actibyti a netmind social project

API MULTIMEDIA JAVASCRIPT

Autor: Ricardo Ahumada

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"

ÍNDICE DE CONTENIDOS

1 CASO PRÁCTICO

Caso Práctico: BananaTube multimedia

"BananaTube" es el proyecto estrella de Banana Apps.

BananaTube será el próximo boom! de las redes sociales; permitirá a sus usuarios gestionar videos, exponerlos en su muro, comentar videos propios y de sus amigos, calificarlos y compartirlos en varios canales.

Continuando con el desarrollo de la interfaz de usuario, se quiere que los elementos multimedia de la aplicación, tengan controles con un look and feel propios de la compañía y que la funcionalidad asociada a ellas, respondan a un desarrollo propio.

Discutamos

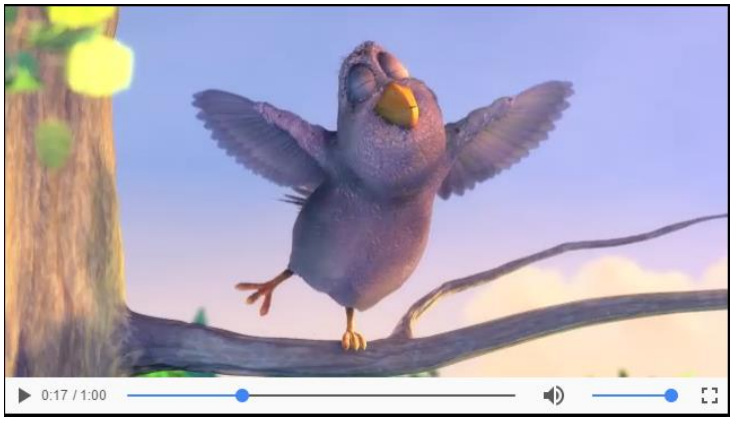
- Seguimos trabajando con Javascript?
- Necesitamos algo más?
- Y nuestro entorno?
- Qué herramientas necesitaremos?

10 Multimedia

Utilización de las etiquetas <video> y <audio>

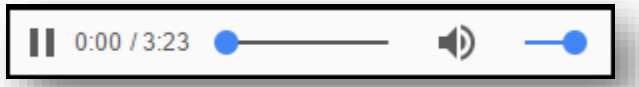
- Se complementan con los elementos <source> y <track>
  - El origen del elemento multimedia puede ir en más de un formato

```
<video src="big_buck_bunny.mp4" controls id="mi_video"></video>
```



- La etiqueta <audio> funciona prácticamente igual que <video>

```
<audio src="un_audio.mp3" controls id="mi_audio"></audio>
```



Acceso a los objetos de video y audio

- Se acceden como cualquier otro elemento DOM

```
var video = document.getElementById('mi_video');

var audio = document.getElementById('mi_audio');
```

- Una vez seleccionados los elementos se puede acceder a sus atributos y métodos:

```
video.duration;
video. play();

audio.src;
audio.load();
```

Atributos Video y Audio

Attribute	Description
autoplay	Sets or returns whether the audio/video should start playing as soon as it is loaded
buffered	Returns a TimeRanges object representing the buffered parts of the audio/video
controls	Sets or returns whether the audio/video should display controls (like play/pause etc.)
currentSrc	Returns the URL of the current audio/video
currentTime	Sets or returns the current playback position in the audio/video (in seconds)
defaultMuted	Sets or returns whether the audio/video should be muted by default
duration	Returns the length of the current audio/video (in seconds)
ended	Returns whether the playback of the audio/video has ended or not
error	Returns a MediaError object representing the error state of the audio/video
loop	Sets or returns whether the audio/video should start over again when finished

Atributos Video y Audio

Attribute	Description
muted	Sets or returns whether the audio/video is muted or not
networkState	Returns the current network state of the audio/video
paused	Returns whether the audio/video is paused or not
played	Returns a TimeRanges object representing the played parts of the audio/video
preload	Sets or returns whether the audio/video should be loaded when the page loads
readyState	Returns the current ready state of the audio/video
src	Sets or returns the current source of the audio/video element
videoTracks	Returns a VideoTrackList object representing the available video tracks
volume	Sets or returns the volume of the audio/video

Métodos Audio/Video

Method	Description
addTextTrack()	Adds a new text track to the audio/video
canPlayType()	Checks if the browser can play the specified audio/video type
load()	Re-loads the audio/video element
play()	Starts playing the audio/video
pause()	Pauses the currently playing audio/video

Eventos Audio/Video

Event	Description
abort	Fires when the loading of an audio/video is aborted
canplay	Fires when the browser can start playing the audio/video
durationchange	Fires when the duration of the audio/video is changed
ended	Fires when the current playlist is ended
error	Fires when an error occurred during the loading of an audio/video
loadeddata	Fires when the browser has loaded the current frame of the audio/video
loadedmetadata	Fires when the browser has loaded meta data for the audio/video
pause	Fires when the audio/video has been paused
play	Fires when the audio/video has been started or is no longer paused
playing	Fires when the audio/video is playing after having been paused or stopped for buffering
progress	Fires when the browser is downloading the audio/video
waiting	Fires when the video stops because it needs to buffer the next frame

Añadiendo controles personalizados

- Crea botones personalizados para iniciar, pausar y parar el video de presentación de un usuario
- Haz lo mismo con el audio.

11 Canvas

<canvas>

- Se usa para el procesamiento dinámico de mapa de bits gráficos o juegos
- Para algunos, resulta el competidor nativo natural de Flash
  - Y de Silverlight, cuando éste se usa como elemento aislado dentro de una página
- Se trata de un sistema de trazado que genera mapas de bits (bitmaps)
- Esto tiene implicaciones
  - Calidad
  - Escalado
  - Deformación.

<canvas>

- No forma parte del DOM (Modelo de objetos del documento) o de ningún otro espacio de nombres
- Tampoco es capaz de detectar interacciones con el usuario
  - Pulsaciones en una zona de su superficie, por ejemplo
- El rendimiento es bueno
  - No tiene que almacenar definiciones de primitivas de dibujo
  - No tienen que descargarse los gráficos del servidor
  - Debe de evitarse cuando pueda hacerse lo mismo con HTML5+CSS3

Dibujando sobre el <canvas>

- La superficie de trabajo viene delimitada por los dos únicos atributos posibles del elemento (aparte de los genéricos): anchura (width) y altura (height).

```
<canvas id="Canvas1" width="300" height="200"/>
```

- Si reasignamos cualquiera de ellos después de haber realizado cualquier operación
  - Borrará todo su contenido y reiniciará todas las propiedades del contexto de dibujo
  - Incluso si se reasigna al mismo valor que tienen en ese momento
- De forma predeterminada no tiene ninguna representación visual

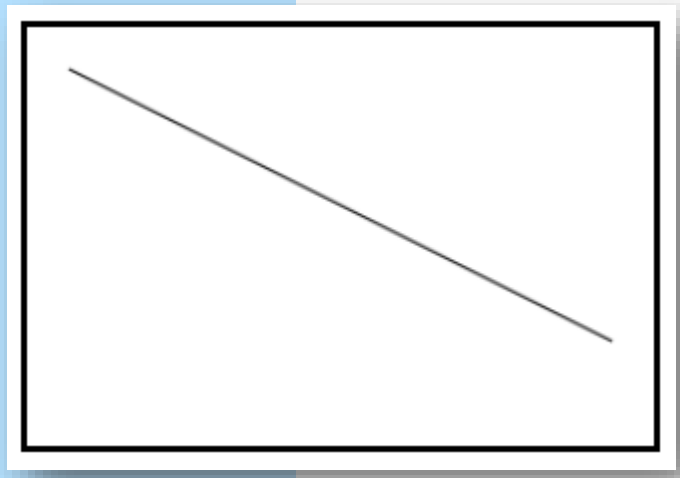
- Referencia útil: http://www.rgraph.net/reference/index.html

Dibujando sobre el <canvas>

```
<canvas id="Canvas1" width="300" height="200" style="border:3px
solid;" />

  <script>
      function basicDrawing() {
            // Gets the element and its context
            var canvas = document.getElementById('Canvas1');
            var context = canvas.getContext('2d');
            // To draw a line we start by moving to the start
            // point, and then draw using lineTo() method
            context.beginPath();
            context.moveTo(20, 20);
            context.lineTo(280, 150);
            // Draw the line
            context.stroke();
            context.closePath();
      }

      basicDrawing();
  </script>
```



Rellenos y otras composiciones

- Podemos dibujar un triángulo mediante la adición de este par de líneas al código anterior

```
context.lineTo(20, 150);
context.lineTo(20, 20);
```

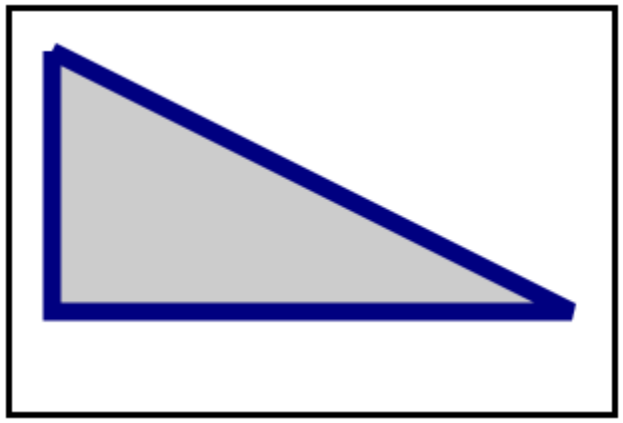
- Si queremos rellenar la superficie definida, solo tendremos que llamar al método **fill**, complementado con fillStyle y strokeStyle
  - Colores de relleno y borde respectivamente
  - Si queremos resaltar más el borde podemos utilizar también lineWidth para cambiar el grosor

Dibujando sobre el <canvas>

- Con estas 3 líneas de código nuevas para nuestro rectángulo,

```
context.fillStyle = "#ccc";
context.strokeStyle = "Navy";
context.lineWidth = 9;
```

- …obtendríamos una salida como esta:



Circunferencias

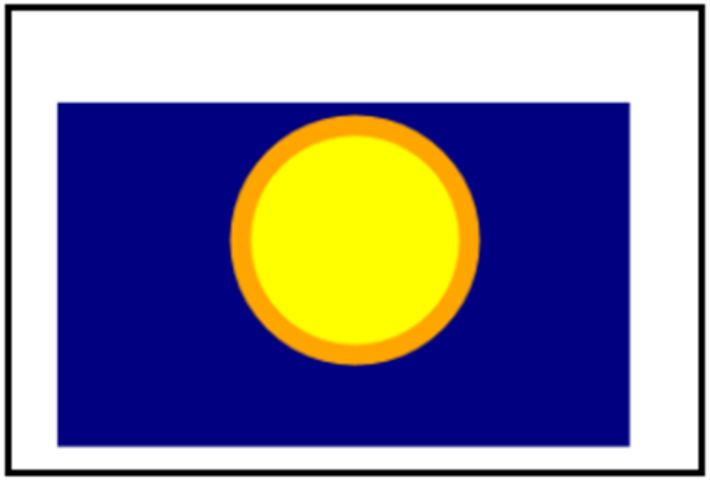
- Para otros tipos de figuras, como circunferencias, círculos, etc., se utiliza context.arc, que tiene esta declaración formal:
  - context.arc(x, y, radius, startAngle, endAngle, counterClockwise)
- Donde
  - x,y son las coordenadas del centro
  - radius resulta evidente…
  - startAngle y endAngle son los ángulos inicial y final, pero expresados en radianes
  - counterClockwise es un valor booleano que indica si el relleno del círculo (del arco en realidad), debe realizarse en el sentido de las agujas de reloj o en el contrario.

Dibujando sobre el <canvas>

- Si queremos trazar un círculo relleno con un borde dentro de nuestro rectángulo:

```
context.fillStyle = "Yellow";
context.arc(150, 100, 50, 0, Math.PI * 2, false);
context.fill();

context.arc(150, 100, 50, 0, Math.PI * 2, false);
context.strokeStyle = "Orange";
context.lineWidth = 9;
context.stroke();
```

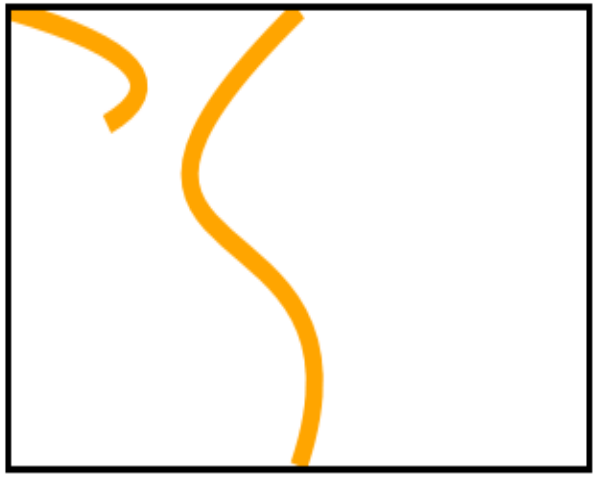


Dibujando sobre el <canvas>

- El API también suministra un mecanismo de trazado de curvas suaves (Bézier)
- Disponible en sus variedades cuadrática y curva
  - quadraticCurveTo(cp1x, cp1y, x, y)
  - bezierCurveTo (cp1x, cp1y, cp2x, cp2y, x, y)

- Vamos a crear un nuevo dibujo que implemente estos dos tipos de curvas en el mismo canvas:

```
context.strokeStyle = "Orange";
context.lineWidth = 9;
context.moveTo(0, 0);
context.quadraticCurveTo(100, 25, 50, 50);
context.moveTo(150, 0);
context.bezierCurveTo(0, 125, 200, 75, 150, 200);
context.stroke();
```

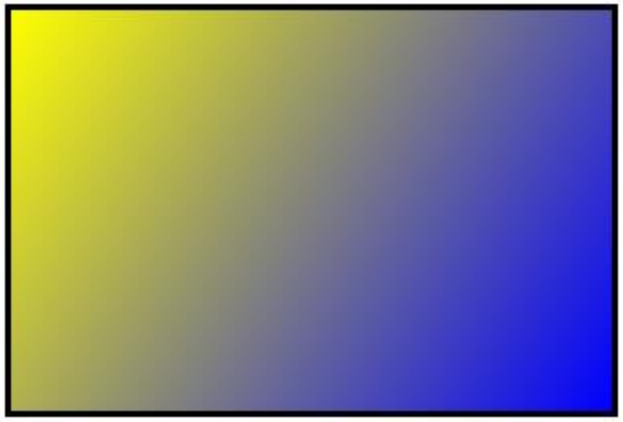


Dibujando sobre el <canvas>

Gradientes de color

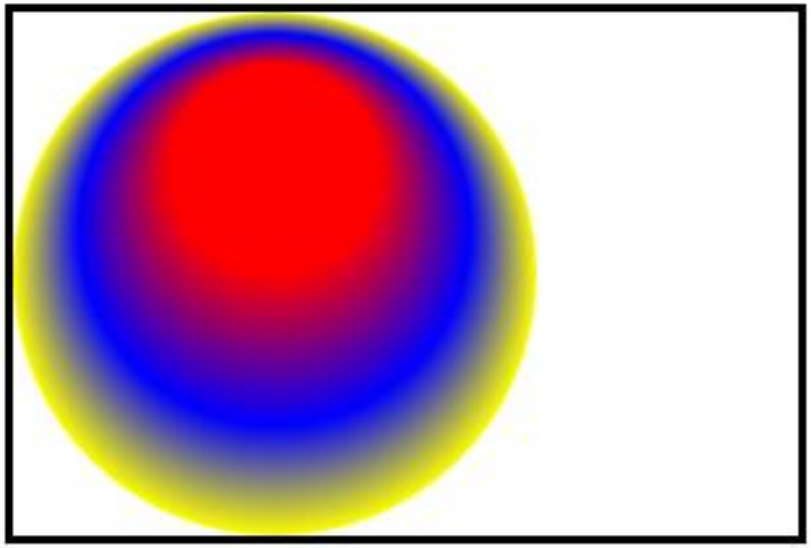
- Gamas de color que van progresando desde un valor inicial a otro final y pueden tener una serie de puntos intermedios de cambio
- Podemos distinguir dos tipos: gradientes radiales (createLinearGradient) y lineales (createRadialGradient):

```
/* Definición del gradiente */
var gradient = context.createLinearGradient(0, 0, 300, 200);
/* Puntos de inicio y fin */
gradient.addColorStop(0, "Yellow");
gradient.addColorStop(1, "Blue");
/* Asignación y proceso de relleno */
context.fillStyle = gradient;
context.fillRect(0, 0, 300, 200);
```



Dibujando sobre el <canvas>

```
var gradient = context.createRadialGradient(100,50,25,100,100,100);
// Define gradient points
gradient.addColorStop(0.2, 'Red');
gradient.addColorStop(.66, 'Blue');
gradient.addColorStop(1, 'Yellow');
// Now define the fillStyle with the previous gradient
context.fillStyle = gradient;
context.arc(100, 100, 100, (Math.PI/180)*0, (Math.PI/180)*360, false);
context.fill();
```



Pongámoslo en práctica

- Crea un cuadro tipo Kandinsky sencillo usando un canvas

- https://www.ibiblio.org/wm/paint/auth/kandinsky/kandinsky.comp-8.jpg

Trabajando con texto

- Lo normal es que combinemos estas características con texto e imágenes, esto es, incorporando elementos ya construidos
  - Letras o contenido en formato .jpg o .png
  - Definimos un tipo de letra, un texto a dibujar, y una posición en el canvas, y ahí se comienza la escritura

- Context nos suministra los elementos necesarios :
  - font, para indicar el tipo de letra
  - textAlign, para la alineación
  - textBaseline, para establecer en qué punto empezamos a dibujar respecto al punto de origen
  - fillText, para generar la salida del texto en pantalla

- La declaración formal es la siguiente:
  - fillText([text],[x],[y],[maxWidth]);

Trabajando con texto

- Si queremos dibujar el normativo “Hello World” en nuestro canvas, con un tipo de letra Segoe UI de 18 píxels

```
context.font = 'bold 28px Segoe UI';
context.fillText('Hello World', 50, 50);
```

- Podemos conseguir efectos curiosos mediante la función strokeText, que solamente dibuja el borde de la letra
  - Y para el relleno la función **fillText**

Trabajando con texto

- Si queremos dibujar la palabra “Hello”, con un borde azul y un relleno en rojo, siendo el tamaño de letra de 68 píxeles:

```
context.font = 'bold 68px Segoe UI';
context.lineWidth = 5;
context.strokeStyle = "Navy";
context.strokeText('Hello', 50, 70);
context.fillStyle = 'Red';
context.fillText('Hello', 50, 70);
```

Trabajando con texto

- Para crear un sombreado, operamos como siempre.
- Definimos un color para la sombra (shadowColor)
- Un desplazamiento u offset (shadowOffset, la cantidad de sombra)
- Y un nivel de nitidez (shadowBlur), estos dos últimos medidos en píxeles.

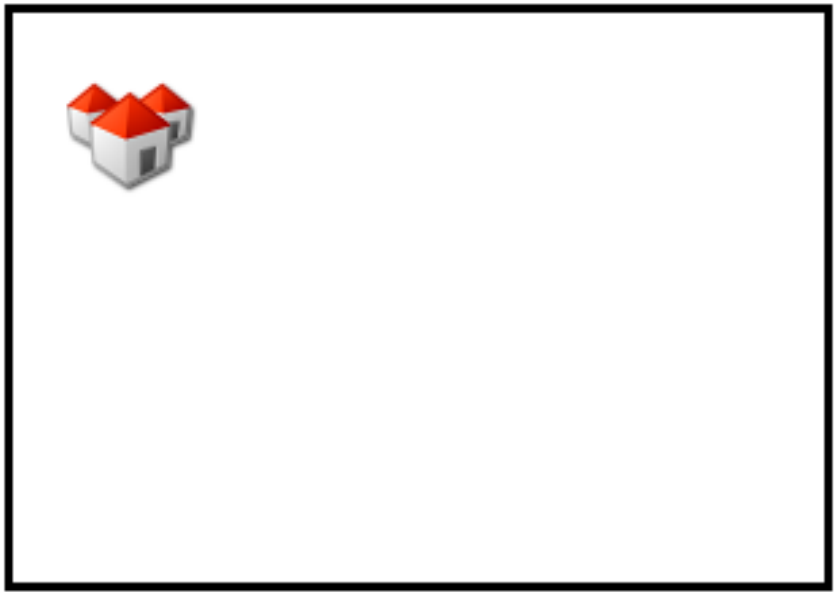
```
// Shadow definition
context.shadowColor = "rgba(100,100,100,.4)";
context.shadowOffsetX = context.shadowOffsetY = 12;
context.shadowBlur = 5;
```

Utilizando imágenes en un <canvas>

- El procedimiento es, hasta cierto punto, similar al del trabajo con texto
- El primer paso es obtener una referencia al gráfico
  - Puede ser un elemento <img>
  - Puede crearse dinámicamente mediante una llamada al método "constructor" del objeto Image de este API
  - Hecho esto, tendremos que asignarle una imagen real, en su propiedad src
    - Que tendrá que apuntar a una URL válida (local o remota)
- Finalmente, llamar al método DrawImage del objeto context, indicándole cuál es la imagen y el resto de parámetros a utilizar
  - Posición donde se comienza a dibujar la imagen
  - Tamaño deseado (anchura y altura).

Utilizando imágenes en un <canvas>

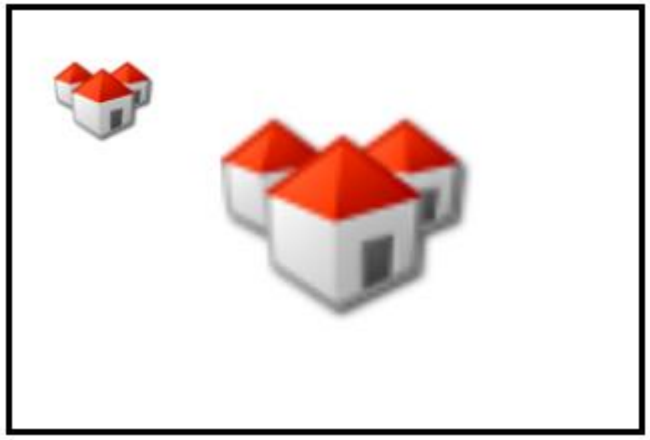
```
var myImage = new Image();
// Esto se programaría realmente en el evento
load de la // imagen
myImage.src = 'images/houses.png';
context.drawImage(myImage, 20, 20, 48, 48);
```



Utilizando imágenes en un <canvas>

- Otra posibilidad interesante es la de utilizar una imagen existente para realizar copias o variantes de la misma
- Indicamos una zona rectangular que nos servirá de punto de partida, y otra que será la de destino
- En ambos casos, debemos indicar punto de origen, anchura y altura, por lo que podremos tener copias a diferentes resoluciones.

```
context.drawImage(myImage, 0, 0, 48, 48, 100, 40, 120, 120);
```



Creando un juego HTML

- Accede al tutorial de Aditya Ravi Shankar que muestra como generar el típico juego Breakout:
  - http://www.adityaravishankar.com/2011/10/html5-game-development-tutorial-breakout-i-introduction-to-canvas/

- Examina el código e intenta reproducirlo

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es

GOBIERNO DE ESPAÑA
MINISTERIO DE ENERGÍA, TURISMO Y AGENDA DIGITAL
red.es
ESTRATEGIA DE EMPRENDIMIENTO Y EMPLEO JOVEN garantía juvenil
Agenda Digital para España
UNIÓN EUROPEA
Fondo Social Europeo
"El FSE invierte en tu futuro"