MULTIMEDIA

Hasta hace no mucho tiempo, la tecnología *Flash* era el dominador indiscutible en el campo multimedia de la web. Gracias a esta tecnología es relativamente sencillo transmitir audio y vídeo a través de la red, y realizar animaciones que de otra manera sería imposible. Prácticamente todos los navegadores tienen incorporado un *plugin* para la reproducción de archivos *flash*, por lo que la elección era clara. **Entonces, ¿por qué una necesidad de cambio?**

Hasta ahora, para incluir un elemento multimedia en un documento, se hacía uso del elemento `<object>`, cuya función es incluir un elemento externo genérico. Debido a la incompatibilidad entre navegadores, se hacía también necesario el uso del elemento `<embed>` y duplicar una serie de parámetros. El resultado era un código de este estilo:

```
<object width="425" height="344">
    <param name="movie"
        value="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"></param>
<param name="allowFullScreen" value="true"></param>
    <param name="allowscriptaccess" value="always"></param>
    <embed src="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"
type="application/x-shockwave-flash"
allowscriptaccess="always"
        allowfullscreen="true" width="425" height="344"></embed>
</object>
```

Dejando de lado que el código es *poco amigable*, en este caso tenemos el problema que el navegador tiene que transmitir el vídeo a un plugin instalado en el navegador, con la esperanza que el usuario tenga instalada la versión correcta o tenga permisos para hacerlo, y muchos otros requisitos que pueden ser necesarios. Los *plugins* pueden causar que el navegador o el sistema se comporte de manera inestable, o incluso podemos crear una inseguridad a usuarios sin conocimientos técnicos, a los que se pide que descarguen e instalen un nuevo software.

De alguna manera, podemos estar creando una barrera para ciertos usuarios, algo que **no es para nada deseable**.

6.1 VÍDEO

Una de las mayores ventajas del elemento `<video>` (y `<audio>`) de HTML5, es que, finalmente, están totalmente integrados en la web. Ya no es necesario depender de *software* de terceros, y esto es una gran ventaja.

Así que ahora, el elemento `<video>` pueden personalizarse a través de estilos CSS. Se puede cambiar su tamaño y animarlo con transiciones CSS, por ejemplo. Podemos acceder a sus propiedades a través de JavaScript, transformarlos y mostrarlos en un `<canvas>`. Y lo mejor de todo, es que podemos manipularlos con total libertad, ya que pertenecen al estándar. Ya no se ejecutan en una *caja negra* a la que no teníamos acceso.

A pesar de que la etiqueta `<video>` se presenta como una alternativa de *flash*, sus funcionalidades van mucho más allá, como veremos a continuación.

6.1.1 MARCADO

Para hacer funcionar el vídeo en HTML, es suficiente con incluir el siguiente marcado, de manera similar que lo hacemos con las imágenes:

```
<video src="movie.webm"> </video>   <!-- Esto funciona en un mundo ideal -->
```

Sin embargo, este ejemplo realmente no hace nada por el momento. Lo único que se muestra es el primer fotograma de la película. Esto es así porque no le hemos dicho al navegador que inicie el vídeo, ni le hemos mostrado al usuario ningún tipo de control para reproducir o pausar el vídeo.

6.1.2 AUTOPLAY

Si bien podemos indicar al navegador que reproduzca el vídeo de manera automática una vez se haya cargado la página, en realidad no es una buena práctica, ya que a muchos usuarios les parecerá una práctica muy intrusiva. Por ejemplo, los usuarios de dispositivos móviles, probablemente no querrán que el vídeo se reproduzca sin su autorización, ya que se consume ancho de banda sin haberlo permitido explícitamente. No obstante, la manera de hacerlo es la siguiente:

```
<video src="movie.webm" autoplay>
    <!-- Your fallback content here -->
</video>
```

6.1.3 CONTROLS

Proporcionar controles es aproximadamente un 764% mejor que reproducir el vídeo de manera automática. La manera de indicar que se muestren los controles es la siguiente:

```
<video src="movie.webm" controls>
    <!-- Your fallback content here -->
</video>
```

Naturalmente, y al igual que ocurre con los campos de formulario, los navegadores muestran los controles de manera diferente, ya que la especificación no indica qué aspecto deben tener. De todas maneras, independientemente del aspecto, todos ellos coinciden en los controles a mostrar: reproducir/pausa, una barra de progreso y un control de volumen. Normalmente, los navegadores esconden los controles, y solamente aparecen al mover el ratón sobre el vídeo al utilizar el teclado para controlar el vídeo.

Figura 6.1 Controles de vídeo en Google Chrome

6.1.4 `POSTER`

El atributo `poster` indica la imagen que el navegador debe mostrar mientras el vídeo se está descargando, o hasta que el usuario reproduce el vídeo. Esto elimina la necesidad de mostrar una imagen externa que después hay que eliminar con JavaScript. Si no se indica este atributo, el navegador muestra el primer fotograma del vídeo, que puede no ser representativo del vídeo que se va a reproducir.

Figura 6.2 Fotograma `poster` del vídeo anterior

6.1.5 `MUTED`

El atributo `muted`, permite que el elemento multimedia se reproduzca inicialmente sin sonido, lo que requiere una acción por parte del usuario para recuperar el volumen. En este ejemplo, el vídeo se reproduce automáticamente, pero sin sonido:

```
<video src="movie.webm" controls autoplay loop muted>
    <!-- Your fallback content here -->
</video>
```

6.1.6 `HEIGHT`, `WIDTH`

Los atributos `height` y `width` indican al navegador el tamaño del vídeo en pixels. Si no se indican estas medidas, el navegador utiliza las medidas definidas en el vídeo de origen, si están disponibles. De lo contrario, utiliza las medidas definidas en el fotograma poster, si están disponibles. Si ninguna de estas medidas está disponible, el ancho por defecto es de `300 pixels`.

Si únicamente se especifica una de las dos medidas, el navegador automáticamente ajusta la medida de la dimensión no proporcionada, conservando la proporción del vídeo.

Si por el contrario, se especifican las dos medidas, pero no coinciden con la proporción del vídeo original, el vídeo no se deforma a estas nuevas dimensiones, sino que se muestra en formato letterbox manteniendo la proporción original.

6.1.7 LOOP

El atributo `loop` indica que el vídeo se reproduce de nuevo una vez que ha finalizado su reproducción.

6.1.8 PRELOAD

Es posible indicar al navegador que comience la descarga del vídeo antes de que el usuario inicie su reproducción.

```
<video src="movie.webm" controls preload>
    <!-- Your fallback content here -->
</video>
```

Existen tres valores definidos para `preload`. Si no indicamos uno en concreto, es el propio navegador el que decide qué hacer. Por ejemplo, en un dispositivo móvil, el comportamiento por defecto es no realizar ninguna descarga hasta que el usuario lo haya indicado. Es importante recordar que un desarrollador no puede controlar el comportamiento de un navegador: `preload` es un *consejo*, no un comando. El navegador tomará una decisión en función del dispositivo, las condiciones de la red y otros factores.

- `preload=auto`: se sugiere al navegador que comience la descarga.
- `preload=none`: se sugiere al navegador que no comience la descarga hasta que lo indique el usuario.
- `preload=metadata`: este estado sugiere al navegador que cargue los metadatos (dimensiones, fotogramas, duración...), pero no descarga nada más hasta que el usuario lo indique.

6.1.9 SRC

Al igual que en elemento `<img>`, el atributo `src` indica la localización del recurso, que el navegador debe reproducir si el navegador soporta el *codec* o formato específico. Utilizar un único atributo `src` es únicamente útil y viable en entornos totalmente controlados, donde conocemos el navegador que accede al sitio web y los *codecs* que soporta.

Sin embargo, como no todos los navegadores pueden reproducir los mismos formatos, en entornos de producción debemos especificar más de una fuente de vídeo.

6.2 CODECS, LA NUEVA GUERRA

En los primeros borradores de la especificación de HTML5, se indicaba que se debía de ofrecer soporte para al menos dos *codecs* multimedia: Ogg Vorbis para audio y Ogg Theora para vídeo. Sin embargo, estos requisitos fueron eliminados después de que Apple y Nokia se opusieran, de modo que la especificación no recomendase ningún *codec* en concreto. Esto ha creado una situación de fragmentación, con diferentes navegadores optando por diferentes formatos, basándose en sus ideologías o convicciones comerciales.

Actualmente, hay dos *codecs* principales que debemos tener en cuenta: el nuevo formato WebM, construido sobre el formato VP8 que Google compró y ofrece de manera libre, y el formato MP4, que contiene el *codec* propietario H.264.

	WEBM	**MP4**	**OGV**
Opera	Sí	No	Sí
Firefox	Sí	Sí	Sí
Chrome	Sí	No	Sí

IE9+	No	Sí	No
Safari	No	Sí	No

WebM funciona en IE9+ y Safari si el usuario ha instalado los codec de manera manual.

Por lo tanto, la mejor solución en estos momentos es ofrecer tanto el formato libre WebM, como el propietario H.264.

6.2.1 MULTIPLES ELEMENTOS `<SOURCE>`

Para poder ofrecer ambos formatos, primeramente debemos codificarlos por separado. Existen diversas herramientas y servicios on-line para realizar esta tarea, pero quizás el más conocido sea Miro Video Converter. Este software, disponible para Windows y Mac, nos permite convertir los vídeos en formato Theora, o H.264 (y muchos otros) optimizados para diferentes tipos de dispositivos como iPhone, Android, PS2, etc.

Una vez dispongamos el vídeo en los distintos formatos, es necesario indicar todas las localizaciones de estos formatos, para que sea el navegador el que decida que formato reproducir. Evidentemente, no podemos especificarlos todos dentro del atributo `scr`, por lo que tendremos que hacerlo de manera separada utilizando el elemento `<source>`.

```
<video controls>
    <source src="leverage-a-synergy.mp4" type='video/mp4; codecs="avc1.42E01E,
mp4a.40.2"'>
    <source src="leverage-a-synergy.webm" type='video/webm; codecs="vp8, vorbis
"'>
    <p>Your browser doesn't support video.
        Please download the video in <a href="leverage-a-
synergy.webm">webM</a>         or <a href="leverage-a-synergy.mp4">MP4</a>
format.
    </p>
</video>
```

6.2.2 MEDIA QUERIES PARA `VIDEO`

Los ficheros de vídeo tienden a ser pesados, y enviar un vídeo en alta calidad a un dispositivo con un tamaño de pantalla reducido es algo totalmente ineficiente. No hay ningún inconveniente en hacerlo, pero si comprimimos el vídeo y disminuimos sus dimensiones, conseguiremos reducir su tamaño (en `MB`), algo de agradecer en entornos móviles y que dependen de una conexión de datos.

HTML5 permite utilizar el atributo `media` en el elemento `<source>`, ofreciendo la misma funcionalidad que los `Media Queries`en CSS3. Por lo tanto, podemos consultar al navegador por el ancho de la pantalla, la relación de aspecto, colores, etc, y definir el video correcto según las características del dispositivo.

```
<video controls>
    <source src="hi-res.mp4" media="(min-device-width: 800px)">
    <source src="lo-res.mp4">
</video>
```

6.3 API MULTIMEDIA

Los elementos multimedia `<video>` y `<audio>` ofrecen un API JavaScript muy completo y fácil de utilizar. Los eventos y métodos de los elementos de audio y vídeo son exactamente los mismos, su única diferencia se da en los atributos. A continuación se muestra una tabla con el API actual:

Atributos	Métodos	Eventos
error state	load()	loadstart
error	canPlayType(type)	progress
network state	play()	suspend
src	pause()	abort
currentSrc	addTrack(label, kind, language)	error
networkState		emptied
preload		stalled
buffered		play
ready state		pause
readyState		loadedmetadata
seeking		loadeddata
controls		waiting
controls		playing
volume		canplay
muted		canplaythrough
tracks		seeking
tracks		seeked
playback state		timeupdate
currentTime		ended
startTime		ratechange
muted		
paused		
defaultPlaybackRate		

playbackRate
played
seekable
ended
autoplay
loop
width [video only]
height [video only]
videoWidth [video only]
videoHeight [video only]
poster [video only]

Gracias a JavaScript y a este nuevo API, tenemos el control completo sobre los elementos multimedia. Esto significa que podemos crearnos nuestros propios controles, extendiendo los que nos ofrece el navegador. Un simple ejemplo de acceso al API del elemento `video`:

```
video.addEventListener('canplay', function(e)
{     this.volume = 0.4;     this.currentTime
= 10;     this.play();
}, false);
```

6.4 FULLSCREEN VIDEO

Flash ha ofrecido un modo de pantalla completa durante muchos años a la que muchos navegadores se han *resistido*. La razón principal es la seguridad; ya que si se fuerza a una aplicación o web a funcionar a pantalla completa, el usuario pierde el control del propio navegador, la barra de tareas y controles estándar del sistema operativo. Puede que el usuario no sepa después volver del modo de pantalla completa, o puede haber problemas de seguridad relacionados, como la simulación del sistema operativo, petición de password, etc.

Este nuevo API establece un único elemento *full-screen*. Está pensado para imágenes, video y juegos que utilizan el elemento `canvas`. Una vez que un elemento pasa a pantalla completa, aparece un mensaje de forma temporal para informar al usuario de que puede presionar la tecla `ESC` en cualquier momento para volver a la ventana anterior.

Las principales propiedad, métodos y estilos son:

- `element.requestFullScreen()`: hace que un elemento individual pase a pantalla completa.

```
document.getElementById("myvideo").requestFullScreen().
```

- `document.cancelFullScreen()`: sale del modo pantalla completa y vuelve a la vista del documento.
- `document.fullScreen`: devuelve `true` si el navegador está en pantalla completa.
- `:full-screen`: se trata de una pseudo-clase CSS que se aplica a un elemento cuando está en modo pantalla completa.

Además, podemos modificar los estilos del elemento utilizando CSS:

```
#myelement
{
    width: 500px;
}
#myelement:full-screen
{
    width: 100%;
}
#myelement:full-screen img
{
    width: 100%;
}
```

6.5 AUDIO

El elemento multimedia `audio` es muy similar en cuanto a funcionalidad al elemento `video`. La principal diferencia existe al indicar el atributo `controls` o no. Si lo especificamos, el elemento se mostrará en la página juntamente con los controles. Si no lo hacemos, el audio se reproducirá, pero no existirá ningún elemento visual en el documento. Por supuesto, el elemento existirá en el DOM y tendremos acceso completo a su API desde JavaScript.

6.5.1 MARCADO

Para hacer funcionar el audio en HTML, al igual que con el video es suficiente con incluir lo siguiente:

```
<audio src="audio.mp3">
</audio>
```

Los formatos soportados por los navegadores son los siguientes:

	MP3	MP4	WAV	OGG
Opera	No	No	Sí	Sí
Firefox	No	No	Sí	Sí

Chrome	Sí	Sí	Sí	Sí
IE9+	Sí	Sí	No	No
Safari	Sí	Sí	Sí	No

Por lo tanto, la mejor solución en estos momentos es ofrecer tanto el formato libre OGG, como el propietario MP3, marcado de la siguiente manera:

```
<audio controls>
    <source src="audio.ogg" type="audio/ogg">
    <source src="audio.mp3" type="audio/mpeg"> </audio>
```