Reconocimiento y síntesis de voz en el navegador

Manipulación de la webcam, micrófono, introducción de subtítulos, reconocimiento de voz, comandos hablados y síntesis de voz en el browser.

Juan Antonio Jiménez Torres

Fecha: 29/02/2016

Índice

1. Introducción

En el cajón de las cosas que me quedan por aprender lleva mucho tiempo dando vueltas el tema del manejo de la webcam y micrófono desde el navegador, jugar con el *text to speech*, y *voice to text*. La única forma de aprender es hacer pruebas y trastear en mi propio laboratorio virtual. Este post nace fruto de esa experimentación, y por eso surge este artículo, para compartir con el lector los resultados obtenidos.

Antes de HTML5 sólo se podía utilizar audio y vídeo en el navegador mediante algún tipo de plugin. Y flash era lo más utilizado para este fin. Pero eso tenía más contras que pros: el navegador requería tener instalado el software de *Adobe*, que no venía de serie en el browser y que no siempre ha demostrado ser seguro a lo largo de sus distintas versiones. Al final ese plugin ha caído en desgracia y cada vez su penetración en el sector es menor.

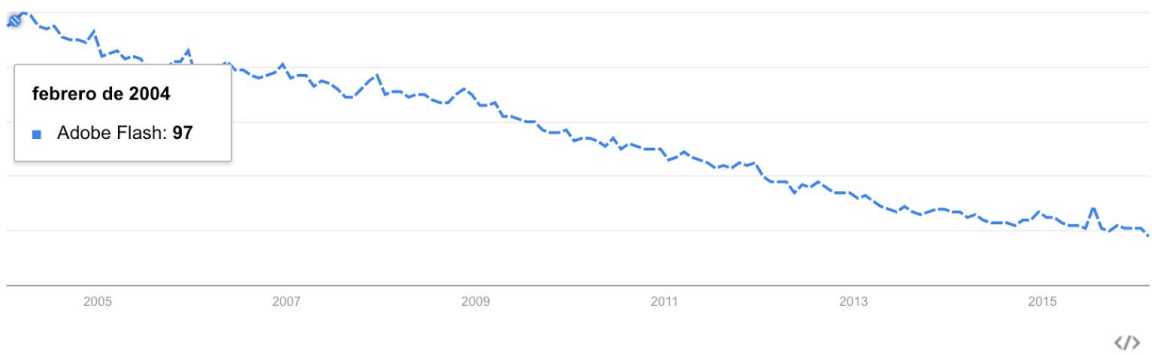

Imagen de Google Trends sobre las búsquedas de Adobe Flash

Pero a la vez que Flash iba perdiendo protagonismo, han aparecido una serie de estándares que han venido al rescate, facilitando de forma natural la manipulación de audio y vídeo en los navegadores, a través de HTML5 y las nuevas WebAPIs.

Y en eso versa este artículo que se divide en dos grandes bloques:

- el primero, trata de ver cómo mostrar audio y vídeo en la web, introduciendo subtítulos en distintos idiomas, y como capturar el stream de la webcam y el micrófono.
- el segundo trata de reconocimiento de voz, dictado, navegación mediante comandos de voz, y la operación inversa, que sería dado un texto, la correspondiente síntesis de voz.

2. Etiqueta <audio> y <video> de HTML

HTML5 introduce dos nuevas etiquetas `<audio>` y `<video>`. Me centraré en el primero, pues es el medio para conseguir el fin, siendo el objeto principal de estudio de este artículo el reconocimiento de voz y viceversa. Como en tantas cosas, los distintos navegadores no se ponían de acuerdo en qué formatos de audio soportar. Hoy la mayoría soportan el formato ogg o mp3, siendo lo normal que soporten ambos, por

lo que es conveniente proveerlos, para que el browser utilice el primero que le resulte compatible.

```
<audio controls>
  <source src="ejemplo.mp3" type="audio/mpeg" />
  <source src="ejemplo.ogg" type="audio/ogg" />
  <p>Your user agent does not support the HTML5 Audio element.</p>
</audio>
```

Ver ejemplo 1: Sample Audio HTML Tag

Si tienes más interés, puedes consultar la lista de los formatos soportados por los distintos navegadores.

Algunos de los atributos que soporta la etiqueta `<audio>` son:

- controls, que nos permite una representación gráfica de un reproductor, con sus correspondientes controles de pausa, play, volumen, y una línea de tiempo.
- autoplay, si se desea que empiece a reproducirse cuando se haya cargado el DOM.
- preload que puede tomar alguno de los siguientes valores ["none", "metadata", "auto"], para precargarlo. Con metadata sólo se cargarían los metadatos (info, duración del fichero, etc...)
- loop: para que se repita una y otra vez.
- mediagroup: para agrupar y hacer listas de reproducción.

Y hay numerosos eventos y propiedades vinculados a esta etiqueta que nos permiten jugar, siendo los siguientes los más comunes.

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

function play(){
  audio.play();
}

function pause(){
  if (audio.paused){
    audio.play();
  }else{
    audio.pause();
  }
}

function stop(){
  audio.pause();
  audio.currentTime = 0;
}

function subirVolumen(){
  if (audio.volume<0.9){
    audio.volume+=0.1;
  }
```

```
}

function bajarVolumen(){
  if (audio.volume>0.1){
    audio.volume-=0.1;
  }
}
```

Ver ejemplo 2: Sample Audio Events

3. ¿Y cómo ponemos subtítulos?

Para esto tenemos la etiqueta `<track>`, que se puede usar en combinación, tanto con la etiqueta `<audio>` como con la etiqueta `<video>`.

```
<audio controls>
  <source src="ejemplo.mp3" type="audio/mpeg" />
  <track kind="subtitles" label="Subtítulos en español"
src="subtitles_es.vtt" srclang="es" default />
  <track kind="subtitles" label="English subtitles" src="subtitles_en.vtt"
srclang="en" />
  <track kind="subtitles" label="Deutsche Untertitel" src="subtitles_de.vtt"
srclang="de" />
  <p>Your user agent does not support the HTML5 Audio element.</p>
</audio>
```

Esta etiqueta se puede usar para diversas finalidades que se pueden definir en el atributo kind:

- captions
- chapters
- descriptions
- metadata
- subtitles

Esto nos puede dar mucho juego para poner subtítulos en nuestros vídeos o audios. Lo bueno, es que se puede facilitar traducciones a distintos idiomas para un mismo audio (o vídeo).

Estos ficheros utilizan un estándar denominado WebVTT que tiene un formato de texto plano muy sencillo: comienza con la palabra WEBVTT, y sigue con una lista de cues. Un CUE es el mínimo grupo de información, que está formado por un identificador (opcional), por un rango de tiempo y el texto a mostrar.

Por ejemplo:

```
WEBVTT

00:00:00.000 --> 00:00:05.170
Bienvenidos a este curso de seguridad web

00:00:06.010 --> 00:00:08.870
En Autentia nos encanta compartir lo que…
```

```
00:00:08.920 --> 00:00:11.580
…aprendemos con la comunidad

00:00:12.680 --> 00:00:14.730
¡¡¡Venga. Vamos al lío!!!
```

El formato de tiempos es ciertamente melindroso. En https://quuz.org/webvtt/ validan si el WebVTT que estamos generando es válido, si los tiempos son secuenciales y no se solapa nada. Muy útil.

Jugando con la posibilidad de poner subtítulos, y preparando un ejemplo para esto, me he tropezado con algún que otro inconveniente:

- crossdomain: video/audio, vtt y html deben compartir dominio, y si no, usar CORS para remediar este problema.
- si se usa la etiqueta `<audio>`, Chrome no muestra los subtítulos, ya que no tiene "espacio" donde mostrarlos en el componente del reproductor de audio. Tendríamos que leer los CUEs mediante JavaScript, que se puede, y ponerlos en un fuera del reproductor de audio. Sin embargo, si cambiamos la etiqueta por `<video>`, nos reproduce el audio y los subtítulos.
- Si los subtítulos aparecen desde el segundo 0 se muestran en la imagen de previsualización del vídeo (antes de darle al "play"). Así que es mejor que aparezcan unos milisegundos después.

En el siguiente ejemplo, he subtitulado el famoso discurso de Martin Luther King Jr., *"I have got a dream"*, para consolidar lo aprendido sobre los ficheros WebVTT.

Ver ejemplo 3: Subtitulando "I have got a dream", el famoso discurso de Martin Luther King

4. Controlando la webcam y el micrófono

Ya sabemos reproducir audio y vídeo en nuestras páginas y aplicaciones web, pero ¿y cómo capturamos fotos, un vídeo o nuestra voz? HTML nos da acceso al micrófono y a la webcam mediante un método del objeto `navigator` llamado `getUserMedia()`.

Lo primero, es independizar la sintaxis del motor del navegador, ya que aún hay alguno que no tiene el método getUserMedia e implementa su propia versión con prefijo:

```
navigator.getUserMedia = navigator.getUserMedia ||
                         navigator.mozGetUserMedia ||
                         navigator.webkitGetUserMedia ||
                         navigator.msGetUserMedia;
```

Luego indicamos si queremos capturar vídeo, audio o ambas cosas. En el ejemplo para ilustrar este ejercicio, en un primer momento capturaba sonido e imagen, y posteriormente los reproducía en una página web. Esto producía el lógico e indeseable acoplamiento del sonido, al escucharse por los altavoces lo mismo que se capturaba por el micrófono.

```
navigator.getUserMedia({video: true, audio: false}, onSuccess, onError);
```

Siendo:

```
function onSuccess(streamVideo) {
    var videoProvider;

    if (window.URL) {
        videoProvider = window.URL.createObjectURL(streamVideo);
    } else {
        videoProvider = streamVideo;
    }

    myVideo.src = videoProvider;
    myVideo.play();

    myButton.value = "stop";
}

function onError(error) {
    console.log("getUserMedia() no longer works on insecure origins. To use
this feature, you should consider switching your application to a secure
origin, such as HTTPS. See https://goo.gl/rStTGz for more details.", error);
}
```

Ver ejemplo 4: Jugando con la webcam y el micrófono

Si todo ha ido bien, el navegador nos habrá pedido permiso para poder capturar el vídeo a través de la webcam. Con `window.URL.createObjectURL(streamVideo)` se crea un objeto de tipo Blob o File que consume recursos, que no se liberarán hasta que se cierre la página. Puedes descargarlo cuando ya no se utilice llamando a `revokeObjectURL`.

Nota: *ambos métodos forman parte de la nueva File API del W3C, que veremos en otro próximo artículo*

Con esto hemos capturado el stream de audio y de vídeo, y se lo enchufamos al source de nuestro componente pero, de esta forma, estamos confinados a nuestra aplicación web. Una pregunta lógica sería ¿se pueden hacer más cosas con esto? ¿se puede grabar? ¿se puede enviar al servidor o a otro dispositivo?

En principio, parece que hay una especificación del W3C, aún en borrador, que propone usar el propio elemento `file` con el atributo `accept`.

Por ejemplo, para tomar una foto de la cámara y enviarla, sería:

```
<input type="file" accept="image/*;capture=camera">
```

Mientras que para grabar sonido o vídeo sería:

```
<input type="file" accept="video/*;capture=camcorder">
<input type="file" accept="audio/*;capture=microphone">
```

La especificación que se está encargando de esto es: https://dev.w3.org/2009/dap/camera/ . Os recomiendo echar un vistazo a los

ejemplos, donde hay uno que explica cómo se haría con AJAX.

Ya sabemos capturar el sonido en bruto del micrófono pero ¿y si lo que queremos es grabar dicho sonido en un formato comprimido en tiempo real en el navegador? Hay un proyecto muy interesante sobre este tema que podéis encontrar en:

http://audior.ec/blog/recording-mp3-using-only-html5-and-javascript-recordmp3-js/

Por un lado hacen uso de la librería `recorder.js` para grabar el sonido y, por otro lado, han hecho una versión en JavaScript de la librería Lame MP3 encoder, `libmp3lame.js`, para comprimir a MP3 en tiempo real.

Si lo que queremos es hacer videoconferencia sobre el navegador habría que utilizar RTCPeerConnection y RTCDataChannel, pero eso lo dejamos para un posible POST más adelante sobre la WebRTC API.

En estos últimos puntos hemos pasado rápido, de puntillas, indicando una serie de enlaces donde averiguar más información, pero sin profundizar, y es que estos experimentos tienen enjundia suficiente como para tratarlos como una entidad propia y separada.

Y ahora retomamos nuestro objetivo inicial de poder dictarle al navegador (VTT: **V**oice **T**o **T**ext), darle comandos de voz y la operación inversa: síntesis de voz a partir de un texto (TTS: **T**ext **T**o **S**peech).

5. Dictándole al navegador

La API que se encarga del reconocimiento de voz es la Web Speech API. Pero antes de presentarla, vamos a ver cómo de sencillo resulta indicarle al navegador que reconozca nuestra voz.

```
var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition;
var recognition = new SpeechRecognition();

recognition.lang = "es-ES";
recognition.continuous = false;
recognition.interimResults = true;

recognition.onresult = function(event) {
  for (var i = event.resultIndex; i < event.results.length; i++) {
    if (event.results[i].isFinal){
      document.getElementById("text").innerHTML +=
        event.results[i][0].transcript;
    }
  }
}
```

Ver ejemplo 5: dictándole al navegador

Evidentemente, hay otros eventos que podemos controlar, a parte de `onresult`, pero es en éste donde se produce casi toda la chicha.

```
recognition.onstart = function(event) { console.log(event); }
recognition.onerror = function(event) { console.log(event); }
```

```
recognition.onend = function(event) { console.log(event); }
```

En el ejemplo anterior, se puede observar que el objeto `recognition` define algunas propiedades en su configuración:

- `lang`: idioma en formato *locale*, ya que tiene importancia el acento del locutor, que puede ser distinto dependiendo del país o región. No es lo mismo español de España (es_ES), que español de Argentina (es_AR). La pronunciación, acentos y localismos, pueden variar.
- `continuous`: indica si el reconocimiento se hace de forma continuada o no. Es decir, si se detiene cuando el usuario deje de hablar. Por defecto, su valor es false, de forma que si deseamos que el reconocimiento se haga continuado, debemos inicializarla a true.
- `interimResults`: valor booleano que señala si se desean mostrar los valores provisionales o no sobre el texto reconocido. Hay que tener en cuenta, que el resultado puede cambiar.

Ahora vamos a centrar nuestra atención en el evento onresult, donde recibimos un array de `results` con longitud `resultIndex`, que se corresponden con cada uno de los textos que ha reconocido. A su vez, este array contiene información sobre cada "pieza".

- `isFinal`: indica si ha terminado de decidir qué texto ha reconocido.
- a su vez contiene un array con los posibles textos que ha creído reconocer, cuya transcripción está en la propiedad `transcript`, y cuya fiabilidad se mide en la propiedad `confidence`, que es la probabilidad entre 0 y 1 de que la transcripción sea correcta.

Jugando con estas propiedades, podemos discernir cuando `isFinal` y cuando no, y utilizarlo para introducir puntos y seguidos. Pero para introducir otros signos de puntuación, sólo se me ocurre midiendo las pausas en la dicción, y para eso habría que medir los tiempos que se producen entre eventos, y extraer un patrón en ellos.

En el siguiente ejemplo, he hecho una versión rudimentaria, para transformar lo dictado a frases.

Ver ejemplo 6: Speech Recognition and transform in phrases

6. Comandos y navegación por voz.

Ya hemos experimentado cómo el navegador puede reconocer nuestra voz. Ahora cabría preguntarse, si podemos vincular ciertas palabras a comandos de voz. La intuición nos parece indicar, que si el navegador puede reconocer una palabra, siempre podremos despachar un evento vinculado a dicha palabra que dispare un determinado método. Pero en ese caso, tendríamos que estar atentos a todas las palabras que se le pueda decir al navegador.

Para facilitar eso, lo que se puede hacer, es restringir la lista de posibles palabras que pueda reconocer el browser, definiendo una gramática. Para esto también hay un

estándar: el JSpeech Grammar Format. Tiene una pinta a lo que sigue:

```
var grammar = '#JSGF V1.0; grammar actions; public <action> = siguiente |
anterior | volver | inicio | baja | arriba;'
```

Este ejemplo, podría definir los comandos para navegar por voz en un blog. Esto puede ser útil, por ejemplo, en los navegadores de las SmartTVs, o para personas con problemas motores.

```
var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition;
var SpeechGrammarList = SpeechGrammarList || webkitSpeechGrammarList;

var grammar = '#JSGF V1.0; grammar actions; public <action> = siguiente |
anterior | volver | inicio | baja | arriba;';

var speechRecognitionList = new SpeechGrammarList();
    speechRecognitionList.addFromString(grammar, 1);

var recognition = new SpeechRecognition();
    recognition.grammars = speechRecognitionList;
    recognition.lang = 'es-ES';
    recognition.interimResults = false;
    recognition.maxAlternatives = 1;

recognition.onresult = function(event) {
  eval("myCommand." + event.results[0][0].transcript + "()");
}
recognition.onend = function() {
  recognition.start();
}
recognition.start();

var myCommand = {
  siguiente : function(){
      //go to the next post
      this.writeCommand("siguiente");
  },
  anterior : function(){
      //go to previous post
      this.writeCommand("anterior");
  },
  volver : function(){
      //history.go(-1);
      this.writeCommand("volver");
  },
  inicio : function(){
      //go to the homepage
      this.writeCommand("inicio");
  },
  baja : function(){
      //scroll down
      this.writeCommand("baja");
  },
  arriba : function(){
      //scroll to top
      this.writeCommand("arriba");
  },
```

```
  writeCommand : function(_command){
    var comma = ", ";
    if (document.querySelector("#text").innerHTML.length == 0){
      comma = "";
    }
    document.querySelector("#text").innerHTML += comma + _command;
    console.log(recognition);
  }
}
```

Ver ejemplo 7: Voice command test with JSpeech Grammar Format

En este ejemplo, la gramática es muy simple, pero JSGF permite definir gramáticas mucho más complejas. Su definición sigue reglas semejantes a las de las expresiones regulares:

- separador de conjunciones disyuntivas (OR) es la barra |
- para conjunciones copulativas (AND) es el ampersand &
- para cuando hay tokens opcionales, los corchetes [ ]
- para el cierre de Klein (repeticiones de 0 ... n), el asterisco *
- para el cierre de Klein positivo el + (1 repetición o más)
- unas reglas pueden componerse con otras ya definidas anteriormente.

De esta forma, se puede llegar a tener gramáticas amplias, y bien definidas, que podemos emplear desde para una navegación por voz, a definir un juego de instrucciones o comandos para una determinada aplicación web.

7. Síntesis de voz a partir de un texto

Una vez recorrido el camino hasta aquí, cabría la posibilidad de preguntarse por la función inversa *"¿y leer un texto? ¿es posible que dado un texto lo pueda convertir a una locución?"*. Y la respuesta es afirmativa. Sí se puede. Dentro de la Web Speech Api, está el objeto `SpeechSynthesis`, que a su vez tiene un array de objetos de tipo `SpeechSynthsisVoice` con las distintas voces que disponemos en nuestro navegador.

Una prueba rápida con el siguiente código, muestra que dispongo de más de 80 voces en Google Chrome

```
var synth = window.speechSynthesis;
var voices = synth.getVoices();
```

Aunque no todas en español. Además, al hacer una prueba con Firefox, para ver de cuántas voces disponen otros navegadores, me doy cuenta de que la prueba parece que no funciona. ¿y eso por qué? Leyendo descubro que mozilla considera esta tecnología experimental y viene implementada a partir de la versión 45. En versiones anteriores hay que habilitarla específicamente en el navegador en `about:config` en el `flag media.webspeech.synth.enabled = true`; Hecho ésto, ya funciona también en Firefox.

Después de solventar este pequeño inconveniente, quería ver cuáles son las voces de que dispongo en en español. Para lo que me hice una pequeña prueba:

Ver ejemplo 8: Spanish voices in my browser

Lo primero que veo es que Google Chrome me devuelve 2 voces más que firefox.

Hay 5 voces de habla hispana: - Monica[es-ES] - Diego[es-AR] - Paulina[es-MX] - Google español[es-ES] - Google español de Estados Unidos[es-US]	Hay 3 voces de habla hispana: - Diego[es-AR] - Monica[es-ES] - Paulina[es-MX]

Google Chrome añade un par de voces más: una con español de España y otra con español de ¿Estados Unidos?.

En los objetos de tipo voz `SpeechSynthsisVoice` echo en falta que entre sus propiedades indiquen si la voz es masculina o femenina. Hay que sacarlo por contexto, por el nombre que le han dado a la voz.

Ahora vamos a intentar leer un texto de ejemplo, y para eso está el objeto `SpeechSynthesisUtterance`

```
var sampleText = "Esto es un texto para probar la síntesis de voz";
var utterance = new SpeechSynthesisUtterance(sampleText);
```

A este objeto, le vamos a indicar que debe leer con la voz de "Mónica", que es la que disponemos con locale "es-ES".

```
utterance.voice = voices.filter(
  function(voice) {
    return voice.name == 'Monica';
  })[0];
```

y le decimos que realice la locución:

```
window.speechSynthesis.speak(utterance);
```

Puedes ver este ejemplo funcionando en:

Ver ejemplo 9: Basic speech synthesis

El objeto `SpeechSynthesisUtterance` nos provee de una serie de eventos con los que controlar la locución. Entre las propiedades cabe mencionar dos:

- `rate`: se trata de la velocidad a la que lee. El valor por defecto es 1. Eso significa que 2 es el doble de la velocidad normal, y que 0.5 es la mitad de la velocidad normal. Cualquier valor que se aleje de estas cifras, lo hace anormalmente alto o bajo, siendo la locución demasiado rápida o lenta.
- `pitch`: se trata del tono con el que se lee la enunciación. El valor por defecto es 1 y puede tomar valores comprendidos entre 0 y 2, siendo el cero un tono muy bajo, y dos un tono muy alto.

8. Posibles aplicaciones y conclusiones

A través de los anteriores ejemplos, hemos visto que se pueden incluir audios y vídeos en nuestros sites sin hacer uso de Adobe Flash, que se pueden subtitular en distintos idiomas, incluir distintas pistas de audio, que podemos dictarle al navegador, despachar eventos mediante comandos de voz o realizar locuciones a partir de un texto.

La primera aplicación que nos viene a la mente es probablemente el tema de la accesibilidad, que en España está muy orientada al colectivo de invidentes. Pero hay que pensar también en personas con problemas motores, que no puedan usar un teclado o un ratón.

Pienso también en los videotutoriales que tanto proliferan por la red: qué fácil sería subtitularlos a distintos idiomas, o incluso añadirles diferentes pistas de audio con traducciones, llegando así a un colectivo mucho mayor.

En una navegación clásica, entramos a la página web de un cine, navegamos por el menú para consultar la cartelera, elegimos la pelicula y la sesión, y de un mapa de asientos, seleccionamos las butacas libres en las que deseamos sentarnos. ¿No se podría hacer esto mismo mediante comandos de voz? Así podría comprar unas entradas, hablándole a mi móvil mientras voy conduciendo de vuelta a mi casa. Y quien dice el cine, dice reservar un billete de tren, avión u hotel.

9. Enlaces y referencias

Puedes ver todos los ejemplos de este tutorial en:
https://jsfiddle.net/user/jjimenezt/fiddles/

Formatos de audio soportados por los distintos navegadores
https://en.wikipedia.org/wiki/HTML5_Audio#Supported_audio_coding_formats

Introducción al elemento pista de HTML5
http://www.html5rocks.com/es/tutorials/track/basics/

WebVTT: The Web Video Text Tracks Format
https://w3c.github.io/webvtt/

HTML Media Capture (Draft)
https://dev.w3.org/2009/dap/camera/

Web Speech API Documentation (Final)
https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

JSpeech Grammar Format
https://www.w3.org/TR/jsgf/