EL ESTÁNDAR HTML5

Manual resumido de transición a HTML5.

El estándar HTML5.

Manual resumido de transición a HTML5.

Por Juan José Salvador Piedra.

IES Celia Viñas 2015-2016

2º Ciclo Formativo Grado Superior Desarrollo de Aplicaciones Web.

Índice de contenido

1. Introducción: el nuevo estándar HTML5

HTML5 (*HyperText Markup Language*, versión 5) es la quinta revisión importante del lenguaje básico de la World Wide Web,HTML. HTML5 especifica dos variantes de sintaxis para HTML: una «clásica», HTML, conocida como *HTML5*, y una variante XHTMLconocida como sintaxis *XHTML5* que deberá servirse con sintaxis XML. Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo. La versión definitiva de la quinta revisión del estándar se publicó en octubre de 2014.

Esta nueva versión de HTML conforma un nuevo estándar en la web, incorporando asimismo una integración más completa con CSS3 y una serie de nuevas APIs (*Application Programming Interface*) en JavaScript, orientadas a depender cada vez menos de frameworks y librerías externas para un desarrollo adaptado a la web 3.0 (web semántica), además de incorporar sustanciales e interesantes novedades.

Dentro de este nuevo estándar podremos encontrar mejoras en el manejo de eventos, elementos HTML eliminados que han pasado a depender completamente de CSS, nuevos elementos y atributos como canvas o audio, además de una mejora sustancial en la forma de organizar la estructura de una página, ayudándonos así a no depender de la etiqueta div para muchos elementos comunes.

La potencia de HTML5 junto a los nuevos requisitos del mercado emergente del desarrollo web, nos brindan posibilidades infinitas, tales como el desarrollo de aplicaciones móviles basadas en HTML5 y JavaScript (aplicaciones multiplataforma usando código nativo y web), llegando incluso a desbancar al anteriormente aclamado Adobe Flash.

2. Documentos HTML5

El estándar HTML5 incluye una novedad de suma importancia: la web 3.0, o **web semántica**[1]. Esta mejora, se basa en la idea de añadir elementos semánticos al código que la máquina interpreta, con el fin de hacerlo más legible al humano que lo desarrolla.

Con la web semántica, HTML5 trae además una serie de etiquetas semánticas que reducen significativamente el uso de en nuestro código, designando etiquetas predefinidas para elementos comunes, como la cabecera, el pié de página, barra de navegación, etc.

Este cambio define un nuevo modelo de estructurado del cuerpo de nuestra página, manteniendo la metaestructura <html>, <head> y <body>, pero incluyendo nuevos elementos dentro de este último.

Ejemplo

En lugar de esto…

```
<div id=”header>>
     <h1>Mi página web</h1>
     <h2>Inicio</h2>
</div>
```

...tendríamos esto.

```
<header>
     <h1>Mi página web</h1>
     <h2>Inicio</h2>
</header>
```

1 Wikipedia- https://es.wikipedia.org/wiki/Web_semántica

2.1 Nueva estructura de layout

Ejemplo de documento HTML5 sencillo

```
<!DOCTYPE html>
<html>
    <head>
        <title>¡HTML5 mola!</title>
        <meta charset="utf8">
    </head>

    <body>
        <header>
            Esto es una cabecera
        </header>

        <nav>
            <ul>
                <li><a href="/">Inicio<a></li>
                <li><a href="www.google.es">Google</a></li>
            </ul>
        </nav>

        <aside>
            Sidebar
        </aside>
        <article>
            <section>Título del post</section>
            <section>Contenido del post</section>
        </article>

        <footer>
            Esto es un pie de página
        </footer>
    </body>

</html>
```

Ilustración 1: Código HTML5

Como puede verse en el ejemplo, se usan las nuevas etiquetas `<header>, <nav>, <aside>, <article>, <section>` y `<footer>` para designar la cabecera, una barra de navegación, la barra lateral, un artículo y pié de página.

Al renderizar la página, divide cada elemento en un bloque diferente, sin necesidad de usar div. Sin embargo, hasta que no le pasemos un estilo CSS, no posicionaremos cada elemento.

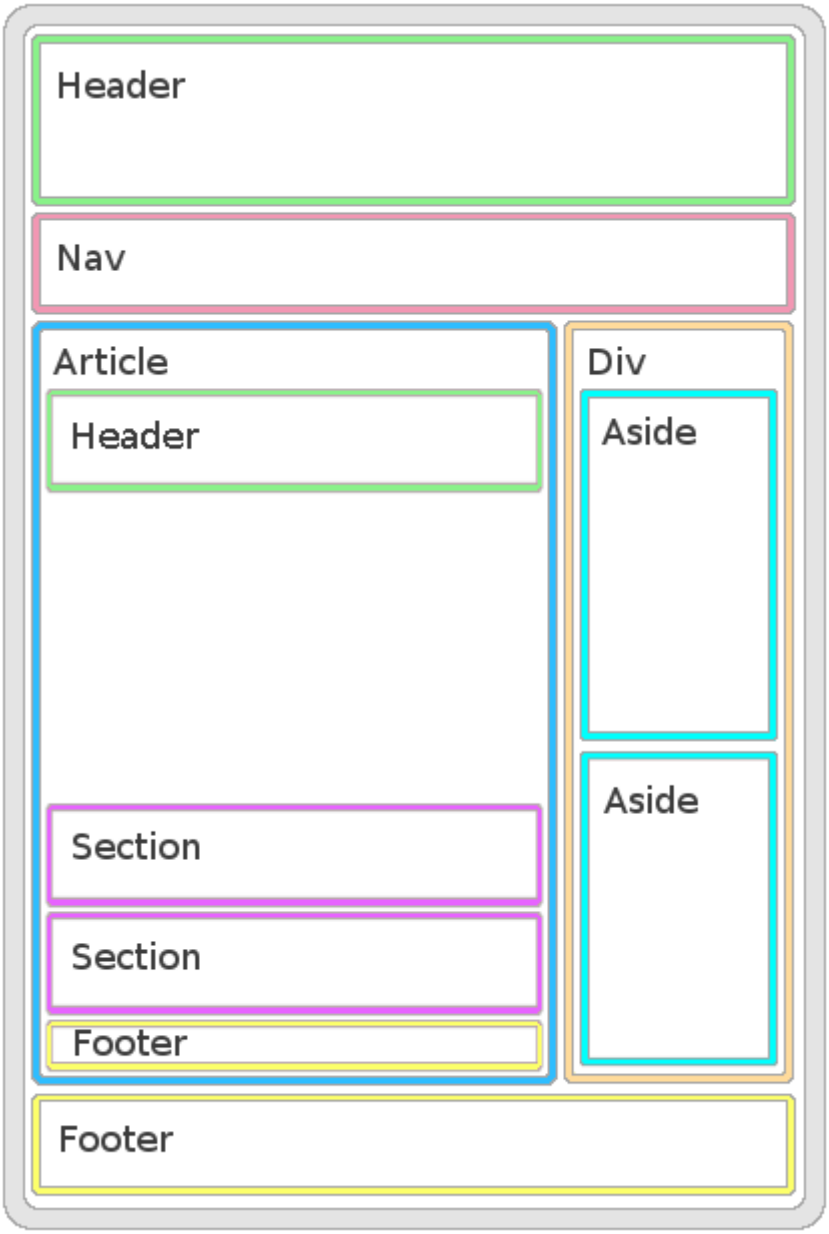

Ilustración 2: Esquema de la estructura HTML5

- **header** indica la cabecera de la página, donde generalmente se incluyen datos sobre en qué sección está el visitante, o una introducción. Se refieren siempre al elemento padre, pudiendo derivar del `body` o de un `article`. No se limita a los h1-6, sino que puede contener cualquier elemento. Es recomendable añadir solamente uno por página o sección.

  Ejemplo

```
<header>
     <h1>Mi página web</h1>
     <h2>Inicio</h2>
     <div id=”eslogan”>Eslogan chachi de lo web</div>
</header>
```

- **nav** representa la parte de la página desde la cual, podremos navegar por las diferentes secciones. Puede contener enlaces o cualquier otro elemento de texto.

  Ejemplo

```
<nav>
     <div class=”no-link”>Enlace no disponible</div>
     <a href=”#”>Enlace</div>
</nav>
```

- **article** representa los elementos autocontenidos de la página, es decir, aquellos que son independientes del resto de la página, como puede ser un artículo de un blog, un mensaje de un foro, etc.

  El elemento article puede contener a su vez un header, varios section y un footer.

  Ejemplo

```
<article>
          <header>
               <h3>Titulo del post<h3>
          </header>

          <section class=”post”>
               Contenido
          </section>

          <footer>
               <p>Pié de página</p>
          </footer>
</article>
```

- **section** simboliza una sección genérica del contenido. En este contexto, podemos hablar del contenido del artículo o mensaje, del título de este (fuera del header), etc. Puede incluir cualquier contenido, aunque es recomendable que excluyamos headings y otros elementos de cabecera o pié de página.

Ejemplo

```
<section class="date">Fecha del post</section>
<section><p>Contenido del post</p></section>
```

- **aside** representa las secciones de una página que, aunque forman parte del contenido, apartamos del contenido principal, como las sidebar (barras laterales) de una web. Pueden contener la navegación, imágenes, textos, artículos…

Ejemplo

```
<div id="sidebar">
     <aside>
          <p>Párrafo en la barra lateral</p>
     </aside>

     <aside>
          <img src="images/imagen.png" alt="Imagen"/>
     </aside>
</div>
```

- **footer** es el elemento que se coloca al final de la página. Si el header va en vanguardia, el footer va retaguardia. Generalmente contiene información sobre el motor de la web, el autor, propietarios, copyright, etc. Puede contener cualquier contenido, igual que header.

Ejemplo

```
<footer>
     <p>Pié de página</p>
     <a href="#">Enlace a la web del autor</a>
</footer>
```

2.2 Renovado de etiquetas: eliminadas y nuevas

En el estándar HTML5 se han añadido nuevas etiquetas y elementos, sin embargo, muchas otras designadas como obsoletas han sido eliminadas de este nuevo estándar, y sustituidas en su lugar por selectores y reglas CSS, proporcionando mayor libertad a la hora de diseñar una web. Algunas simplemente han sido sustituidas por una nueva etiqueta más clara que su predecesora, como es el caso de **<b>** y ****.

En esta sección hablaremos sobre las etiquetas añadidas, las eliminadas y las que han cambiado. No obstante, además de las mencionadas aquí, existen algunas más etiquetas de las que hablaremos en páginas siguientes, como son <audio>. <video> y <svg>.

Etiquetas nuevas

La siguiente lista de etiquetas representan una serie de nuevas etiquetas semánticas añadidas en el estándar HTML5.

Etiqueta	Descripción
<bdi>	Define una parte del texto que debe mostrarse en sentido contrario (izquierda-derecha o derecha-izquierda).
<details>	Define detalles adicionales que el usuario puede ver.
<dialog>	Define un cuadro de diálogo o ventana.
<figcaption>	Define la leyenda de un elemento <figure>.
<figure>	Define un elemento autocontenido (similar a <article>) con imágenes, citas, etc.
<main>	Define el contenido principal de una página.
<mark>	Define texto marcado o resaltado.
<menuitem>	Define un comando o menú que el usuario puede invocar desde una ventana emergente.
<progress>	Define el progreso de una tarea.
<summary>	Define una cabecera visible para los elementos <details>.
<time>	Define una fecha/hora.

<wbr>	Define un posible salto de línea.

Etiquetas eliminadas

Lista de etiquetas no soportadas por HTML5. Aunque el navegador puede seguir interpretándolas, su uso no está aceptado ni recomendado.

Etiqueta	Descripción
<acronym>	Asigna un acrónimo a un texto
<applet>	Inserta un applet de Java en HTML
<basefont>	Especifica un color y tipo de letra por defecto
<big>	Hace un texto mayor de lo normal
	Centra un texto
<dir>	Lista de contenidos de un directorio
<font>	Define la fuente, tamaño y color de un texto (sin CSS).
<noframes>	Designa qué mostrar cuando el navegador no soporta frames
<strike>	Define un texto tachado
<u>	Define un texto subrayado

Etiquetas que han cambiado

Algunas etiquetas simplemente han cambiado de nombre para pasar a ser etiquetas semánticas.

Etiqueta	Descripción
<i>	Define un texto en cursiva. En su lugar, se usa
<b>	Define un texto en negrita. En su lugar, se usa
<xmp>	Define un texto preformateado. En su lugar se usa <pre>

3. Multimedia

Una de las más significativas mejoras de HTML5, es la posibilidad de renderizar y visualizar contenido multimedia, como imágenes vectoriales, audio y vídeo, desde el navegador, de forma nativa, sin flash ni contenidos de terceros.

Todas estas nuevas etiquetas pueden ser controladas a través de JavaScript, permitiendo al desarrollador definir sus propios métodos de control de audio, vídeo o imágenes.

Para este cometido, podemos utilizar tres de las nuevas etiquetas añadidas al estándar.

Ejemplo

```
<audio controls>
     <source src=”play.ogg” type=”audio/ogg”>
     Tu navegador no soporta esta característica
</audio>

<video>
     <source src=”intro.mp4” type=”video/mpeg”>
     Tu navegador no soporta esta característica
</video>
```

3.1 Etiqueta Audio

La etiqueta <audio> nos permite insertar sonidos en nuestro documento web siguiendo un estándar. Antes de HTML5, no existía ningún estándar que permitiese esto, por lo que, la única forma de hacerlo era mediante Flash.

Esta etiqueta además, debe ir siempre acompañada de la etiqueta complementaria <source>, que indica la ruta y el tipo de audio que va a reproducir.

Atributo	Descripción
autoplay	Reproduce el archivo en cuanto se carga el arbol DOM
controls	Añade un panel de control definido por el navegador para controlar el audio.

La etiqueta <source> tiene como atributos `src` (ruta) y `type` (tipo MIME). Se pueden indicar varios <source>, de forma que reproducirá siempre el primero disponible. Si el cliente no pude reproducirlo, saltará al siguiente.

Formato	Tipo MIME
MP3 (.mp3)	audio/mpeg
OGG Vorbis (.ogg)	audio/ogg
WAV .(wav)	audio/wav

Ejemplo

```
<audio controls>
     <source src=”play.ogg” type=”audio/ogg”>
     <source src=”play.mp3” type=”audio/mp3”>
     Tu navegador no soporta esta característica
</audio>
```

3.2 Etiqueta Video

Igual que con la etiqueta <audio>, hasta la llegada de HTML5 no había forma de reproducir vídeo directamente en el navegador sin recurrir a flash o insertar vídeo desde fuentes externas. Mediante este nuevo estándar podemos insertar vídeos en formato MP4, WebM y OGG Video.

Su funcionamiento es exactamente igual que el de la etiqueta <audio>: necesita un <source> para reproducir, y un respaldo por si el cliente no puede reproducirlo. Añade además la etiqueta <track>, la cual inserta una pista de texto en el vídeo (como subtítulos).

Atributo	Descripción
autoplay	Reproduce el archivo en cuanto se carga el arbol DOM
controls	Añade un panel de control definido por el navegador para controlar el vídeo.
height	Define la altura del reproductor (no la del vídeo)
Width	Define la anchura del reproductor (no la del vídeo)

Formato	Tipo MIME
MP4 (.mp4)	video/mp4
OGG Vorbis (.ogg)	video/ogg
WebM (.webm)	video/webm

Ejemplo

```
<video controls>
     <source src=”bigbuckbunny.ogg” type=”video/ogg”>
     <source src=”bigbuckbunny.mp4” type=”video/mp4”>
     <track src="sub_esp.srt" kind="subtitles" srclang="es"
label="Español">
     Tu navegador no soporta esta característica
</video>
```

3.3 Etiqueta SVG

El lenguaje SVG, es utilizado para describir gráficos 2D usando XML. En HTML5 podemos usarlo para dibujar directamente en el navegador sin necesidad de que las imágenes estén almacenadas en un directorio, haciéndolas escalables y accesibles más fácilmente.

La etiqueta SVG, crea un contenedor donde se renderizará la imagen SVG que añadamos dentro.

Ejemplo

```
<svg width="100" height="100">
     <!-- Circulo amarillo con borde de 4 px verde, 100x100
píxeles -->
     <circle cx="50" cy="50" r="40" stroke="green"
     stroke-width="4" fill="yellow" />
</svg>
```

En HTML5, para renderizar gráficos, también podemos usar la etiqueta <canvas>, sin embargo, hay ciertas situaciones en las que es conveniente usar gráficos vectoriales en lugar de mapas de bits, como canvas.

3.4 Plugins

Además de todo lo mencionado en apartados anteriores, HTML5 soporta la inserción de elementos externos que no son audios o vídeos estándar, ni imágenes vectoriales. Por ejemplo, elementos Flash, un lector de PDF, o applets Java.

Para esta tarea, podemos echar mano de las etiquetas <object> y <embed>.

La etiqueta <object> nos permite insertar cualquier elemento. La soportan la mayoría de navegadores.

Ejemplo

```
<!-- Objeto flash -->
<object width="400" height="50" data="flash.swf"></object>

<!-- Imagen -->
<object data="imagen.jpg"></object>

<!-- Documento HTML -->
<object width="100%" height="500" data="formulario.html"></object>
```

En contraparte, la etiqueta <embed> (que era soportada antes de HTML5, pero no se encontraba dentro del estándar), permite insertar también objetos en la web la misma forma que usamos <object>, solo que esta, no tiene cierre, por lo que no permite insertar un mensaje si el contenido no puede mostrarse.

Ejemplo

```
<!-- Objeto flash -->
<embed width="400" height="50" src="flash.swf">

<!-- Imagen -->
<embed src="imagen.jpg">

<!-- Documento HTML -->
<embed width="100%" height="500" src="formulario.html">
```

4. Formularios HTML5

HTML5 incorpora al estándar una serie de nuevos atributos para la etiqueta <form> y la etiqueta <input>, permitiendo así manejar de forma más natural nuestros formularios-

Estos nuevos atributos permiten definir nuevos campos de entrada de datos más semánticos, además de permitirnos validar un formulario sin necesidad de mucho código JavaScript, usando el API Forms incluida en este nuevo estándar.

Ejemplo

```
<form>

     <input type=”text”
          name=”clientName”
          autocomplete=”name”
          pattern=”[A-Za-z]+s\[A-Za-z]”
          autofocus
          required />

     <input type=”email”
          name=”clientEmail”
          autocomplete=”email”
          pattern="@misitio\.com$"
          required />

     <input type=”submit” name=”send” />

</form>
```

4.1 Nuevos atributos

La etiqueta <input> se ha renovado añadiendo 16 atributos nuevos, siendo la mayoría booleanos, lo cual significa que no necesitan que se especifique un valor: si está presente, se toma como true, si no, como false.

Estos atributos nos permite, de forma nativa en HTML, limitar los datos introducidos por el usuario y validar directamente el formulario en el navegador, sin JavaScript (ver 4.3)

Atributo	Descripción
`autocomplete`	Proporciona una pista de autocompletado al navegador. Puede colocarse también en <form>, de forma que afecta a todo el formulario.
`autofocus`	Coloca el foco en este campo automáticamente cuando carga la página
`form`	Indica el ID del formulario al que pertenece este input, de modo que podemos colocarlo en cualquier lugar de la página.
`formaction`	Especifica la URI donde enviaremos el formulario al darle a submit. Sobreescribe el `action` especificado en <form>
`formenctype`	Especifica el encriptado que usaremos para enviar el formulario. Sobreescribe el utilizado en <form>
`formmethod`	Especifica el método de envío (POST o GET). Sobreescribe el usado en <form>
`formnovalidate`	Especifica que los datos no deben ser validados en el navegador al enviarse
`formtarget`	Especifica una ventana o frame objetivo para sobreescribir el target del <form>
`height y width`	Especifica el tamaño en alto y ancho del campo. Solo se usa con los campos de tipo imagen.
`list`	Apunta a una datalist (véase punto 4.2)
`min y max`	Define el número mínimo y máximo de caracteres soportados.

`multiple`	Especifica que pueden añadirse más de un valor. Se usa con los tipos email y file.
`pattern`	Define un patrón (regex) que usará para validar el formulario.
`placeholder`	Texto que aparecerá por defecto dentro del campo, y que desaparecerá al hacer clic sobre el.
`required`	Especifica que el campo es requerido. Necesario para validar el formulario.
`step`	Define un valor gradual.

Un campo `<input>` puede tener tantos atributos como hagan falta.

Ejemplo

```
<input type=”text”
     name=”clientName”
     autocomplete=”name”
     pattern=”[A-Za-z]+s\[A-Za-z]”
     autofocus
     required />
```

4.2 Nuevos campos

Los nuevos formularios también poseen nuevos tipos de campos de entrada de datos, tanto `<input>` como etiquetas nuevas. El uso de estas nuevas etiquetas es puramente semántico, facilitando la distribución de la web y el posterior paso de validación en el navegador (si está disponible) evadiendo el uso de JavaScript.

Nuevos tipos de <input>

Tipo	Descripción
email	Introduce una dirección de email
tel	Introduce un número de teléfono. No tiene una sintaxis estricta, pero no admite saltos de línea.
url	Introduce una URL
search	Introduce un número de teléfono. No admite saltos de línea.
number	Introduce un número de punto flotante
date	Introduce una fecha en formato MM-DD-AAAA
datetime	Introduce una fecha, hora, minutos, segundos y milisegundos en el formato UTC de la zona horaria del cliente
datetime-local	Similar a datetime, solo que no admite zona horaria
moth	Introduce un mes y año sin zona horaria
week	Introduce un número de semana sin zona horaria
time	Introduce la hora sin zona horaria definida
color	Especifica un color
range	Slider con un rango de números concretos para elegir

Ejemplo

```
<input type=”email”
          name=”clientEmail”
          autocomplete=”email”
          required />

<input type=”tel”
          name=”clientTel”
          autocomplete=”telephone”
          pattern="[0-9]{9}"
          required />
```

Pero además de los nuevos tipos de campos `<input>`, encontramos la etiqueta `<datalist>`. Se trata de un elemento que contiene una lista de datos a los que cualquier `<input>` del documento puede acceder y utilizar.

Para acceder a estas listas de datos, debe utilizarse el atributo `list` dentro del campo donde queremos acceder. Puede utilizarse para hacer una lista desplegable o simplemente para que vayan apareciendo los posibles resultados conforme escribimos en el campo de tipo texto.

Ejemplo

```
<input type="text" name="navegador" list="navegadores" />

<datalist id="navegadores">
     <option value="Internet Explorer" />
     <option value="Mozilla Firefox" />
     <option value="Google Chrome" />
</datalist>
```

Además, la etiqueta `<option>` puede cerrarse con otra etiqueta, encerrando dentro la descripción larga del elemento en cuestión. De modo que, al acceder a la lista, se mostrará el valor (izquierda) junto a la descripción (derecha).

Ejemplo

```
<input type="text" name="navegador" list="navegadores" />

<datalist id="navegadores">
     <option value="IE">Internet Explorer</option>
     <option value="Firefox">Mozilla Firefox</option>
     <option value="Chrome">Google Chrome</option>
</datalist>
```

4.3 Validar un formulario usando el navegador

A través de HTML5 podemos validar formularios sencillos usando cuatro elementos básicos que hemos visto en puntos anteriores, usando un procedimiento similar al que usaríamos para validar un formulario en HTML 4.0 usando JavaScript.

Este método se basa en la Constraint Validation API[2] de HTML5, la cual permite al navegador saber qué campos no están debidamente cumplimentados, de forma que puede notificar al usuario de forma amigable, que debe rellenar o cumplimentar correctamente un campo.

Cuando añadimos un tipo a un campo, automáticamente le estamos añadiendo una restricción, que será validada cuando lo enviemos.

```
<input type=”email” />
```

Patrón

Si no le añadimos ningún patrón, el navegador interpretará cualquier cadena que introduzca el usuario como un dato del tipo del campo, ya sea un teléfono, un email o una URL. Añadiendo patrones, limitamos el formato que debe tener el texto que introduzcamos.

```
<input type=”email”
 pattern=”@misitio\.com$” />
```

De esta forma, el navegador devolverá un error si el texto introducido no cumple con la expresión regular que le indicamos como patrón, ergo estamos restringiendo el campo de tipo email a una cadena que cumpla con “nombre@servidor.com”.

2 Constraint Validation API http://www.html5rocks.com/en/tutorials/forms/constraintvalidation/

Requerido

Muchos de los campos que tendremos en nuestro formulario, serán campos requeridos, es decir, de cumplimentado obligatorio por el usuario. Si no le hemos indicado el atributo correspondiente, el campo carecerá de dicha restricción.

```
<input type="text"
 required />
```

Simplemente añadiendo el atributo `required`, indicamos al navegador que si este campo no está relleno, no puede enviar el formulario.

Tamaño máximo

Aunque el tamaño máximo de un campo es algo que podemos determinar en el propio patrón del campo, también podemos delimitar el número máximo de caracteres que entran en un campo, usando `maxlength`. Esto añade otra restricción al campo, aunque el navegador no avisa cuando se sobrepasa el límite, simplemente deja de introducir caracteres.

```
<input type="tel"
 maxlength="9" />
```

Min y Max

También podemos especificar el valor máximo y el valor mínimo permitido en un campo, de modo que, si el usuario introduce un valor fuera del rango, el navegador le notifica.

```
<input type="text"
 min="0"
 max="255" />
```

En el ejemplo, solo permitimos valores entre 0 y 255, ambos incluídos.

La validación de campos más complejos, como por ejemplo, confirmación de contraseña, ya si que requiere el uso de JavaScript, aunque podríamos seguir usando la Constraint Validation API para limitar el número de caracteres que se introducen, así como el patrón que definamos en la política de contraseñas de nuestro site.

5. HTML5 APIs

El estándar HTML5 implementa una serie de nuevas APIs (Application Programming Interface) proporcionan una enorme cantidad de recursos para el desarrollo de aplicaciones web más sencillas y de mayor usabilidad.

Ofrecen posibilidades como manejar dibujar en un lienzo imágenes bitmap usando JavaScript (recomendado para gráficos 2D/3D), arrastrar y soltar elementos dentro de una aplicación web usando el drag and drop, o la posibilidad de acceder a ficheros locales del cliente a través del navegador web (API File).

A lo largo de este capítulo, veremos como funcionan y como manejar las API de Canvas, Drag and Drop, Geolocation, Storage, File, History y Offline.

5.1 Canvas

5.1.1 Introducción

El elemento <canvas> nos proporciona un lienzo donde, mediante JavaScript, podemos dibujar gráficos 2D y 3D basados en mapas de bits (bitmap), es decir, píxel a píxel. En capítulos anteriores, pudimos ver el elemento SVG, el cual proporciona una funcionalidad similar pero basada en gráficos vectoriales y utilizando el lenguaje de definición de gráficos homónimo.

El lienzo de Canvas, el desarrollador puede insertar imágenes 2D, dibujarlas dentro, e incluso, utilizando la librería WebGL, gráficos 3D.

Canvas es el elemento idóneo para aquellos gráficos que necesitan actualizarse constantemente, como por ejemplo, los gráficos de un videojuego basado en HTML5.

Ejemplo sencillo

```
<!DOCTYPE html>
<html>
     <body>
          <canvas id="myCanvas"
               width="200"
               height="100"
               style="border:1px solid #000000;">
          Tu navegador no soporta Canvas
          </canvas>
     </body>
</html>
```

5.1.2 Uso

En este apartado, veremos el uso básico del API Canvas creando un cuadrado mediante JavaScript.

Ejemplo de Canvas con JavaScript

```
<!DOCTYPE html>
<html>
     <head>
     <script type="text/javascript">
          function Cuadrado () {
          var lienzo = document.getElementById('miCanvas');
               if (lienzo.getContext) {
                    var context = lienzo.getContext('2d');
                    context.fillStyle = "rgb(150,29,28)";
                    context.fillRect (2,2,96,96);
               } else {
                    Tu navegador no soporta Canvas
               }
          }
     </script>
     </head>
     <body>
          <canvas id="miCanvas" width="100" height="100"></canvas>
     </body>

</html>
```

En el código JavaScript de encima, encontramos la función Cuadrado(). Esta función dibujará un cuadrado en el canvas que le indicamos.

Primero, define una variable llamada “lienzo”, que corresponde al elemento HTML con ID “miCanvas”. Este elemento tiene adherido un contexto. Llamamos contexto a la relación dimensional del lienzo, ya sea 2D (altura y anchura) o 3D (altura, anchura y profundidad).

Antes de empezar a dibujar, debemos asegurarnos de que el elemento canvas existe y lo soporta el navegador. De ser así, al obtener el contexto mediante el método

`getContext`, devolvería un valor true.

A continuación, asignamos una variable llamando a un contexto del lienzo, ya que este, por defecto tiene tanto el 2D como el 3D, y debemos indicarle con cual vamos a generar nuestros gráficos.

Con nuestro contexto designado, llamamos a los métodos `fillStyle` y `fillRect` para definir un estilo concreto y crear una figura rectangular del tamaño que le indicamos como parámetro al método.

El apartado `else` de nuestro código, designa que debe mostrar el navegador en caso de no soportar la etiqueta Canvas. También podemos designar esto dentro de las etiquetas `<canvas>...</canvas>`.

Igual que hemos creado un cuadrado, pueden crearse más formas básicas, así como otras más complejas.

5.1.3 Métodos y propiedades

El API de Canvas es extensa. Los métodos y propiedades disponibles para manejar su contenido se diferencian en grupos: color, estilo y sombras; líneas, rectángulos, rutas, transformaciones, texto, dibujo de imágenes, manipulación a nivel de píxel, composición y otros.

Cada grupo recoge unas funciones concretas.

Color, estilo y sombras

Recoge los métodos y propiedades que nos permiten definir el estilo de una figura, el color y las sombras.

Propiedad	Descripción
fillStyle	Define o devuelve el color, patrón o degradado usado para rellenar la figura.
strokeStyle	Define o devuelve el color, patrón o degradado usado para rellenar un borde.
shadowColor	Define o devuelve el color de una sombra.
shadowBlur	Define o devuelve el nivel de difuminado de una sombra.
shadowOffsetX	Define o devuelve la posición en el eje X de una sombra.
shadowOffsetY	Define o devuelve la posición en el eje Y de una sombra.

Método	Parámetros	Descripción
createLinearGradient()	Coordenadas XY de inicio y XY de final del degradado.	Crea un degradado lineal para usar en el contenido del canvas.
createPattern()	Imagen, repetición	Crea un patrón para usar en el contenido del canvas
createRadialGradient()	XY y Radio de inicio, XY y Radio de final.	Crea un degradado radial para usar en el contenido del canvas
addColorStop()	Posición, color	Especifica una parada de color a un degradado

Lineas

El grupo de líneas está ocupado por las propiedades usadas para definir o devolver el estilo, tipo o tamaño de una línea dentro del contenido de un canvas.

Propiedad	Descripción
lineCap	Define o devuelve el estilo de las tapas finales de las líneas
lineJoin	Define o devuelve el estilo de la esquina que se crea cuando dos líneas se encuentran.
lineWidth	Define o devuelve el ancho de una línea.
miterLimit	Define o devuelve el tamaño máximo del miter

Rectángulos

Aquí encontramos los métodos específicos relacionados con la creación de cuadrados y rectángulos.

Método	Parámetros	Descripción
rect()	Coordenadas XY, ancho y alto.	Crea un rectángulo
fillRect()	Coordenadas XY, ancho y alto.	Dibuja un rectángulo relleno de color
strokeRect()	Coordenadas XY, ancho y alto.	Dibuja un rectángulo sin rellenar
clearRect()	Coordenadas XY, ancho y alto.	Vacía el rectangular especificado dentro de un rectángulo

Rutas

Contiene los métodos usados para trabajar con rutas, es decir, dibujos no predefinidos.

Método	Parámetros	Descripción
fill()	Sin parámetros	Rellena el dibujo actual (ruta)
stroke()	Sin parámetros	Dibuja la ruta que has definido.

beginPath()	Sin parámetros	Inicia una ruta o resetea la actual
moveTo()	Coordenadas XY	Mueve la ruta al punto especificado del canvas sin crear una línea.
closePath()	Sin parámetros	Cierra una ruta.
lineTo()	Coordenadas XY	Añade un nuevo punto y crea una línea a ese punto desde el último punto especificado en el lienzo
clip()	Sin parámetros	Recorta una sección de cualquier tamaño y forma del canvas original.
quadraticCurveTo()	Coordenadas XY del punto de control (cpx, cpy) y coordenadas XY del final	Crea una curva Bézier cuadrática
bezierCurveTo()	Coordenadas XY del punto de control (cpx1, cpy1, cpx2, cpy2) y coordenadas XY del final	Crea una curva Bézier cúbica
arc()	Coordenadas XY, Radio, angulo de inicio, angulo de final (en radianes)	Crea un arco/curva. Se usa para crear círculos.
arcTo()	Coordenadas XY1, XY2 y radio.	Crea un arco/curva entre dos tangentes.
isPointInPath()	Coordenadas XY	Devuelve si el punto especificado es parte de la ruta actual o no.

Transformaciones

Podemos redimensionar, rotar o transformar una imagen bitmat que se encuentre dentro de un canvas usando los métodos de este grupo.

Método	Parámetros	Descripción
scale()	% ancho, % alto	Cambia el tamaño del dibujo, más pequeño o más grande
rotate()	angulo	Gira el dibujo actual
translate()	Coordenadas XY	Desplaza el dibujo actual
transform()	Matriz de 6 valores	Reemplaza la transformada actual. Incluye escala, posición y sesgo.
setTransform()	Matriz de 6 valores	Reinicia la transformada actual.

Texto

El elemento Canvas también puede renderizar texto y aplicarle sombras, estilo, color, etc.

Propiedad	Descripción
font	Define o devuelve la fuente actual del texto
textAlign	Define o devuelve la alineación del texto
textBaseline	Define o devuelve la línea de base del texto

Método	Parámetros	Descripción
fillText()	Coordenadas XY, texto	Crea un texto en las coordenadas X Y.
strokeText()	Coordenadas XY, texto	Crea un texto en las coordenadas X Y, sin relleno
measureText()	Texto	Devuelve un objeto que contiene el ancho del texto

Dibujo de imágenes

Podemos replicar una imagen insertada en el HTML dentro de un canvas usando el método drawImage().

Método	Parámetros	Descripción
DrawImage()	Img, coordenada X, coordenada Y	Inserta la imagen especificada en las coordenadas indicadas del canvas

Manipulación a nivel de píxel

Permite manipular los píxeles de cada elemento dentro del canvas, es decir, leerlos, escribirlos, etc.

Propiedad	Descripción
width	Devuelve el ancho de un objeto ImageData
height	Devuelve el alto de un objeto ImageData
data	Devuelve un objeto que contiene datos de imagen de un objeto ImageData especificado

Método	Parámetros	Descripción
createImageData()	Ancho, alto	Crea un objeto ImageData vacío con el ancho y alto especificado
getImageData()	Coordenadas XY, ancho y alto	Devuelve un objeto ImageData con los mismos datos de pixel del rectangulo especificado.
PutImageData()	Coordenadas XY, imgData	Coloca los datos de imagen (del objeto ImageData especificado) en las coordenadas señaladas

Composiciñón

Las propiedades de composición permiten manipular la transparencia o como una nueva imagen se dibuja sobre otra.

Propiedad	Descripción
globalAlpha	Define o devuelve el valor de la transparencia actual (canal Alpha)
globalCompositeOperation	Define o devuelve como una nueva imagen será dibujada sobre la imagen existente.

Otros

Existen más métodos del API de Canvas, pero al no ser métodos y propiedades generales, sino específicas de cada navegador. Por ejemplo, Mozilla y Microsoft tienen su propia API específica que extiende la estándar, con funciones solo disponibles bajo Firefox e Internet Explorer/Microsoft Edge, respectivamente.

5.2 Drag and Drop

5.2.1 Introducción

El API Drag and Drop proporciona funciones de *arrastrar y soltar* en el navegador del cliente. Mediante esta característica, los usuarios pueden seleccionar y arrastrar un elemento *arrastrable* y colocarlo en otro elemento receptor.

Puede utilizarse, por ejemplo, para cargar archivos a la web simplemente arrastrando el icono desde nuestro sistema al interior de una web en el navegador.

Los elementos *arrastrables* y receptores no están definidos por defecto, sino que podemos personalizar qué elementos podremos mover y qué elementos no.

Esta API se basa en el objeto `DataTransfer`

Ejemplo de Drag and Drop

```
<!DOCTYPE html>
<html>
<head>
     <script>
          function allowDrop(ev) {
               ev.preventDefault();
          }
          function drag(ev) {
               ev.dataTransfer.setData("text", ev.target.id);
          }
          function drop(ev) {
               ev.preventDefault();
               var data = ev.dataTransfer.getData("text");

     ev.target.appendChild(document.getElementById(data));
          }
     </script>
</head>
<body>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

5.2.2 Uso

Usando el ejemplo anterior, vamos a explicar cómo funciona el API Drag and Drop

Ejemplo de Drag and Drop

```
<!DOCTYPE html>
<html>
<head>
     <script>
          function allowDrop(ev) {
               ev.preventDefault();
          }
          function drag(ev) {
               ev.dataTransfer.setData("text", ev.target.id);
          }
          function drop(ev) {
               ev.preventDefault();
               var data = ev.dataTransfer.getData("text");

     ev.target.appendChild(document.getElementById(data));
          }
     </script>
</head>
     <body>

          <div id="contenedor"
               ondrop="drop(event)"
               ondragover="allowDrop(event)">
          </div>

          <img id="arrastrar"
               src="img_logo.gif"
               draggable="true"
               ondragstart="drag(event)"
               width="336" height="69" />

     </body>
</html>
```

Partimos de dos elementos en el HTML: un div contenedor, y una imagen. El contenedor, tiene a su vez asociados dos eventos del API que estamos tratando: `ondrop` y `ondragover`.

Dichos eventos se lanzan cuando un elemento se suelta dentro de un contenedor válido y cuando un elemento se arrastra sobre el contenedor válido. Son los eventos

que manejan la situación.

En la imagen encontramos el atributo `draggable` y el evento `ondragstart`. El atributo en cuestión, le indica al JavaScript que la imagen es arrastrable, mientras que el evento indica qué debe hacerse cuando el elemento, en este caso una imagen, empiece a arrastrarse.

Función `drag`

Se lanza con el evento `ondragover` y especifica qué información va a ser arrastrada.

Función `allowDrop`

Dado que por defecto, un elemento no puede soltarse sobre otro, llamando a *event.preventDefault()* cancelamos cualquier oposición y habilitamos que se pueda arrastrar.

Función `drop`

Por último, mediante esta función obtenemos los datos que están siendo arrastrados y los escribimos dentro del contenedor div.

5.2.3 Métodos, propiedades y eventos

Los métodos y propiedades de Drag and Drop son los métodos y propiedades del objeto `DataTransfer`, en el que se basa esta nueva API.

Método	Parámetros	Descripción
clearData()	Tipo de dato (opcional)	Elimina todos los datos del tipo especificado. Sin argumentos, elimina todos los datos.
getData()	Tipo de dato	Devuelve los datos del tipo especificado
setData()	Tipo de dato	Define los datos del tipo indicado.
setDragImage()	Img, coordenadas XY	Define la imagen que se mostrará al arrastrar, si se desea una personalizada.

Propiedades	Descripción
dropEffect	Define o devuelve el tipo de operación drag-and-drop
effectAllowed	Devuelve la lista de todos los tipos de operación posibles
files	Devuelve una lista de archivos locales disponibles. Si no hay archivos involucrados, devuelve cero.
items	Devuelve un objeto `DataTransferItemList` con todos los datos arrastrados.
type	Devuelve un array con los formatos definidos en los eventos `ondragstart`

Sin embargo, el API Drag and Drop también proporciona una serie de eventos que manejan la situación cuando queremos arrastrar algo.

Evento	Manejador	Descripción
drag	ondrag	Se lanza al ser arrastrado un elemento o texto.
dragend	ondragend	Se lanza al terminar el arrastre (por ejemplo al ser cancelado).
dragenter	ondragenter	Se lanza cuando el elemento arrastrado entra en un objetivo válido.
dragexit	ondragexit	Se lanza cuando el elemento deja de ser el objetivo a arrastrar
dragleave	ondragleave	Se lanza cuando el elemento abandona un contenedor válido.
dragover	ondragover	Se lanza cuando el elemento pasa por encima del contenedor objetivo válido.
dragstart	ondragstart	Se lanza cuando el usuario empieza a arrastrar un elemento.
drop	ondrop	Se lanza al soltar el elemento.

5.3 Geolocation

5.3.1 Introducción

El API de geolocalización, permite mediante un objeto, obtener las coordenadas del dispositivo en este momento. Si está disponible, accede mediante el GPS, si no, utiliza la geolocalización basada en IP a través de la red. Esta última, no es demasiado precisa. El objeto usado para obtener las coordenadas pertenece a la clase `Navigator`.

El API de geolocalización puede combinarse con Google Maps, OpenStreetMaps u otros sistemas de mapeo, para obtener la localización actual del dispositivo en un mapa, lo cual presenta un amplio abanico de posibilidades. Además, existen librerías basadas en este API que proporcionan capacidad de calcular una ruta, por ejemplo, o de guardar lugares de interés.

Ejemplo de Geolocation

```
<!DOCTYPE html>
<body onload=”getLocation()”>

<p id="coordenadas"></p>

     <script>
     var x = document.getElementById("coordenadas");
     function getLocation() {
         if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(showPosition);
         } else {
             x.innerHTML = "Tu navegador no soporta esta
función.";
         }
     }

     function showPosition(position) {
         x.innerHTML = "Latitud: " + position.coords.latitude +
         "<br>Longitud: " + position.coords.longitude;
     }
     </script>
</body>
</html>
```

5.3.2 Uso

El Geolocation API tiene como base el objeto `Position.coords`, del que obtiene las coordenadas (por defecto) y otros datos de interés (si disponibles).

En el ejemplo anterior, tenemos dos funciones JavaScript, `getLocation()` y `showPosition()`. Desde la primera, llamamos a un método del API (`getCurrentPosition()`) el cual nos devuelve un objeto `Position`, precisamente el que necesitamos para esta tarea.

Dicho objeto lo pasa como parámetro a nuestra segunda función, la cual simplemente escribe dentro de un elemento contenedor la longitud y latitud (coordenadas geográficas) de la posición relativa del dispositivo, accediendo a ellas mediante dos propiedades del objeto `Position`.

Sin embargo, así queda un poco minimalista e ineficiente. Necesitamos añadirle una función para manejar posibles errores. Los errores, los devuelve el objeto `error` y podemos manejarlos mediante un switch. Dicha función la añadiremos como parámetro a `getLocation()`

Código

```
function mostrarError(error) {
    switch(error.code) {
        case error.PERMISSION_DENIED:
     x.innerHTML = "Permiso denegado"
     break;
        case error.POSITION_UNAVAILABLE:
     x.innerHTML = "Información no disponible."
     break;
        case error.TIMEOUT:
     x.innerHTML = "La petición ha caducado."
     break;
        case error.UNKNOWN_ERROR:
      x.innerHTML = "Error desconocido."
     break;
    }
}
```

Si queremos, además podemos añadir una imagen estática de Google Maps con el mapa de la localización actual del dispositivo, mediante el API de Google Maps.

Código

```
function showPosition(position) {
        var latlon = position.coords.latitude + "," +     position.coords.longitude;

        var img_url ="http://maps.googleapis.com/maps/api/staticmap?center=
    "+latlon+"&zoom=14&size=400x300&sensor=false";

        document.getElementById("coordenadas").innerHTML = "<img
        src='"+img_url+"'>";
}
```

Hemos modificado la función `showPosition()` para que ahora obtenga la latitud y la longitud en una cadena separando ambos valores por una coma, introduciéndolos en una URL del API de Google Maps, e insertando dicha URL (que da lugar a una imagen) en el contenedor que añadimos antes, donde se mostraban las coordenadas.

Sin embargo, el API de Geolocation, aunque parezca sencilla, posee un enorme potencial gracias a las librerías que extienden estas funciones.

5.3.3 Métodos y propiedades

El objeto `Navigator.geolocation` de por si, no incluye propiedades, sin embargo, usa las del objeto Position, al que hace alusión.

Propiedades del objeto Position

Propiedades	Descripción
coords.latitude	Latitud de las coordenadas actuales
coords.longitude	Longitud de las coordenadas actuales
coords.accuracy	Precisión de la posición
coords.altitude	Altitud actual
coords.altitudeAccuracy	Precisión de la altidud
coords.heading	Desvío en grados en sentido horario desde el Norte
coords.speed	Velocidad en metros por segundo
timestamp	Fecha/hora de la respuesta

Métodos de Geolocation

Métodos	Parámetros	Descripción
getCurrentPosition()	Función callback	Devuelve un objeto con datos sobre las coordenadas
watchPosition()	Función callback	Llama a la función callback cada vez que el dispositivo cambia de posición y devuelve un ID
clearWatch()	id	Elimina el ID introducido de watchPosition()

5.4 Local Storage

5.4.1 Introducción

El API Local Storage, permite a las aplicaciones web almacenar datos localmente en nuestro dispositivo, a través del navegador web.

Antes de HTML5, para almacenar estos datos debíamos utilizar las cookies, incluidas en cada respuesta del servidor. Sin embargo, esta nueva capacidad permite almacenar datos de forma más segura, de mayor tamaño (hasta 5 MB) sin afectar al rendimiento de la web, y lo más importante: esta información nunca será transmitida al servidor sin autorización previa del usuario.

Este almacenamiento es según dominio, de modo que todas las páginas pertenecientes a un mismo dominio, pueden leer y escribir si tienen permiso para ello, sin necesidad de que cada una pida permiso por separado.

El API Local Storage, proporciona dos objetos de la clase `Window`: `window.localStorage`, y `window.sessionStorage`.

Código

```
<!DOCTYPE html>
<html>
     <body>
          <div id="result"></div>
          <script>
               if (typeof(Storage) !== "undefined") {
                    // Almacena
                    localStorage.setItem("nombre", "Juanjo");
                    // Recupera
                    document.getElementById("result").innerHTML =
                         localStorage.getItem("nombre");
               } else {
                    document.getElementById("result").innerHTML =
                         "Tu navegador no soporta esta
característica.";
               }
          </script>

     </body>
</html>
```

5.4.2 Uso

localStorage

Para almacenar y recuperar datos locales, haremos uso de dos métodos: `setItem()` y `getItem()`.

Código

```
// Almacena
     localStorage.setItem("nombre", "Juanjo");
// Recupera
     var dato = localStorage.getItem("nombre");
     alert(dato);
```

Primero almacenamos localmente una variable llamada “nombre”, con valor “Juanjo”. Después, la leemos y guardamos el resultado en una variable, para por último, mostrarla mediante un `alert()`. Al ser un dato almacenado localmente, podemos acceder a el, desde CUALQUIER página del dominio. Por ejemplo, si almacenamos esto desde nuestro index, en page.php?id=1, podemos acceder también.

sessionStorage

El objeto sessionStorage funciona igual que localStorage, sin embargo, este solo guarda datos durante una sesión, es decir, que al cerrar sesión, se eliminan los datos almacenados.

Código

```
if (sessionStorage.clickcount) {
     sessionStorage.clickcount = Number(sessionStorage.clickcount)
+ 1;
} else {
     sessionStorage.clickcount = 1;
}
     document.getElementById("result").innerHTML = "Has hecho
click " + sessionStorage.clickcount + " veces esta sesión.";
```

Los datos se guardarán hasta que se cierre la sesión, o hasta que se cierre la ventana.

5.4.3 Métodos y propiedades

Para manear el API de almacenamiento local, se usan una serie de propiedades y métodos definidos por el estándar HTML5.

Propiedades

Propiedades	Descripción
lenght	Devuelve el tamaño del almacenamiento local

Métodos

Métodos	Parámetros	Descripción
key()	número	Devuelve la clave en el index #
setItem()	clave, valor	Guarda una clave con el valor asociado
getItem()	clave	Devuelve el valor de la clave indicada
removeItem()	clave	Elimina el valor de clave indicada
clear()		Elimina TODAS las claves

5.5 File

5.5.1 Introducción

El API File, permite acceder a los archivos locales de la máquina cliente a través del navegador. Dentro de la especificación de este API, encontramos dos niveles:

- API File Reader (nivel 1, lectura)
- API File Writer (nivel 2, escritura)

Mediante cada nivel del API podremos satisfacer unas necesidades concretas. Por ejemplo, en el primer nivel encontramos una interfaz completa para leer archivos desde la máquina cliente, sin necesidad de cargar estos en el servidor.

Con la interfaz que encontramos en el segundo nivel, podemos escribir un objeto binario, concretamente un objeto `Blob` en el sistema de archivos local del cliente.

Pero además, a través de esta interfaz podremos escribir múltiples binarios en lo que llamaríamos un sandbox (caja de arena). Un sandbox es un sistema de archivos completo, privado y aislado del resto de funciones y del servidor, accesible solamente desde el navegador.

Este API nos obliga a ser más cuidadosos con el código, a capturar correctamente los eventos y excepciones, así como manejar los errores.

5.5.2 File Reader

Leyendo archivos

El primer nivel describe los métodos utilizados para leer y cargar archivos desde el navegador, en nuestra aplicación web. De este apartado, nos interesan cuatro métodos concretos.

Métodos	Parámetros	Descripción
readAsText()	archivo, codificación	Procesa un archivo como texto en la codificación que le indiquemos. Por defecto, UTF-8
readAsBinaryString()	archivo	Procesa un archivo como una cadena de bytes
readAsDataURL()	archivo	Genera una cadena de tipo data: url codificada en base64
readAsArrayBuffer()	archivo	Devuelve un objeto de tipo ArrayBuffer

Al leer un fichero, se lanza el evento `load` que puede ser capturado por el método `addEventListener()`, el cual permite lanzar la función que procese el fichero como queramos. Por ejemplo, que muestre su contenido en pantalla.

Para leer ficheros, `File Reader` solo soporta la subida mediante `<input>` y drag and drop.

Propiedades

Los archivos subidos tienen una serie de propiedades a las cuales, el API File nos permite acceder. Esta información nos sirve para determinar si el archivo es válido, por ejemplo. Es una característica importante pensando en la seguridad de nuestra aplicación.

Propiedades	Descripción
name	Devuelve el nombre del fichero subido
size	Devuelve el tamaño del fichero en bytes
type	Devuelve el tipo MIME del fichero

Blobs

Anteriormente hemos mencionado los blobs, pero sin especificar qué son concretamente. Se trata de objetos que representan datos en crudo. Se crearon para solventar la limitación de JavaScript para trabajar con ficheros. Un blob se crea a partir de un fichero, pero no necesariamente tiene que ser siempre así. Puede generarse un blob a partir de datos obtenidos directamente sobre la aplicación y generar un fichero a partir de este.

Su principal propósito, es trabajar con grandes volúmenes de datos. Por ello, incorpora un método que nos permite "trocear" un blob en conjuntos de datos más pequeños.

Métodos	Parámetros	Descripción
slice()	comienzo, longitud, tipo	Genera un nuevo blob a partir de otro blob o archivo.

Eventos

El API proporciona una serie de eventos para manejar las subidas de forma más eficiente. Además del visto anteriormente evento `load`, existen otros tantos eventos similares. Para su manejo, cada evento tiene un manejador nativo, sin embargo, exite un método (mencionado anteriormente, `addEventListener()` que permite escuchar cualquier evento y manejarlo desde ahí, de forma universal.

Evento	Manejador	Descripción
loadstart	onloadstart	Lanzado cuando empieza la carga
progress	onprogress	Se lanza periódicamente mientras el archivo está siendo cargado/leído
abort	onabort	Se lanza al abortar la operación
loadend	onloadend	Se lanza al terminar la carga/lectura
error	onerror	Se lanza si ocurre un error durante la carga/lectura.

5.5.3 File Directories and System

Es el nivel del API File que nos permite escribir archivos directamente sobre el disco duro de la máquina cliente, utilizando lo que llamamos un sandbox. Un sandbox es un espacio reservado en el disco duro, privado y aislado, al que únicamente la aplicación web y a través del navegador, puede acceder. Esto nos permite escribir blobs en ese espacio reservado, por lo tanto, escribir directamente sobre el disco duro.

Disco duro

Para trabajar con la unidad virtual que el File reserva para su uso, primero tenemos que inicializar el sistema de archivos dentro de esta. El método encargado de esta tarea, es **requestFileSystem()**

Método	Parámetros	Descripción
requestFileSystem()	tipo, tamaño, función éxito, función error	Inicia un sistema de archivos.

El tipo puede ser TEMPORARY o PERSISTENT, dependiendo de como nos interese. El tamaño es el espacio total que reservaremos en el disco duro para inicializar nuestra unidad virtual, y las funciones de éxito y error, son aquellas a las que llamará en caso de que todo haya ido bien o algo haya salido mal. Este método devuelve un objeto FileSystem, con dos propiedades fundamentales.

Propiedades	Descripción
name	Devuelve información sobre el FS, como el nombre que le asigna el navegador.
root	Devuelve una referencia al directorio raiz del FS.

La propiedad root, referencia un objeto DirectoryEntry, y es la base para trabajar con ficheros.

Código

```
function iniciar(){
     cajadatos = document.getElementById('cajadatos');
     var boton = document.getElementById('boton');
     boton.addEventListener('click', crear, false);
     requestFileSystem(PERSISTENT, 5*1024*1024, creardd, errores);
}
```

Crear archivos y directorios

Con nuestro FS iniciado, podemos crear archivos recurriendo a la interfaz DirectoryEntry, incluida en el API. Esta interfaz nos proporciona un total de 4 métodos para crear y manejar archivos y directorios.

Métodos	Parámetros	Descripción
getFile()	ruta, bandera, función éxito, función error	Crea o abre un archivo dentro del FS. Las banderas pueden ser create (crea el archivo si no existe) y exclusive (devuelve un error si intentamos sobreescribir un archivo).
getDirectory()	ruta, bandera, función éxito, función error	Funciona exactamente igual que getFile(), pero con directorios.
createReader()		Devuelve un objeto DirectoryReader para leer entradas en un directorio específico.
removeRecursively()		Elimina un directorio y todo su contenido.

Código

```
function crear() {
     var nombre = document.getElementById('entrada').value;
     if (nombre != '') {
          dd.getFile(nombre, {create: true, exclusive: false},
          mostrar, funcError);
     }
}
```

Listar archivos

Para listar archivos, primero tenemos que crear un objeto DirectoryReader (con el método que vimos anteriormente, createReader()) el cual contiene el método necesario para leer una entrada.

Método	Parámetros	Descripción
readEntries()		Lee todas las entradas de un bloque. Es posible que haya que llamarlo varias vece.

Código

```
function leerdir(dir) {
     var lector = dir.createReader();
     var leer = function() {
          lector.readEntries (
          function(archivos {
               if (archivos.length) {
                    listar(archivos);
                    leer();
               }
          }, funcError);
     }
     leer();
}
```

Código completo

```
function iniciar(){
     cajadatos = document.getElementById('cajadatos');
     var boton = document.getElementById('boton');
     boton.addEventListener('click', crear, false);
     requestFileSystem(PERSISTENT, 5*1024*1024, creardd,
funcError);
}
function crear() {
     var nombre = document.getElementById('entrada').value;
     if (nombre != '') {
          dd.getFile(nombre, {create: true, exclusive: false},
          mostrar, funcError);
     }
}
function mostrar() {
     document.getElementById('entrada').value='';
     cajadatos.innerHTML='';
     dd.getDirectory(ruta, null, leerdir, funcError);
}
function leerdir(dir) {
     var lector = dir.createReader();
     var leer = function() {
          lector.readEntries (
          function(archivos {
               if (archivos.length) {
                    listar(archivos);
                    leer();
               }
          }, funcError);
     }
     leer();
}
function listar(archivos) {
 for(var i = 0; i < archivos.length; i++) {
   if(archivos[i].isFile {
     cajadatos.innerHTML += archivos[i].name + '<br>';
   } else if(archivos[i].isDirectory) {
     cajadatos.innerHTML += '<span
onclick="cambiardir(\''+archivos[i].name+'\')"
     class="directorio"> + '+archivos[i].name+' </span><br>';
   }
 }
```

```
}

function creardd(sistema) {
     dd = sistema.root;
     ruta='';
     mostrar();
}

function funcError(e) {
     alert('Error: '+e.code);
}

window.addEventListener('load', iniciar, false);
```

Este sería un ejemplo completo de como podemos crear un sistema de ficheros virtual con File, y acceder a el creando un fichero dentro, además de listar el contenido.

Eliminar, copiar y mover

Aunque en el ejemplo anterior hemos visto como iniciar un sistema de archivos y como leerlo, aún no podemos eliminar entradas, copiarlas o moverlas a otras ubicaciones del FS. Para ello, haremos uso del interfaz provista por Entry.

En todos estos métodos, primero debemos obtener la entrada con la que vamos a trabajar. Para ello, usaremos el método **getFile()** visto anteriormente.

Métodos	Parámetros	Descripción
remove()		Elimina la entrada que le pasemos como objetivo.
moveTo()	directorio, nombre, función éxito, función error	Mueve la entrada objetivo al destino que le indiquemos. Si le pasamos un nombre, además renombra la entrada.
copyTo()	directorio, nombre, función éxito, función error	Copia la entrada objetivo en el destino indicado. Podemos además pasarle un nombre para la copia.

Ejemplo Copiar

```
function copiar(){
     var origen=document.getElementById('origen').value;
     var destino=document.getElementById('destino').value;

     dd.getFile(origen,null,function(archivo) {
          dd.getDirectory(destino,null,function(dir) {
               archivo.copyTo(dir,null,exito,errores);
          }, funcError);
     }, funcError);
}
```

Ejemplo Mover

```
function mover(){
     var origen=document.getElementById('origen').value;
     var destino=document.getElementById('destino').value;

     dd.getFile(origen,null,function(archivo) {
          dd.getDirectory(destino,null,function(dir) {
               archivo.moveTo(dir,null,exito,errores);
          }, funcError);
     }, funcError);
}
```

Ejemplo Eliminar

```
function modificar() {
     var origen = document.getElementById('origen').value;
     var origen = ruta+origen;
     dd.getFile(origen, null, function(entrada) {
          entrada.remove(exito,errores);
     }, funcError);
 }
```