de tareas: en desarrollo web es común repetir tareas rutinariamente, mucho más cuando se trabaja con tecnologías “del lado del cliente”, como HTML, CSS y JavaScript. Grunt y Gulp son dos herramientas ideales para esto y deben considerarse como indispensables sobretodo para mantener automatizado el proceso de llevar a producción una versión en desarrollo.

Minimización y concatenación de recursos: otra cuestión vinculada con la automatización y algo altamente recomendable es la compresión de scripts y la concatenación de los recursos involucrados en un sólo archivo. Esto facilita la distribución del Web Component y permite que -en producción- se reduzcan significativamente la cantidad de solicitudes (requests) individuales por cada librería necesaria. Vulcanize es una buena herramienta para concatenar Web Components, Grunt y Gulp ofrecen plugins para minimizar scripts.

Publicar para distribución: el último paso para compartir un componente es disponerlo públicamente. Bower y NPM son las dos herramientas más utilizadas para el manejo de dependencias en el ámbito web, pero Bower en particular lo es para Web Components. En https://www.polymer-project.org/0.5/articles/distributing-components-with-bower.html hay un práctico tutorial que resume lo que debe tenerse en cuenta.

Bibliografía

Libros impresos

BRYANT, JAY; MIKE JONES (2012). *Pro HTML5 Performance*. Apress.

CRAVENS, JESSE; BURTOFT, JEFF (2012). *HTML5 hacks*. O'Reilly Media, Inc.

DE RYCK, PHILIPPE (2015). *Protected Web Components*. IEEE.

ELLIOTT, ERIC (2014). *Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries*. O'Reilly Media, Inc.

FAIN, YAKOV; RASPUTNIS, VICTOR; TARTAKOVSKY, ANATOLE; GAMOV, VIKTOR (2014). *Enterprise Web Development: Building HTML5 Applications: From Desktop to Mobile.* O'Reilly Media, Inc.

GASSTON, PETER (2013). *The Modern Web: Multi-device Web Development with HTML5, CSS3, and JavaScript.* No Starch Press.

HALES, WESLEY (2012). *HTML5 and JavaScript Web Apps.* O'Reilly Media, Inc.

NEWMAN, SAM (2015). *Building microservices.* O'Reilly Media, Inc.

OVERSON, JARROD; STRIMPEL, JASON (2015). *Developing Web Components. UI From jQuery to Polymer.* O'Reilly Media, Inc.

SATROM, BRANDON (2014). *Building Polyfills.* O'Reilly Media, Inc.

WEYL, ESTELLE (2013). *Mobile Html5.* O'Reilly Media, Inc.

Documentación Online

W3C. *Introduction to Web Components. Working Draft 6 June 2013*
<http://www.w3.org/TR/components-intro/> [Consulta: 10 de junio de 2015]

W3C. *Shadow DOM. Working Draft 14 May 2013*
<http://www.w3.org/TR/shadow-dom/> [Consulta: 10 de junio de 2015]

W3C. *HTML Templates. Working Group Note 18 March 2014*
<http://www.w3.org/TR/html-templates/> [Consulta: 10 de junio de 2015]

W3C. *HTML Imports. Working Draft 11 March 2014*
<http://www.w3.org/TR/html-imports/> [Consulta: 10 de junio de 2015]

W3C. *Custom Elements. Last Call Working Draft 24 October 2013*
<http://www.w3.org/TR/custom-elements/> [Consulta: 10 de junio de 2015]

W3C. *Introduction to Web Components. Editor's Draft 2 June 2014*
<http://w3c.github.io/webcomponents/explainer/> [Consulta: 10 de junio de 2015]

W3C. *Shadow DOM. Editor's Draft 02 June 2014*
<http://w3c.github.io/webcomponents/spec/shadow/> [Consulta: 10 de junio de 2015]

W3C. *HTML Imports. Editor's Draft 2 June 2014*
<http://w3c.github.io/webcomponents/spec/imports/> [Consulta: 10 de junio de 2015]

W3C. *Custom Elements. Editor's Draft 2 June 2014*
<http://w3c.github.io/webcomponents/spec/custom/> [Consulta: 10 de junio de 2015]

ZAYTSEV, JURIY. *ECMAScript 6 compatibility table*
<http://kangax.github.io/compat-table/es6/> [Consulta: 10 de junio de 2015]

Tutoriales y links de interés

A continuación se lista, de manera informal y sin un orden particular, una serie de recursos que puden ser de utilidad para quien tenga interés en ahondar sobre Web Components de manera práctica.

JSF y Web Components
http://www.nuxeo.com/blog/web-components-a-great-standard-for-a-platform-approach/

A Guide to Web Components
http://css-tricks.com/modular-future-web-components/

Polymer
http://www.polymer-project.org/

Introduction to Polymer: The Next Generation of Web Development
https://www.youtube.com/watch?v=8-Zq2KUN6jM

Web Components: A Tectonic Shift for Web Development
https://developers.google.com/events/io/sessions/318907648

WebComponents.org a place to discuss and evolve web component best-practices
http://webcomponents.org/

Are We Componentized Yet?
http://jonrimmer.github.io/are-we-componentized-yet/

Custom Elements. A web components gallery for modern web apps
http://customelements.io/

Web Components Resources
http://ebidel.github.io/webcomponents/

Custom Elements. Defining new elements in HTML
http://www.html5rocks.com/en/tutorials/webcomponents/customelements/

Visual Test-Driven Development For Responsive Interface Design
http://www.smashingmagazine.com/2015/04/07/visual-test-driven-development-responsive-interface-design/

Responsive Images in Practice
http://alistapart.com/article/responsive-images-in-practice

HTML5 Doctor. Helping you to implement HTML5 Today
http://html5doctor.com/

A future called Web Components
https://www.youtube.com/watch?v=XYlgxre_AF4

Web Components – The Future Web
http://thejackalofjavascript.com/web-components-future-web/

Unit Testing Polymer Elements. Authoring Unit Tests for your elements
https://www.polymer-project.org/0.5/articles/unit-testing-elements.html

Concatenating Web Components with Vulcanize
https://www.polymer-project.org/0.5/articles/concatenating-web-components.html

Distributing Components With Bower
https://www.polymer-project.org/0.5/articles/distributing-components-with-bower.html

Writing Fast, Memory-Efficient JavaScript
http://www.smashingmagazine.com/2012/11/05/writing-fast-memory-efficient-javascript/