UNIVERSITAT JAUME I

Grado en Ingeniería Informática

Trabajo Final de Grado

Plataforma online de reconocimiento de voz para la realización de informes radiológicos en tiempo real

---

Autor:
Khadija HARBAZ

Supervisor:
Rafael FORCADA MARTÍNEZ
Tutor académico:
Cristina CAMPOS SANCHO

Fecha de lectura: 14 de septiembre de 2017

Curso académico 2016/2017

Resumen

El presente documento se corresponde con la memoria del Trabajo Final de Grado realizado en la empresa ActualTec Innovación Tecnológica, S.L. En el mismo, se va a proceder a describir el proceso de re-implementación de una plataforma online de dictado de voz orientada principalmente al ámbito radiológico.

La empresa ActualTec, dispone de un servicio que ofrece a sus clientes para transcribir informes radiológicos en tiempo real. Actualmente, la parte del cliente depende de un componente Flash que realiza la grabación y compresión de audio, que es enviado al servicio online de reconocimiento suministrado por la empresa Vocali mediante la API INVOX Medical Dictation. Esta API incorpora la última tecnología de reconocimiento de voz para conseguir los mejores resultados de transcripción en el dictado de informes médicos.

Este proyecto ha sido creado para mejorar la productividad y la calidad de la plataforma, y especialmente su sostenibilidad. Para ello, se ha propuesto eliminar la dependencia con el componente Adobe Flash y sustituirla por la tecnología estándar HTML5 + Javascript. Ajustando, para ello, la interfaz de modo que sea capaz de capturar el audio del micrófono desde el navegador, comprimirlo, enviarlo al servidor externo Vocali y recibir la transcripción que éste devuelve, haciendo uso de WebSockets para la comunicación con el servidor.

Palabras clave

HTML5, Javascript, WebSockets, MediaStream API, Speex, Opus, Web Workers, JSON, reconocimiento

Keywords

HTML5, Javascript, WebSockets, MediaStream API, Speex, Opus, Web Workers, JSON, recognition

Índice general

Capítulo 1

1. Introducción

1.1. Contexto y motivación del proyecto

ActualTec Innovación Tecnológica S.L fundada por dos ingenieros informáticos titulados por la Universitat Jaume I, inició su actividad en 2007 como una empresa de base tecnológica y en la actualidad, cuenta con dos líneas de negocio fuertemente consolidadas.

Por una parte, ActualWeb (servicios web), que se dedica al desarrollo y mantenimiento web, así como al alojamiento web y gestión de dominios, disponiendo de un servicio de hosting propio con el cual abastece tanto a todos sus clientes como a la misma empresa. Y por otra parte, ActualMed[1] (servicios médicos) que se dedica a ofrecer servicios informáticos a centros radiológicos. Entre dichos servicios se encuentra el sistema Actualpacs que permite alojar, editar y acceder a estudios radiológicos.

Entre sus principales clientes destacan por una parte: médicos radiólogos, clínicas radiológicas y hospitales, y por la otra, empresas de desarrollo y alojamiento web, así como pymes con presencia online y servicios de email.

El proyecto propuesto desde ActualTec Innovación Tecnológica S.L consiste en la migración de la aplicación cliente de reconocimiento de voz que actualmente está implementada usando Adobe Flash a un producto desarrollado en HTML5 + Javascript.

1.2. Alcance

Respecto al alcance del proyecto, la nueva adaptación sólo afectará a la parte cliente de la plataforma encargada del audio.

Funcionalmente, se permitirá a los usuarios, tal y como venía haciendo hasta ahora, generar informes radiológicos a través de dictado de voz, con la salvedad de no tener que descargar

[1] Donde se ha realizado la estancia en prácticas

ningún complemento adicional y, además, de poder realizarlo desde cualquier dispositivo ya sea móvil, tablet u ordenador.

1.3. Objetivos del proyecto

El objetivo principal que se pretende conseguir con este proyecto es desarrollar una plataforma que soporte la misma funcionalidad que la actualmente está en producción, pero eliminando la dependencia de la tecnología Adobe Flash usada en el lado del cliente para el registro y compresión del audio.
Esto conlleva a desarrollar una plataforma de reconocimiento de voz que cumpla con las siguientes funcionalidades:

- Que funcione en los navegadores más populares.
- Que sea compatible con terminales móviles y tabletas.
- Que no requiera de Adobe Flash y/o componentes adicionales.
- Que sea libre de los problemas causados por el Flash en la producción de audio.
- Que sea capaz de capturar el audio del navegador y que sea capaz de guardarlo sin distorsiones ni ruidos.
- Que sea capaz de enviar el audio correctamente comprimido y codificado a la API de Vocali.
- Que sea capaz de mostrar por pantalla en tiempo real, la transcripción correspondiente al audio.

Se considera que la nueva implementación muestra una serie de ventajas frente a la anterior plataforma, y que traerá beneficios a la empresa tanto a corto como largo plazo, ya que esta nueva plataforma, con el uso de HTML5 y Javascript, tendrá una perspectiva de sostenibilidad ante la eventual retirada de Adobe Flash de los navegadores.

1.4. Estructura de la memoria

La documentación del desarrollo del proyecto comienza en el capítulo 2 con una descripción de la plataforma anterior, las tecnologías empleadas en el proyecto y de cómo se utilizan para conseguir la implementación de la plataforma. A continuación, en el capítulo 3, se habla de la metodología empleada y la planificación seguida para la consecución del proyecto, se describen las tareas llevadas a cabo y los recursos utilizados con sus respectivos costes. En el capítulo 4, se

describen los requisitos técnicos definidos para el proyecto y el diagrama de casos de uso. Así como el diseño de la arquitectura del sistema. En el capítulo 5 se han especificado los detalles de la implementación de las funcionalidades más importantes del sistema y un desglose de las pruebas realizadas para verificar el correcto funcionamiento del sistema. Y, por último, en el capítulo 6 se ha realizado una valoración del cumplimiento de los objetivos y se ha incluido una opinión personal acerca del desarrollo de un proyecto de estas características.

Capítulo 2

2. Descripción del proyecto

En este capítulo se describe el estado de la plataforma actual con su estructura tecnológica. Además se incluye una descripción sobre las distintas tecnologías empleadas en el proyecto a desarrollar.

2.1. Estado actual

Como se ha mencionado, la empresa ActualMed dispone de una plataforma para el dictado de voz cuya parte cliente depende del componente Flash. Internamente, ese componente se encarga de capturar el audio a una frecuencia de muestreo de 16 KHz y comprimirlo utilizando para ello el *codec* [1] *Speex* [2], comunicándose con el servidor de transcripción mediante WebSockets para el envío del audio y la recepción de la transcripción. En la Figura 2.1 se puede apreciar la estructura actual de la plataforma.

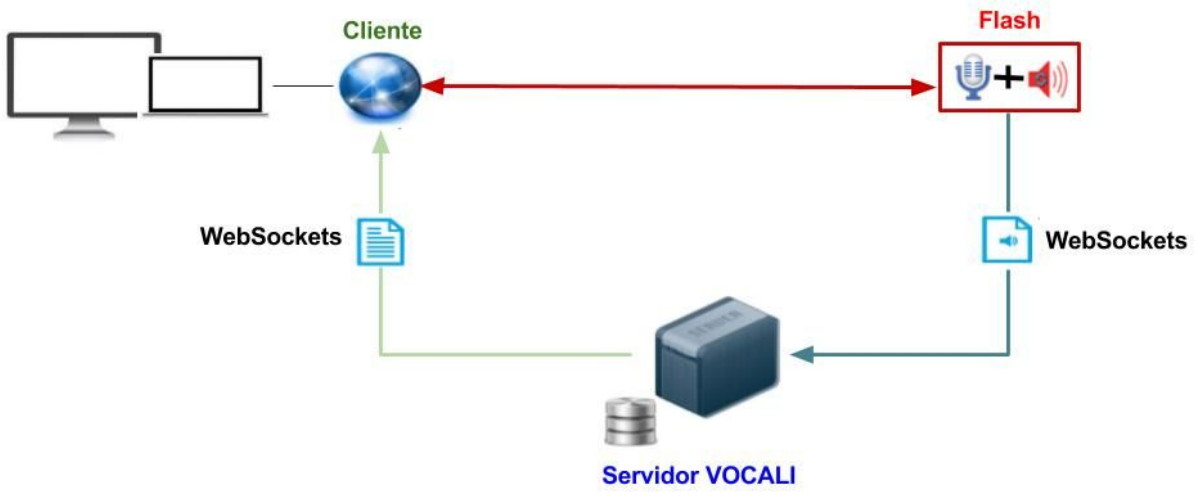


Figura 2.1: La estructura tecnológica actual

En la Figura 2.2 se muestra una captura de la interfaz de la plataforma actual de edición de informes. Como se aprecia en esta figura, tal interfaz dispone de tres desplegables, que se comentan a continuación:

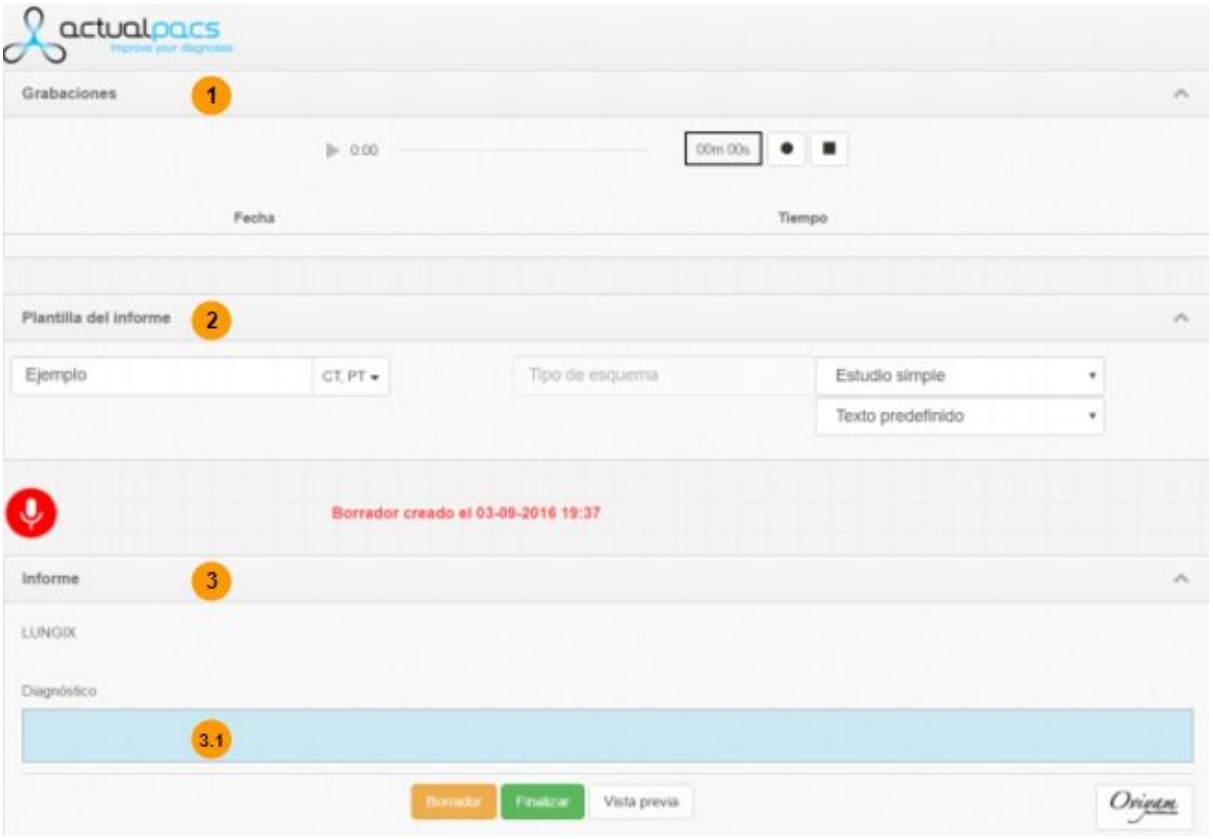


Figura 2.2: Captura de la plataforma de edición de informes mediante dictados de voz

1. **Grabaciones**, en esta sección se realizan las grabaciones de voz y su administración (reproducir, eliminar, pausar, parar y descargar). Además, muestra de forma visual e intuitiva el estado de la grabación; rojo si el dictado está activado (Figura 2.2.1).

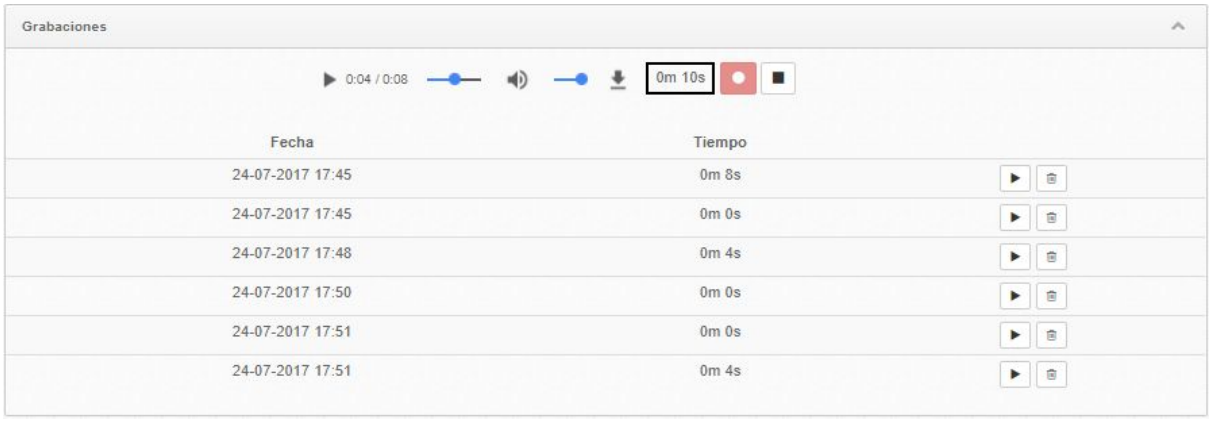


Figura 2.2.1: La sección de grabación en la plataforma de edición de informes

2. **Plantilla del informe,** en esta sección se puede elegir una plantilla usada posteriormente en el informe radiológico. Una plantilla no es más que una configuración predefinida del editor de estudios radiológicos. Dicha configuración engloba aspectos

como la distribución de los cuadrados de texto, la firma del radiólogo y textos predefinidos. (Figura 2.2.2)

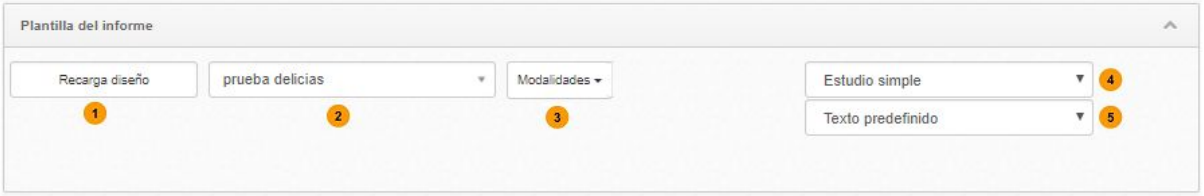
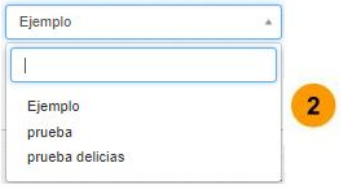
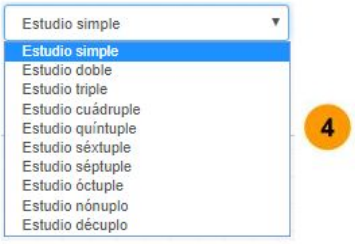


Figura 2.2.2: La sección de plantilla del informe en la plataforma de edición de informes

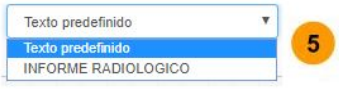
Una vez elegidas las opciones se recarga el diseño para visualizarlo en la siguiente sección del informe.



En este desplegable se pueden elegir entre los diferentes formatos personalizados creados anteriormente.



Aquí se puede elegir un tipo de estudio



Aquí se elige el tipo de informe (predefinido o radiológico)

En este desplegable se pueden elegir las diferentes modalidades del aparato de captación: radiografías convencionales (CR), tomografías (CT), resonancias magnéticas (MR), mamografías (MG), ecografías (US), etc.

3. **Informe**: en esta sección se muestra el informe generado, con el diseño especializado en la sección anterior (Figura 2.2.3) y la transcripción correspondiente al audio dictado en la primera sección (punto 3.1 Diagnóstico).

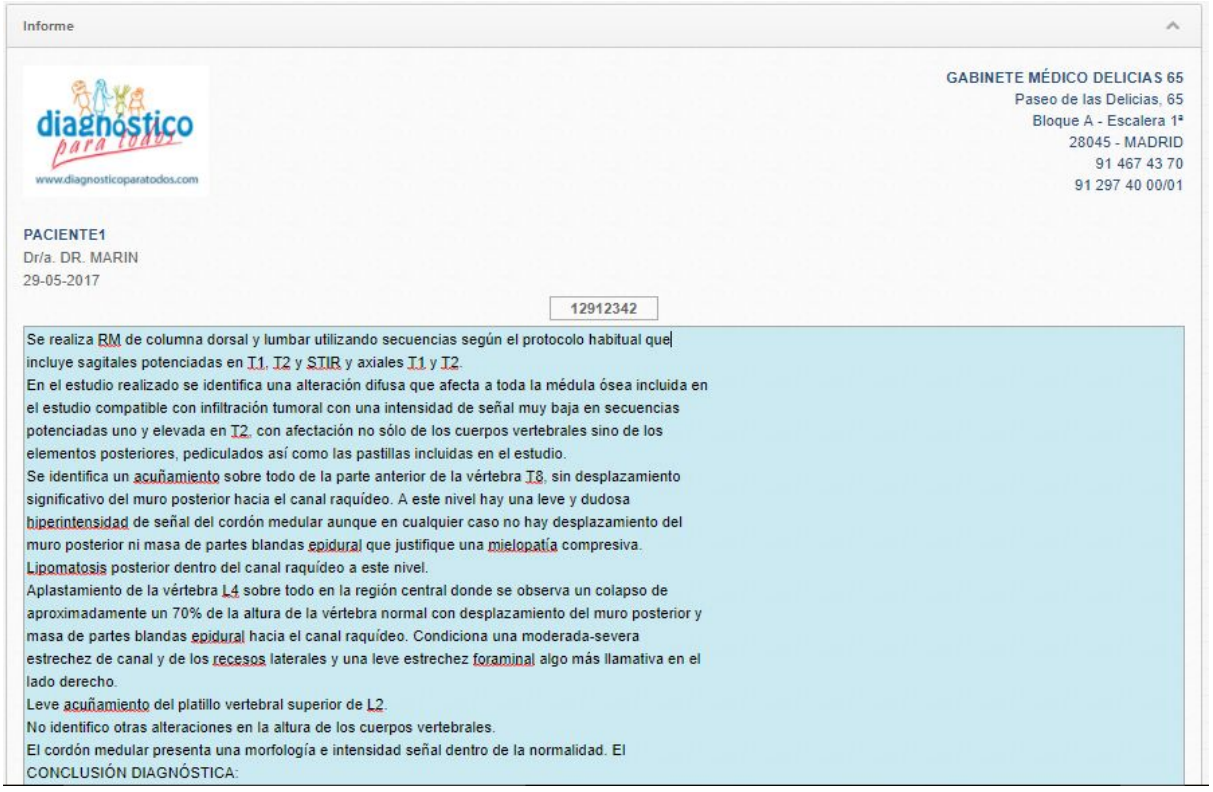


Figura 2.2.3: La sección del informe generado en la plataforma de edición de informes

Se puede guardar un **borrador** del informe y generar otros. **Finalizar** la edición de informes o **visualizar** cómo se ha quedado el informe. (Figura 2.2.4)

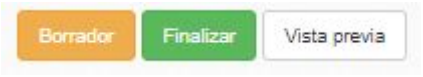


Figura 2.2.4: Las diferentes opción de edición de informes

2.2. Definición del sistema a implementar

Teniendo en cuenta los objetivos, el sistema a implementar seguirá un modelo cliente-servidor. Donde los usuarios del portal actuarán como clientes, comunicándose con el servidor (Vocali) que gestionará los servicios requeridos (Figura 2).

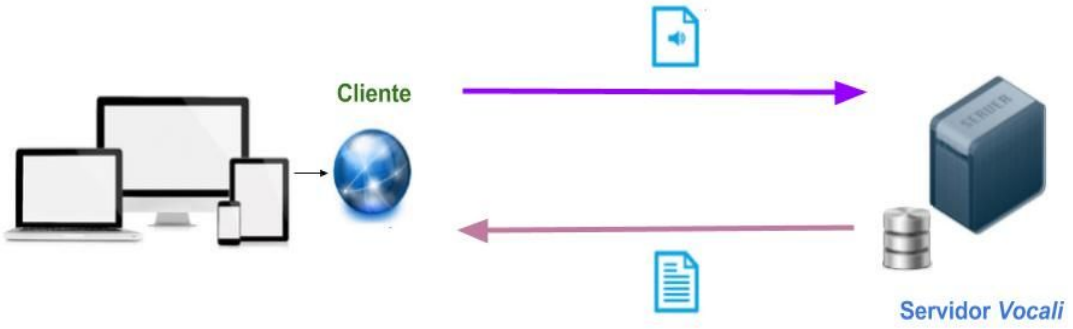


Figura 2: Estructura general del sistema

2.2.1 Cliente

La aplicación cliente deberá recoger el audio del micrófono desde el navegador, realizará un procesamiento del audio que marcan los requerimientos de la plataforma INVOX, y lo enviará al servidor según dichos requerimientos.

2.2.2 Servidor

En este proyecto se realizan peticiones al servidor INVOX de Vocali, dedicado a dar servicio a Actualmed. Emplear los servicios de este servidor es un requerimiento de la empresa y forma parte de la especificación del proyecto. La empresa ha elegido previamente Vocali por las siguientes cualidades:

- Diccionarios especializados para diferentes disciplinas médicas.
- Elevadísima tasa de acierto.
- Precio competitivo.

Entre sus servicios más interesantes destacan:

- Entrenamiento del perfil de voz del usuario.
- Transcripción de voz a texto a partir de modelos de lenguaje de texto libre.
- Reconocimiento de frases definidas en gramáticas de comandos al estilo SRGS.
- Comandos y otras facilidades para la edición de documentos.
- Gestión de diccionarios y macros (sustituciones), incluyendo la posibilidad de compartir.

- Almacenamiento centralizado y actualización automática del perfil de voz de los usuarios.

En este proyecto se hará uso del servicio de "Transcripción de voz a texto", que en Vocali se especifica como "Modo Motor de Dictado".

Dado que no tenemos ningún tipo de control sobre el server de Vocali, no debemos preocuparnos de él, sino interactuar debidamente.

2.3. Justificación de la sustitución del Flash

En 1996 la empresa Macromedia introduce Flash [3] [4]. Flash permite, mediante una plataforma de desarrollo de fácil uso, crear animaciones vectoriales de calidad, de modo que artistas y diseñadores pueden introducir mejores contenidos en las páginas web mediante una etiqueta <object>. Para que funcione Flash en el navegador sólo es necesario una librería que suministra Macromedia y que se instala fácilmente en los sistemas operativos en su momento más habituales y que dispone de plugins para los distintos navegadores, incluyendo el navegador oficial de MS Windows, Internet Explorer [5], que adelanta en popularidad a Netscape en 1996.

Flash se puede programar con un lenguaje muy similar a Javascript (ActionScript) y se convierte el la herramienta más popular para desarrollar juegos online embebidos en páginas web. Hasta 2010 Flash, absorbido por Adobe, no para de crecer, introduciendo antes que nadie la posibilidad de grabar y reproducir audio y vídeo. En 2010, Apple anuncia que no incluirá Flash en los dispositivos móviles, Google dejará también de soportarlo en Android. Actualmente se considera una tecnología obsoleta que pronto dejará de ser soportada incluso por los navegadores de escritorio. Siendo Chrome el más influyente del momento, ya ha anunciado que en breve dejará de funcionar Flash, al igual que Java.

El declive del Flash se debe al crecimiento simultáneo de Javascript y HTML que se consideran estándares del WWW. Javascript fue renombrado a ECMAScript [6], que en su versión 5 es un potente lenguaje de programación empleado incluso en el lado del servidor. En 1997 el W3C describe el DOM, una especificación formal de las páginas web para que puedan ser manipuladas desde Javascript. Esto permite la creación de páginas dinámicas, que se actualizan según eventos provocados por el usuario (por ejemplo con el ratón). Además en 2002 JS es capaz de obtener datos del servidor mediante Ajax (acuñado así en 2005) que no es más que

una librería que permite realizar consultas HTTP desde Javascript y actualizar la página web dinámicamente desde JS. Pese a este incremento de funcionalidad, JS sigue sin poder competir con Flash en elementos visuales.

Pero en 2014 W3C presenta el nuevo HTML5 que incluye elementos de audio y vídeo. JS incluye entonces librerías (API) que permiten reproducir y grabar audio y vídeo incluso manipularlo. Esto supone una alternativa potente a Flash y empieza a sustituirlo en las plataformas multimedia más populares como Youtube o Facebook. Javascript es nativo en los navegadores, no requiere software externo ninguno. Además Flash sólo muestra un buen rendimiento en las plataformas Windows, en MacOS y en Linux, su interacción con los gráficos es muy pobre y carga mucho los ordenadores, esto es muy diferente al Javascript que es nativo del navegador y está optimizado para cada plataforma.

A fecha de hoy es conveniente dejar de usar Flash en las páginas WWW ya que para 2020 es posible que deje de funcionar definitivamente, tal como lo anuncia Adobe [4].

2.4. Tecnologías utilizadas

A continuación se va a describir con mayor detalle las tecnologías específicas que se proponen desde la empresa como alternativa existente.

2.4.1 MediaStream API

Como se ha explicado antes, la grabación y el procesado de audio en general es nuevo en los navegadores, mediante HTML5 y JS. HTML5 introduce la etiqueta <audio> que permite incrustar audio en páginas web para su reproducción. La grabación de audio fue introducida en 2010 por Mozilla [7] de manera unilateral, pero W3C propuso una API [8] que es la que se emplea actualmente, realizada en colaboración por Mozilla y Chrome. Aún así está en evolución y las aplicaciones JS que la emplean tienen que intentar mantener compatibilidad con las versiones que han aparecido en los últimos 5 años.

Esta API es mucho más que un conjunto de funciones. Se define el concepto de *AudioNode*, que representa tanto las fuentes de audio como la salida de audio y algunas formas de procesamiento incluidas en la API. Es importante aclarar que esta API no dispone de compresión de audio por lo que habrá que construir o buscar código ajeno para realizarlo.

Los *AudioNode* existen dentro de un *AudioContext*, que puede imaginarse como un grafo que interconecta los distintos nodos. La siguiente figura muestra un ejemplo muy básico:

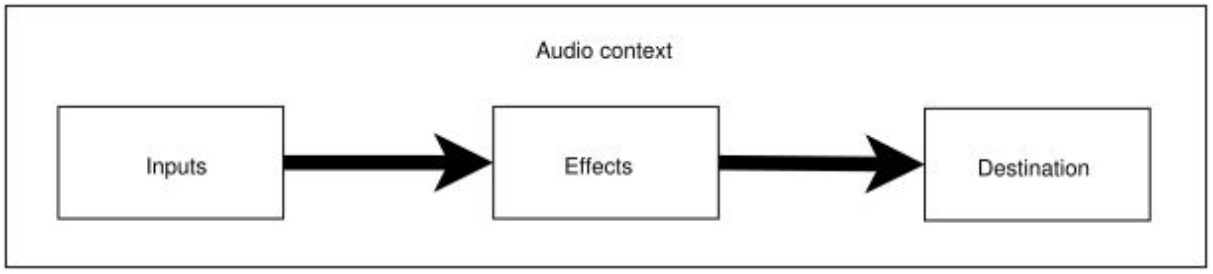


Figura 2.4.1: Estructura básica de nodos

Cuando se graba un audio se digitaliza [9] mediante un muestreo. El muestreo se realiza a 44KHz y 16 bits. Cada elemento del buffer de audio es un float32.

Para poder hacer la grabación, primero es necesario crear algún contexto. El objeto *audioContext* está asociado al propio objeto *navigator* por lo que se puede crear directamente con:

```
var audioCtx = new AudioContext();
```

Como se quiere obtener audio del micrófono, es necesario preparar un stream y un procesador, tal como sigue:

```
1 function Init (stream) {
2   var sourcenode  = audioCtx.createMediaStreamSource( stream );
3   var monitorNode = audioCtx.createGain();
4   sourcenode.connect(monitorNode);
5   var scriptProcessorNode = audioCtx.createScriptProcessor(1024,1,1);
6   scriptProcessorNode.onaudioprocess = function() { … }
7 }
```

Se ha creado un *stream* de audio (línea 2), un nodo para controlar volumen que se conecta con el nodo *sourcenode* dentro del contexto creado (línea 3). Y se ha preparado un procesador para que el buffer sea de 1024, con un canal de entrada y uno de salida (línea 5). El evento *onaudioprocess* se produce cada vez que se registren 1024 frames del micrófono (línea 6).

La verdadera toma de audio se realiza con *getUserMedia*, un método nativo de *navigator* en las primeras versiones de la API y actualmente conectado con *mediaDevices*. Lo invocamos con:

```
navigator.mediaDevices.getUserMedia.{audio: true}.then(Init,Error);
```

Donde *Error* es una función invocada en caso de error e *Init* la función definida antes.

2.4.2 Códec de audio

Un códec de audio es una forma de compresión de datos específica para sonido. Una fuente de audio (analógica) se muestrea habitualmente a 44KHz y se emplea una resolución de 16bit para garantizar que la calidad del sonido registrado sea toda la que el oído humano puede apreciar.

El problema de este formato (RAW) es que el flujo generado ocupa gran cantidad de memoria y en ciertas circunstancias, no es necesaria tanta calidad sonora. La compresión de audio consiste, pues, en reducir el consumo de recursos de un audio al mínimo requerido.

Existen modos de compresión sin pérdida, pero la tasa reducción de tamaño es baja, de modo que los compresores con pérdida son la elección conveniente en este proyecto. En los compresores con pérdida se maneja normalmente el concepto de bps, el número de bits que se requieren para representar un segundo de audio. Cuanto menor es el número de bits, más baja es la calidad, pero puede ser suficiente para una determinada aplicación [9].

2.4.3 Web Workers

Los *Web Workers* hacen posible ejecutar un script en un subproceso en segundo plano separado del subproceso de ejecución principal de una aplicación web. La ventaja es que el procesamiento costoso puede realizarse en un hilo separado, permitiendo que el hilo principal (usualmente la interfaz de usuario) se ejecute sin ser bloqueado y/o ralentizado [10].

Un *worker* se crea como un objeto que ejecuta un fichero JS separado, que contiene el código a ejecutar por el *worker*. El contexto global no es *window* como es lo habitual, sino uno propio.

Desde el *worker* ejecutamos cualquier código JS, pero no podemos acceder al DOM. La forma de comunicar el *worker* con su creador es mediante *postMessage()* y su correspondiente manejador de eventos. Los datos enviados se copian, no se comparten.

Por ejemplo se crea un *worker* y se comunica con él:

```
if (window.Worker) {
  var myWorker = new Worker("worker.js");
  myWorker.postMessage('Pedido');
  myWorker.onmessage = function(e) {
    console.log('Mensaje del worker' + e.data);
  }
}
```

El código del worker:

```
onmessage = function (e) {
   postMessage('Este es el mensaje');
}
```

2.4.4 Websockets

El paradigma clásico del HTTP responde a un modelo cliente servidor donde se realiza una petición desde el cliente que implica abrir un socket TCP. Después de ser atendida, la conexión se cierra básicamente para liberar recursos del servidor. Esta forma de trabajo es adecuada cuando el cliente solicita una página HTML estática, y es como se concibió el WWW en sus inicios. Pero cuando una página es realmente una aplicación web, pensemos por ejemplo en Facebook, cliente y servidor necesitan estar en permanente contacto. El cliente, sin romper el esquema, podría realizar un *polling* preguntando periódicamente al servidor si hay cambios que notificar al usuario. Surgen dos problemas, por una parte una posible desincronización y por otra un consumo alto de recursos de red [11].

Para resolver el problema de las notificaciones, los navegadores han introducido los *server-sent events*, que de forma totalmente integrada en JS, permiten recibir eventos del servidor. El servidor deberá tener un proceso (que consume recursos de memoria) que se mantenga conectado con el cliente y envíe los eventos cuando desee.

Los *websocket* son una forma más explícita de mantener una conexión permanente entre cliente y servidor. Siguen el protocolo HTTP, pero la conexión permanece abierta, salvo error, hasta que se cierra intencionadamente.

Su uso es muy similar al ya clásico AJAX, la diferencia estriba en que AJAX realiza una petición HTTP clásica, que no está pensada para permanecer abierta y que no admite mucho más allá de una operación GET o POST. En cambio, los *WebSockets* están pensados para una comunicación bidireccional y permanente de datos a modo de *stream* [12].

Los *WebSocket* están actualmente integrados en todos los navegadores y su creación consiste, como cualquier otro recurso, en la instanciación de una clase JS:

```
var wss= new WebSocket('wss://speech.actualpacs.com:8443/');
```

Existen dos esquemas para *websocket*, el esquema 'ws' que es para transmisión de datos en claro, y el 'wss' que se denomina "seguro" y consiste, como es de suponer, en añadir una capa TLS completa para dotar de autenticidad y confidencialidad la comunicación. El puerto típico en este caso es el 8443, que el servidor puede abrir sin necesidad de privilegios.

Con el servidor a la escucha, la instanciación implica abrir la conexión TCP, de modo que el socket queda abierto y ya se pueden enviar y recibir datos:

```
wss.addEventListener('open', function (event) {
    socket.send('Hola servidor');
});
wss.addEventListener('message', function (event) {
    console.log('Mensaje del servidor : ', event.data);
});
```

2.4.5 JSON

JavaScript Object Notation, abreviadamente JSON es una sintaxis para la serialización de datos, desde los tipos más simples a objetos. JSON es un subconjunto de JS (con pequeñas excepciones) que se ha impuesto en la práctica como una forma fácil de intercambiar datos entre aplicaciones, incluso cuando JS no está presente, por ejemplo en PHP [13].

La forma correcta de usar JSON desde JS es emplear siempre los métodos que hoy en día están en todos los navegadores actuales. Por ejemplo para convertir un texto JSON que representa un objeto en un objeto JS [14]:

```
var obj = JSON.parse('{ "name":"John", "age":30}');
```

Y para generar un texto JSON a partir de un objeto JS:

```
var obj = { "name":"John", "age":30, "city":"New York"};
var myJSON = JSON.stringify(obj);
```

2.4.6 Decompilador de Adobe Flash

Los objetos de Adobe Flash se distribuyen como archivos (extensión *.swf*). Estos archivos pueden ser descompilados mediante la herramienta adecuada. El decompilador, a partir del archivo *swf* devuelve un texto con el código fuente *ActionScript* o JS que se utilizó para crearlo. Para poder analizar el código fuente de la utilidad Flash proporcionada por Vocali, se ha empleado *Sothink SWF Decompiler* de *SourceTec*.

2.4.7 Stunnel

Aunque no es necesaria para desarrollar un proyecto como éste, en este caso lo ha sido, debido a problemas que se describen en el apartado 3.5 de esta memoria. Pese a su nombre, *stunnel* [15] es un *proxy* TLS, acepta una conexión TLS configurable con el certificado X509 que corresponda y redirige el tráfico a un sitio externo, que puede a su vez ser TLS.

Se expone el ejemplo utilizado en este proyecto, donde *stunnel* quedaría escuchando el puerto 8443, utilizando el certificado y la llave privada correspondiente. El certificado sería totalmente válido y corresponde a *nisu.org*. El tráfico es bidireccionalmente reenviando al puerto 8444 del servidor local, que a su vez es manejado por el propio *stunnel* y enviado a Vocali, puerto 8443. Se indica a *stunnel* que no debe verificar el certificado expuesto por Vocali, que no es válido, como se explica en el apartado 3.5.

Obsérvese que *stunnel* permitiría, por ejemplo mediante *strace*, espiar todo el tráfico entre el navegador y Vocali, permitiendo comprobar que la comunicación vía *WebSocket* es correcta.

```
[servidor]
accept  = 8443
connect = 8444
cert = /etc/letsencrypt/live/nisu.org/fullchain.pem
key = /etc/letsencrypt/live/nisu.org/privkey.pem

[cliente]
client = yes
accept = 127.0.0.1:8444
connect = actualmed.vocali.net:8443
verifyChain = no
```

Capítulo 3

3. Planificación del proyecto

En este capítulo se va a describir la metodología usada para llevar a cabo el proyecto, así como las tareas que se han llevado a cabo para su realización y cómo han sido planificadas. También se incluye una justificación de las desviaciones que se han producido respecto a la planificación inicial y finalmente un apartado donde se detallan los costes de los recursos empleados en el proyecto.

3.1. Metodología

Para el desarrollo y la planificación del proyecto se ha utilizado la metodología **PMBOK** [16], tal y como se puede ver en la Figura 3.1. Esta metodología está basada en procesos, y es dentro de estos donde se describe el trabajo realizado. Estos procesos se superponen e interactúan a lo largo de las fases del proyecto, y se describen en términos de entradas, herramientas, técnicas y salidas. Para ello, se descompone el proceso de desarrollo en cinco bloques, los cuales son:

1. Inicio, en esta etapa se define el proyecto a llevar a cabo, lo que se quiere obtener y si el proyecto es viable, así como la documentación que es necesario analizar para comprender el funcionamiento y las características de la tecnología que va a ser utilizada.

2. Planificación del proyecto, una vez es considerado el proyecto viable en la fase anterior se pasa desarrollarlo, que significa hacer la planificación detallada del proyecto y la de su programación, definiendo las tareas a realizar de acuerdo a los objetivos del proyecto, estableciendo unas fechas de inicio y fin con los hitos claros planificados.

3. Ejecución (Desarrollo técnico del proyecto): en esta etapa se materializan y/o adquieren los componentes hardware y se implementan los componentes software del sistema.

4. Seguimiento y control: Se medirá el rendimiento, se revisarán los informes anteriores y se comprobará si el proyecto ha avanzado. Si es necesario se corregirá el proyecto, se gestionan y se documentan los cambios.

5. Cierre: Se entrega definitivamente el proyecto, eso incluye las pruebas de rendimiento y la calidad y robustez del sistema, así como la documentación.

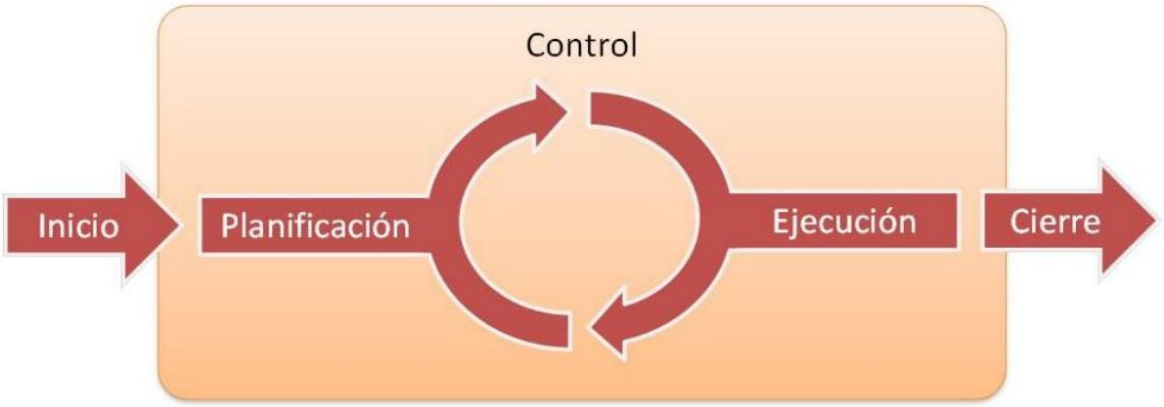


Figura 3.1: Esquema de la metodología PMBOK

3.2. Proceso de desarrollo

Para asegurar un software de mayor calidad, se ha aplicado la metodología TDD (Test Driven Development) en las diferentes fases del proyecto. Esta práctica de desarrollo de software gestiona los proyectos de tal forma que lo primero en implementar son las pruebas del sistema para, posteriormente, generar el código que satisface dichas pruebas [17]. La Figura 3.2 ilustra el ciclo de desarrollo de pruebas de éste método.

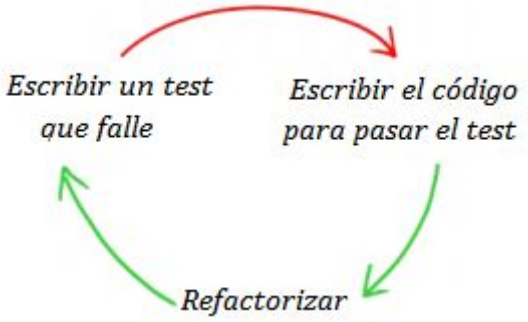


Figura 3.2: Ciclo del desarrollo de pruebas con TDD

3.3. Planificación del desarrollo

En este apartado se muestra la planificación seguida durante la realización del proyecto. La lista de tareas, la planificación temporal así como las dependencias entre ellas.

3.3.1 Definición de tareas

Para completar el proyecto se han llevado a cabo las siguientes tareas, las cuales se presentan agrupadas en etapas equivalentes a las de la metodología PMBOK:

- Planteamiento del proyecto y toma de requisitos

    1. Definir el proyecto con el tutor y el supervisor: el supervisor enuncia la idea general del proyecto y se procede a la toma de requisitos.

    2. Definir la metodología de trabajo y la documentación a utilizar: se ha procedido a revisar y buscar en la red toda la documentación que deberá ser utilizada. Así mismo, se ha estudiado la mejor forma de afrontar el desarrollo de un proyecto de estas características para un estudiante en prácticas.

    3. Definir las tareas y estimar el tiempo de ejecución: se han definido de forma general las tareas necesarias para completar el proyecto y se les ha asignado un tiempo de ejecución.

- Preparación previa

    4. Documentarse sobre el audio en HTML5 y JavaScript: se ha procedido a revisar y estudiar toda la documentación facilitada por la empresa y los manuales online para comprender el funcionamiento de las tecnologías utilizadas para implementar el proyecto.

    5. Instalar el software de trabajo: se ha instalado el software necesario para desarrollar el código del proyecto.

    6. Familiarizarse con las herramientas de trabajo software y hardware: se ha reservado un periodo de tiempo para familiarizarse con las tecnologías más

novedosas y poder realizar pruebas con ellas con el fin de comprender su funcionamiento y cómo tratar con ellas. Se han creado pequeños programas en HTML5 + Javascript para capturar audio del micrófono grabarlo y reproducirlo.

- Definición de las especificaciones del proyecto y diseño de la arquitectura

    7. Definir las especificaciones técnicas del proyecto: con los conocimientos adquiridos a través de la documentación y pruebas iniciales se han determinado las características del sistema y se han rechazado aquellos requisitos inviables.

    8. Diseñar la arquitectura del proyecto: se ha procedido a estudiar qué elementos software eran necesarios para cumplir con el propósito del proyecto. Se ha determinado que para la compresión y el resampleado del audio se utilizarán librerías externas, y se aprovechará código implementado de la plataforma anterior.

- Desarrollo técnico del proyecto. Se han desglosado las tareas (funcionalidades) de forma que al finalizar cada una de ellas se disponga de una parte funcional del sistema lista para ser utilizada en la posterior parte que la necesita. Las tareas implementadas son:

    9. Capturar audio del micrófono.
    10. Grabar audio.
    11. Comprimir audio.
    12. Codificar audio.
    13. Enviar audio al servidor.
    14. Recibir la transcripción y decodificarla.
    15. Mostrar el dictado por pantalla.

- Pruebas. Se han realizado diversas pruebas para guiar el desarrollo y para corroborar el buen funcionamiento de cada una de las tareas mencionadas anteriormente siguiendo la metodología TDD (Test Driven Development).

- Integración. Esta fase ha quedado pendiente por incompatibilidades que en el momento de acabar la estancia en prácticas estaban siendo estudiadas.

3.3.2 Planificación temporal

Las tareas descritas en el punto anterior se han organizado de forma cronológica tal y como aparece en la Tabla 3.3. Además, en esta tabla se han incluido las tareas docentes relacionadas con la preparación y de desarrollo de la memoria del TFG. Con el asesoramiento del supervisor de la empresa, se le ha asignado a cada una de las tareas el número de horas considerado más adecuado para llevarla a cabo. Además, se ha prestado especial atención en cubrir las 300 horas requeridas por la universidad a razón de una dedicación en la empresa de 25 horas semanales (5 horas diarias). En la Figura 3.3 se puede observar un diagrama de Gantt en el que se representan de forma gráfica las tareas distribuidas a lo largo del periodo de tiempo correspondiente a la estancia en prácticas.

Id	Nombre de tarea	Duración	Dependencias
1	**Trabajo Fin de Grado**	**450 h**	
2	**Inicio**	**67 h**	
3	Definir el proyecto con el tutor y el supervisor	2 h	-
4	Definir la metodología de trabajo y la documentación a utilizar	5 h	3
5	Definir formato y estándares de trabajo	5 h	3
6	Revisar el contexto y autoformación	50 h	5
7	Identificar alcance y objetivos	5 h	6
8	**Planificación**	**23 h**	
9	Definir tareas y estimar fechas	6 h	7
10	Crear el diagrama de Gantt	2 h	9
11	Redactar la propuesta técnica del proyecto	15 h	10
12	Entregar la propuesta técnica	0 h	11
13	**Ejecución (desarrollo técnico del proyecto)**	**200 h**	
14	Análisis	20 h	12
15	Diseño	10 h	14
16	Desarrollo	90 h	14, 15
17	Pruebas	80 h	16
18	**Seguimiento y control**	**10 h**	
19	Medir el rendimiento	10 h	17
20	**Cierre**	**150 h**	
21	Redactar informes quincenales	10 h	14, 15, 16, 17
22	Redactar la memoria del proyecto	110 h	21
23	Entregar la memoria del proyecto	0 h	22
24	Preparar la presentación	29 h	23
25	Realizar la presentación oral	1 h	24
26	**Fin de proyecto**	0 h	25

Tabla 3.3: Desglose de tareas y asignación del tiempo de ejecución en horas

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Trabajo Fin de Grado**	**450 horas**	**lun 13/03/17**	**vie 15/09/17**	
2	**Inicio**	**67 horas**	**lun 13/03/17**	**vie 31/03/17**	
3	Definir el proyecto con el tutor y el supervisor	2 horas	lun 13/03/17	lun 13/03/17	
4	Definir la metodología de trabajo y la documentación a utilizar	5 horas	lun 13/03/17	mar 14/03/17	3
5	Definir formato y estándares de trabajo	5 horas	mar 14/03/17	mié 15/03/17	3
6	Revisar el contexto y autoformación	50 horas	mié 15/03/17	jue 30/03/17	5
7	Identificar alcance y objetivos	5 horas	jue 30/03/17	vie 31/03/17	6
8	**Planificación**	**23 horas**	**lun 06/03/17**	**lun 06/03/17**	
9	Definir tareas y estimar fechas	6 horas	vie 31/03/17	lun 03/04/17	7
10	Grear el diagrama de Gantt	2 horas	lun 03/04/17	lun 03/04/17	9
11	Redactar la propuesta técnica del proyecto	15 horas	mar 04/04/17	jue 06/04/17	10
12	Entregar la propuesta técnica	0 horas	jue 06/04/17	jue 06/04/17	11
13	**Ejecución**	**200 horas**	**mar 28/03/17**	**lun 01/05/17**	
14	Análisis	20 horas	vie 07/04/17	mié 12/04/17	12
15	Diseño	10 horas	jue 13/04/17	vie 14/04/17	14
16	Desarrollo	90 horas	mar 18/04/17	vie 12/05/17	14;15
17	Pruebas	80 horas	lun 15/05/17	lun 05/06/17	16
18	**Seguimiento y control**	**10 horas**	**mar 06/06/17**	**jue 08/06/17**	
19	Medir el rendimiento	10 horas	mar 06/06/17	jue 08/06/17	17
20	**Cierre**	**150 horas**	**lun 13/03/17**	**vie 15/09/17**	
21	Redactar informes quincenales	10 horas	lun 13/03/17	jue 08/06/17	14;15;16;17
22	Redactar la memoria del proyecto	110 horas	sáb 01/07/17	lun 31/07/17	21
23	Entregar la memoria del proyecto	0 horas	lun 31/07/17	lun 31/07/17	22
24	Preparar la presentación	29 horas	vie 01/09/17	jue 14/09/17	23
25	Realizar la presentación oral	1 hora	vie 15/09/17	vie 15/09/17	24
26	Fin de proyecto	0 horas	vie 15/09/17	vie 15/09/17	25

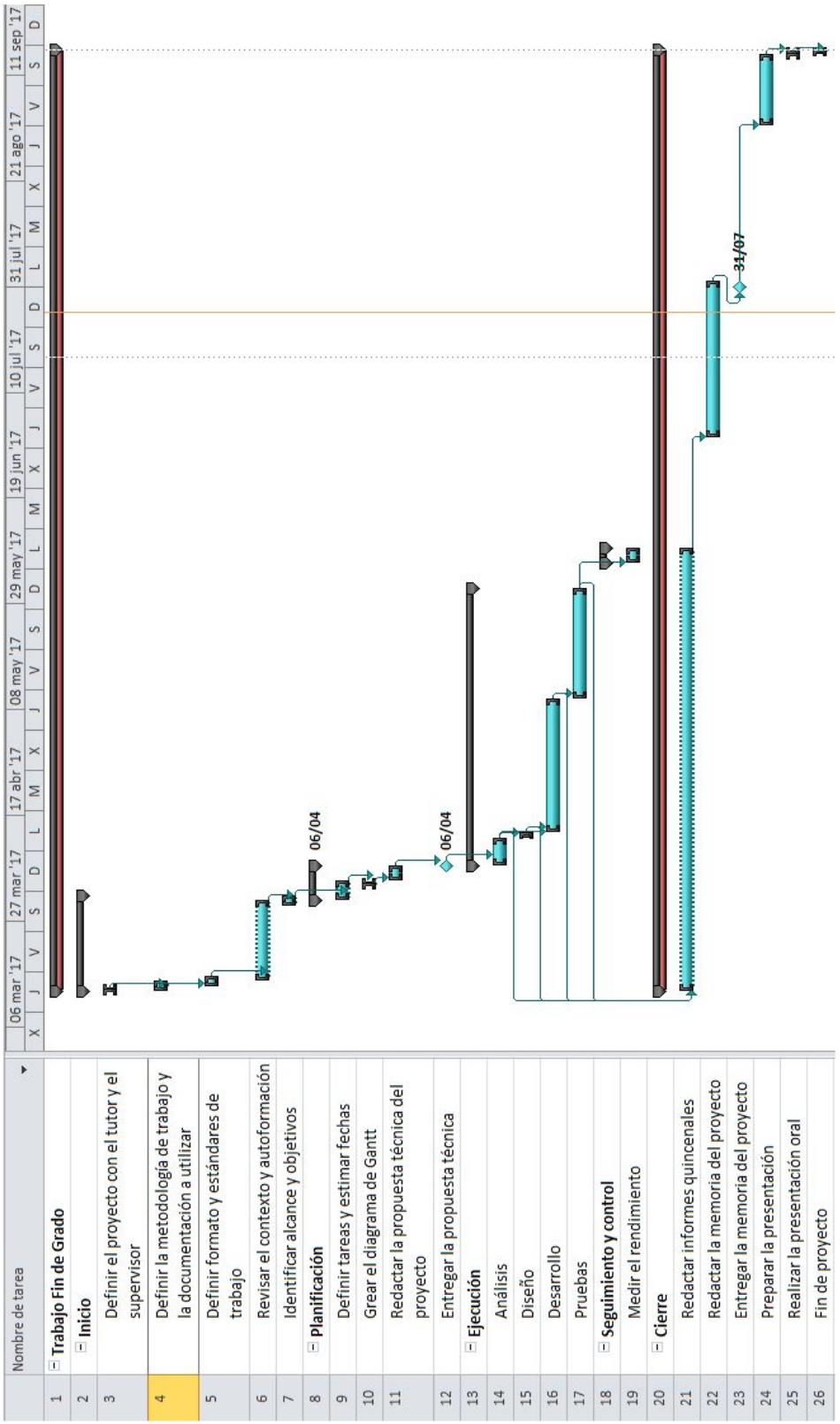


Figura 3.3: Diagrama de Gantt

3.4. Estimación de recursos y costes del proyecto

Al inicio de cada proyecto se debe hacer una estimación de los recursos necesarios para su realización. Estos recursos se pueden clasificar en recursos software, hardware y humanos. Por lo que a continuación se muestran las estimaciones de costes de los recursos utilizados en este proyecto. Se estima un total de 1896.25€ que se desglosa a continuación.

3.4.1 Recursos Software

Nombre del recurso	Coste (amortización 4 años)
Windows 7 profesional	120€ /4 /4 =7€
Virtual Box	0€
Git Hub private repositories	0€
phpStorm license student	0€

Tabla 3.4.1: Estimación de recursos software

3.4.2 Recursos Hardware

Nombre del recurso	Coste
Ordenador HP Compaq	600 € /4 /4 = 37 €
Monitor LG	120 € /4 /4 = 7 €
Teclado + Ratón Logitech	30 € /4 /4 = 2 €
Micrófono Olympus RecMic DR-1200	250 €/4/4 =31.25 €

Tabla 3.4.2: Estimación de recursos hardware

3.4.3 Recursos Humanos

En cuanto a coste de recursos humanos, partiendo de un coste de 6€/hora para un programador junior, trabajando 25 horas semanales. El total viene a ser (300 horas * 6€/hora) 1800€.

3.4.4 Otros

Nombre del recurso	Coste
Libro HTML5 para Mentes Maestras (Versión Google Play)	12 €

Tabla 3.4.4: Estimación de otros recursos

3.5. Seguimiento del proyecto

Todas las fases de este proyecto en general han sido supervisadas por el tutor y el supervisor de la empresa mediante informes quincenales en los que se han descrito las tareas llevadas a cabo durante la quincena así como los objetivos a cumplir para el siguiente período.

Respecto a la parte del desarrollo técnico, se han establecido unos objetivos funcionales para cada bloque, así como a qué entradas debe responder y qué salidas debe producir para el siguiente bloque o bloques. Tras la consecución de un bloque (por ejemplo, la captura del audio) se han realizado pruebas individuales con el fin de comprobar que dicha parte del sistema es capaz de responder a las entradas definidas y producir la salida esperada.

No obstante, ha habido algunos problemas técnicos que si bien no han afectado a las tareas a llevar a cabo o al orden de ejecución, sí que han conllevado retrasos en el tiempo total que se había previsto para algunas de ellas.

Uno de esos problemas, es que durante las pruebas iniciales con *WebSocket* se encontró con que no se podía conectar con el servidor que aloja la API de Vocali, es decir no se podía conectar con la URL que la empresa facilitó para enviar el audio mediante *WebSocket* TLS. Empleando el depurador se observó que había un problema de seguridad, detectado por el navegador. Tras estudiar lo necesario sobre certificados TLS, se observó que el problema era que el certificado empleado no era de una autoridad válida, sino un certificado autofirmado emitido por Vocali y que requiere instalar una excepción en el navegador.

Para no tener que hacer esto en cada navegador y teniendo en cuenta que no se veía una pronta solución, se decidió utilizar un *proxy* TLS como se explica en el apartado 2.4.7 de esta memoria. La empresa ActualMed suministró a Vocali un certificado válido y tras varios intentos por parte de ambos, el sistema quedó funcionando sin necesidad del *proxy*.

La fase de pruebas se extendió más de lo previsto porque la librería empleada en la compresión de audio [18] daba unos resultados deficientes en la compresión de audio. El audio comprimido con *Speex* tenía la calidad deseada a 8KHz, según las pruebas preliminares realizadas con los ejemplos suministrados por el autor, pero producía resultados deficientes a 16KHz que no estaban documentados en la librería como un bug. La detección de este problema requirió de numerosas pruebas, en tanto que el problema se achacaba inicialmente a la interacción con la plataforma de transcripción INVOX, concretamente a las ratios de envío, como se detalla en el apartado 5.2 de pruebas realizadas. Esta es la causa principal de que el proyecto no se haya completado en las horas establecidas, dado que el cambio de librería de compresión se realizó la última semana y la integración se ha dejado fuera del alcance del proyecto.

Capítulo 4

4. Análisis y diseño del sistema

4.1 Análisis del sistema

4.1.1 Requisitos

El proyecto llevado a cabo ha sido motivado por la propia empresa para resolver problemas causados por el Flash a los clientes de la aplicación. A continuación se van a presentar esos problemas:

- Cuando se inicia la captura de audio, el plugin de Flash muestra automáticamente una ventana que pide permiso al usuario (Figura 4.1), pero por problemas de renderizado de subpixel, a veces es muy difícil hacer click en el botón Permitir en Mozilla Firefox.

Figura 4.1: Ventana de permiso de Flash

- El Flash impide a los usuarios del sistema realizar dictados de voz a través de un dispositivo móvil o tablet, ya que Flash se ha declarado obsoleto (no soportado) en esos dispositivos.

- Dificultades por parte de los clientes para la instalación y actualización del plugin Flash.

- Necesidad de configuración extra para activar el Flash en algunos navegadores.

Los problemas presentados, dan lugar a unos requisitos de calidad que se han de cumplir, especialmente aquellos requisitos que son de importancia para los usuarios, que son: disponibilidad, eficiencia, flexibilidad, integridad, interoperabilidad, robustez y usabilidad.

Por otro lado el sistema tiene que seguir cumpliendo los requisitos que venía haciendo hasta ahora, esos requisitos se pueden clasificar como requisitos funcionales; definiéndolos como las funciones que éste sistema ha de proporcionar al usuario, y que ya se habían descrito en el alcance, estos requisitos son:

- Grabar audio, el sistema debe permitir al usuario grabar audio del micrófono desde el navegador.

- Administrar audio, el sistema debe permitir al usuario administrar el audio grabado; eso implica; reproducirlo, pausarlo, guardarlo, descargarlo y/o eliminarlo.

- Transcribir audio a texto, el sistema debe permitir enviar el audio y obtener la transcripción correspondiente de INVOX en tiempo real.

- Corregir una transcripción, el sistema debe permitir al usuario corregir una transcripción realizada.

- Generar informe, el sistema debe permitir al usuario generar un informe a partir de la transcripción realizada en formato pdf.

Aunque los requisitos funcionales no son muchos, la complejidad del proyecto reside en las diferentes tecnologías y su interoperabilidad con la API de Vocali. Emplear los servicios de Vocali es un requerimiento de la empresa y forma parte de la especificación del proyecto.

4.1.2 Diagrama de casos de uso

En la Figura 4.1.2 se muestra el diagrama de casos de uso correspondiente al sistema del proyecto teniendo en cuenta los requisitos funcionales explicados en el apartado anterior. En dicho diagrama se representan los diferentes actores del sistema y las distintas interacciones que llevan a cabo.

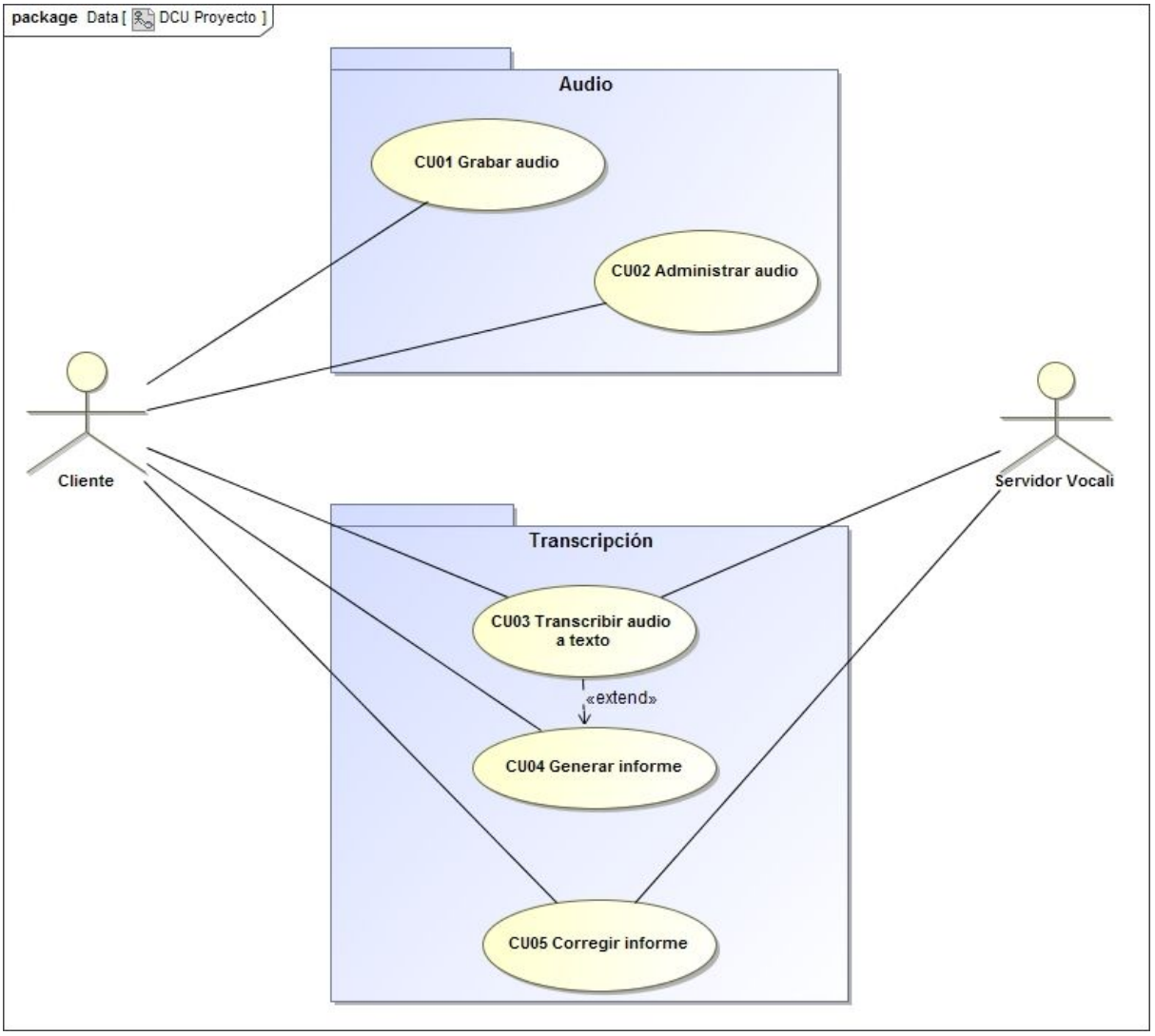


Figura 4.1.2: Diagrama de casos de uso

A continuación se procede a describir los casos de uso que conforman el diagrama de la Figura 4.1.2:

Descripción caso de uso: CU01 Grabar audio	
Identificador: CU01 **Nombre**: Grabar audio **Autor**: HARBAZ **Fuente:** *ActualMed*	**Fecha de creación**: 01/04/2017 **Fecha revisión:** 03/06/2017 **Fecha aprobación:** **Versión:** 1
Descripción: *Este caso de uso representa la funcionalidad que permite al usuario de la plataforma grabar audio desde un micrófono.*	
Escenario: **1.** El sistema solicita permiso para el acceso al micrófono **2.** El usuario permite tal acceso **3.** El usuario comienza a grabar **4.** El sistema recoge audio del micrófono. **5.** El sistema muestra el audio grabado.	
Excepciones:	
Precondiciones: Un micrófono.	
Comentarios:	

Descripción caso de uso: CU02 Administrar audio	
Identificador: CU02 **Nombre**: Administrar audio **Autor**: HARBAZ **Fuente:** *ActualMed*	**Fecha de creación**: 01/04/2017 **Fecha revisión:** 03/06/2017 **Fecha aprobación:** **Versión:** 1
Descripción: *Este caso de uso representa la funcionalidad que permite al usuario de la plataforma administrar audio, esto permite; reproducir audio, pausarlo, pararlo, descargarlo y/o eliminarlo,*	
Escenario: 1. El sistema permite al usuario elegir una acción para el audio (reproducir, parar, pausar, etc) 2. El usuario selecciona la acción que le interesa. 3. El sistema reproduce la acción solicitada	
Excepciones:	
Precondiciones: Audio grabado	
Comentarios:	

Descripción caso de uso: CU03 Convertir voz en texto	
Identificador: CU03 **Nombre**: Convertir voz en texto **Autor**: HARBAZ **Fuente:** *ActualMed*	**Fecha de creación**: 01/05/2017 **Fecha revisión:** 03/06/2017 **Fecha aprobación:** **Versión:** 1
Descripción: *Este caso de uso representa la funcionalidad que permite al usuario enviar audio y obtener su transcripción en tiempo real.*	
Escenario: 1. El usuario dicta voz a través del micrófono. 2. El sistema captura el audio del micrófono desde el navegador. 3. El sistema comprime y codifica el audio. 4. El sistema envía segmentos de audio al servidor VOCALI. 5. El sistema recibe la transcripción en formato JSON y la transforma en texto,	
Excepciones: - VOCALI no acepta algún segmento de audio. - El usuario no acepta alguna o varias palabras de la transcripción.	
Precondiciones:	
Comentarios: Para el dictado de voz, la grabación y la transcripción deben ser simultáneas (en tiempo real). Vocali va añadiendo texto a medida que se registra el audio y mejorando contenidos que podía haber ignorado previamente. La transcripción también va evolucionando con la mejora de los contenidos .	

Descripción caso de uso: CU04 Generar informe	
Identificador: CU04 **Nombre**: Generar informe **Autor**: HARBAZ **Fuente:** *ActualMed*	**Fecha de creación**: 01/05/2017 **Fecha revisión:** 03/06/2017 **Fecha aprobación:** **Versión:** 1
Descripción: *Este caso de uso representa la funcionalidad que permite al usuario generar un informe a través de una plantilla predefinida y una transcripción.*	
Escenario: 1. El usuario elige una configuración de la plantilla del informe 2. El sistema genera un informe médico acorde a la configuración elegida y con la transcripción realizada.	
Excepciones:	
Precondiciones: transcripción y plantilla del estilo del informe	
Comentarios: El informe se genera en formato pdf	

Descripción caso de uso: CU05 Corregir una transcripción	
Identificador: CU05 **Nombre**: Corregir una transcripción **Autor**: HARBAZ **Fuente:** *ActualMed*	**Fecha de creación**: 30/05/2017 **Fecha revisión:** 03/06/2017 **Fecha aprobación:** **Versión:** 1
Descripción: *Este caso de uso representa la funcionalidad que permite al usuario de la plataforma corregir una secuencia de palabras en una transcripción realizada*	
Escenario: 1. El usuario selecciona el dictado a corregir 2. El usuario selecciona la sentencia a corregir. 3. El sistema responde con una posible corrección. 4. Si la corrección es la deseada se acepta, en caso contrario, se vuelve a repetir el paso 2.	
Excepciones:	
Precondiciones: Transcripción	
Comentarios:	

4.2 Diseño de la arquitectura del sistema

El diseño en este proyecto, ha implicado la elección de los elementos software y su interacción para conseguir el producto final. Tanto el diseño de la interfaz como el de la base de datos se mantienen de la anterior plataforma en funcionamiento. A partir del análisis del sistema se ha definido la arquitectura representada en la Figura 4.2.

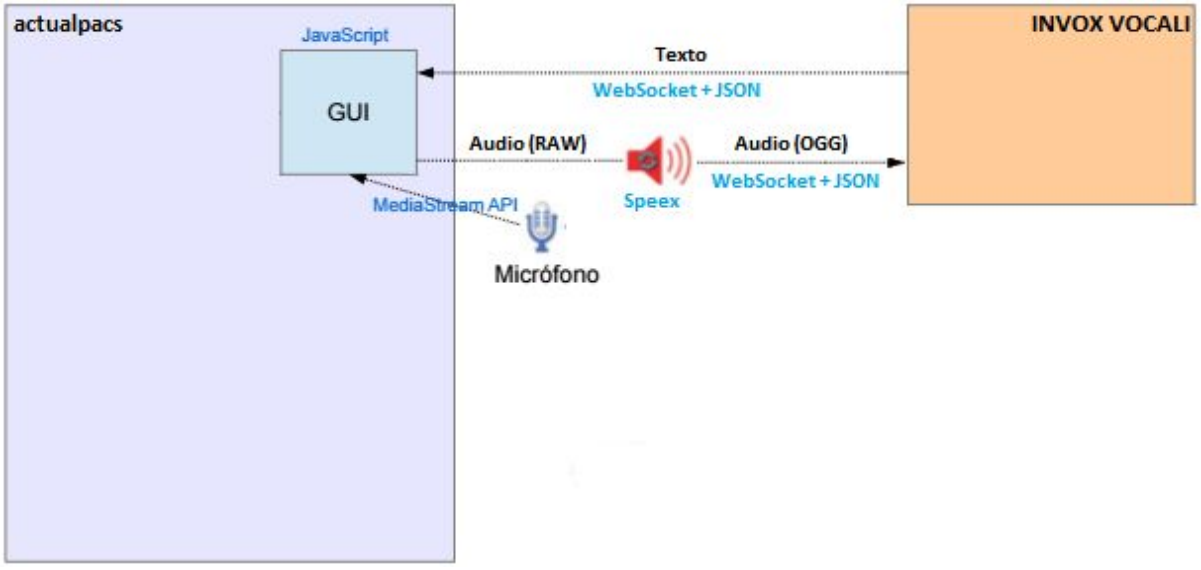


Figura 4.2: Esquema de tecnologías usadas y flujo de la aplicación

El requerimiento del proyecto era utilizar exclusivamente HTML5+JavaScript, por tanto, se han escogido las tecnologías que parecen más adecuadas para cada una de las fases de la implementación.

- **Captura de audio,** se ha elegido la Web Audio API [8], que es la tecnología recomendada en 2017 para la captura de audio.

- **Para el remuestreo y la compresión,** Javascript no dispone de ningún nodo procesador que permita el remuestreo y/o la compresión, por tanto se ha buscado librerías de código abierto. Para minimizar el consumo de recursos, se opta por remuestrear el audio a una frecuencia menor, 16KHz. Se elegirá uno de los codecs aceptados por el servidor INVOX, que son:

    - **Speex** el formato recomendado por el servidor, por su alto nivel de compresión con muy baja pérdida de calidad. El servidor sólo acepta el modo WB (WideBand).
    - **Opus**. Es el sucesor de Speex y consigue un pequeño aumento de calidad a costa de un mayor consumo de ancho de banda.

- **Para la comunicación con el servidor**, INVOX de Vocali requiere del uso de WebSocket con JSON que son nativos de Javascript.

Capítulo 5

5. Implementación y pruebas

En este capítulo se detallan las funcionalidades más importantes del desarrollo del sistema de transcripción y las pruebas llevadas a cabo teniendo en cuenta que se ha seguido un enfoque TDD para corroborar el funcionamiento general de todos sus componentes en conjunto.

5.1 Detalles de implementación

En la Figura 5.1 se muestra un esquema lógico de los ficheros que componen el producto desarrollado y que se detallan en los siguientes puntos.

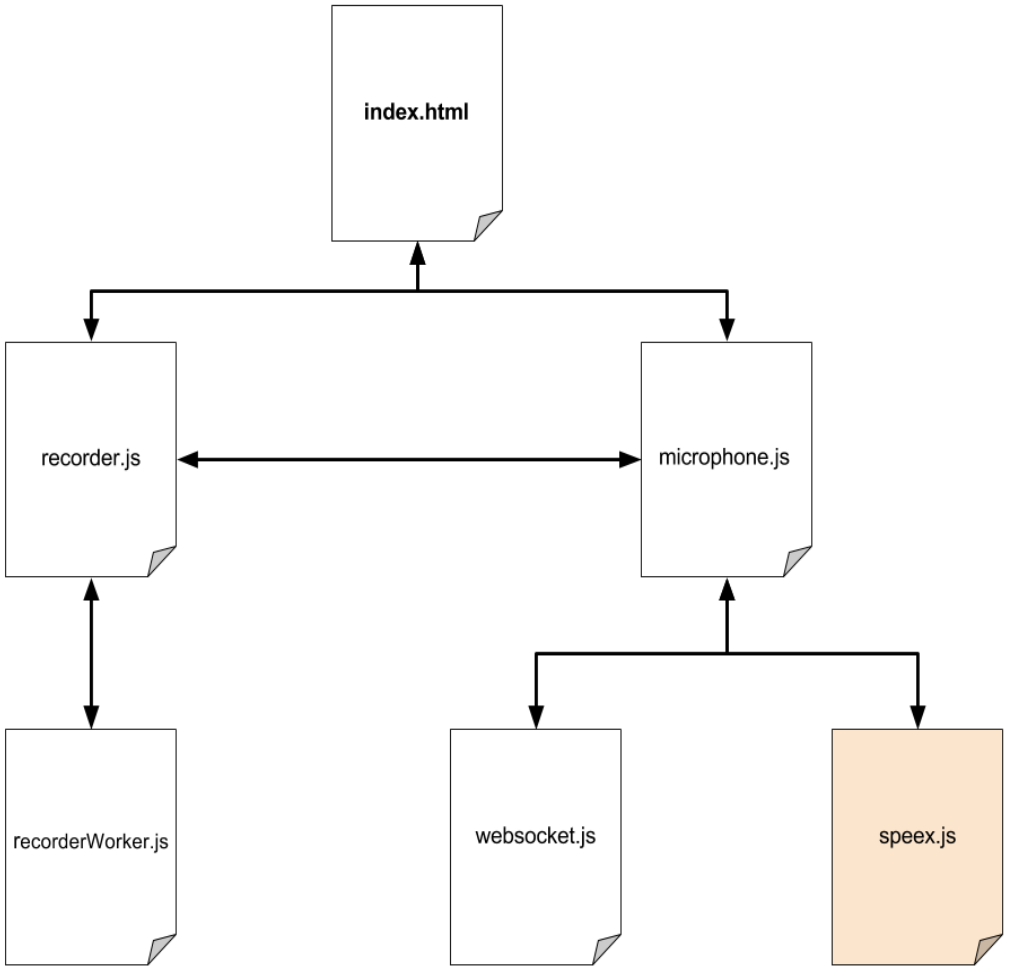


Figura 5.1: Esquema lógica de ficheros

Las funcionalidades incluidas en los ficheros del esquema anterior son las siguientes:

- Apertura del webSocket
- Grabación de audio.
- Captura de audio del micrófono
- Remuestreo de audio.
- Compresión de audio.
- Codificación de audio.
- Envío de audio.
- Recepción de la transcripción.

5.1.1 Interfaz principal

Se ha implementado una interfaz básica (index.html), aunque esta interfaz no será la utilizada como plataforma cliente definitiva, sí que sirve para realizar las diferentes pruebas de la transcripción y grabado de audio, para su posterior integración.

Como se puede ver en la Figura 5.2, hay una parte de registro que contiene las credenciales del usuario para poder establecer una comunicación posteriormente con el servidor INVOX de Vocali, una grabación de audio y su administración. Y finalmente, una transcripción de voz a texto en tiempo real mientras se dicta el audio.

Figura 5.2: Interfaz básica para pruebas

5.1.2 Apertura del webSocket

Para comenzar el dictado, se establece una conexión segura con el servidor INVOX de Vocali sobre el puerto 8443, una vez abierto el socket se puede comenzar a enviar mensajes al servidor. El servidor de dictado elimina las sesiones que no han sido utilizadas en los dos últimos minutos. Para evitar una eliminación involuntaria, INVOX recomienda enviar un mensaje ('KeepAlive') cada 30 segundos para mantener la sesión activa. Tal como se puede ver en el siguiente código:

```
var serverHost = 'speech.actualpacs.com:8443';
var host = "wss://" + serverHost;
window.wss = new WebSocket(host);
window.wss.onopen = function(){
   console.log("Socket abierto");
   setInterval('keepAlive()', 30000); //Mantener la conexión abierta
};
```

Código 5.1.2: Código de apertura del WebSocket

En el fichero 'websocket.js', además de la apertura del websocket, se sobredefine el método de la recepción de los mensajes del servidor, para mostrar la transcripción en el formato adecuado.

5.1.3 Captura del audio

El archivo 'microphone.js' es el que invoca al método getUserMedia para enlazar con el micrófono, como se ha explicado en el apartado 2.4.1. El método getUserMedia recibe como parámetro una función de inicialización. En este caso, esta función instancia la clase 'Recorder' (ver apartado 5.1.4) asociando la fuente de audio al micrófono. Lo más significativo que realiza este archivo es definir un manejador de evento que se invoca cuando el buffer de audio está preparado, tal como se explicó en el apartado 2.4.1. El tratamiento del evento realiza el resto de las funciones requeridas por la plataforma (remuestreo, compresión, codificación y envío).

```
var processor = audioContext.createScriptProcessor(1024, 1, 1);
processor.onaudioprocess = function (event) {
   //Remuestreo de audio
   ...
   //Compresión de audio
   ...
   //Codificación de audio en base64
   ...
   //Envío de audio
   ...
}
```

Código 5.1.3: Tratamiento del evento del audio

5.1.4 Grabación de audio

El fichero 'recorder.js' está implementado para grabar y exportar audio en la página web. Además permite la administración del audio (reproducirlo, pararlo y/o pausarlo). En este fichero, se define una clase 'Recorder', cuya misión es implementar un grabador de audio sin especificar la fuente (según se explica en el apartado 2.4.1) Para acelerar algunas operaciones de manejo buffer, la clase 'Recorder' se comunica con un web Worker ('recorderWorker.js'). Este worker, además de las operaciones sobre buffers permite la exportación a formato WAV para que el audio se puede escuchar si es necesario. Un posible uso de la clase 'Recorder' se muestra a continuación.

```
var microfono = audioContext.createMediaStreamSource( stream );
var recorder  = new Recorder(microfono);
recorder.record();       //Grabar
recorder.stop();          //Parar la grabación
//Reproducir
recorder.exportWAV(function(s) {
      audio.src = window.URL.createObjectURL(s);
});
```

Código 5.1.4: Ejemplo de utilización de la grabación de audio

5.1.5 Remuestreo de audio

El modelo acústico utilizado en el servidor externo ha sido entrenado a una frecuencia de muestreo de 16KHz con 1 canal y formato de muestra entero con signo de 16 bits LE. Sin embargo, el audioContext de la API webAudio trabaja por defecto con una frecuencia de 44.1KHz estéreo 16-bit, permitiendo elegir el número de canales a usar pero no la frecuencia. Para lograrlo, se ha empleado una librería de remuestreo de jpemartins [18] que permite

reducir la frecuencia de muestreo efectiva a 16KHz de modo que el audio ocupa menos espacio en memoria.

5.1.6 Compresión de audio

Según la especificación de INVOX, por su alta tasa de compresión, se ha hecho uso de una librería externa de Speex [18]. La librería permite la compresión de muestras a 8KHz y 16KHz. La librería necesita mejoras como realizar el procesado por WebWorker, según indica el propio autor, pero es la única librería que se ha encontrado para Javascript, aparte que la empresa Actualmed ya tenía experiencia con esta librería.

Durante la fase de pruebas se han encontrado problemas graves de la librería que se explicarán en el apartado 5.2.

5.1.7 Codificación de audio

El pasar una información a su representación de base64 nos brinda la seguridad de que no habrán problemas al transmitirla, almacenarla o leerla. Por lo que el audio pasa por esta fase para su posterior envío. Se ha aprovechado la implementación hecha en la plataforma anterior que convierte un array de bytes a base64.

5.1.8 Envío de audio

Al servidor se le envía un mensaje por vía webSocket, el mensaje consta de una marca que identifica el tipo de codec utilizado y una lista de segmentos de audio capturados durante el período de envío establecido. Este período se establece entre 100 y 200 milisegundos; más pequeño que 100ms provoca fallos en las interfaces de red y mayor que 200ms perjudica la interactividad del dictado. Tal como viene indicado en los requerimientos del servidor. Manteniendo un flujo constante y continúo de paquetes de audio se logra evitar deficiencias en el reconocimiento de voz.

5.1.9 Recepción de la transcripción

El servidor envía a la aplicación cliente información asíncrona en forma de objetos JSON para notificar al cliente. Cuando el dictado es aceptado, se muestra por pantalla, en caso contrario, simplemente se ignora. Aunque un audio es rechazado en un principio por el servidor, porque no lo logra reconocer, conforme avanza el dictado, la transcripción se va mejorando.

5.2 Escenario de pruebas

A continuación se van a explicar las distintas pruebas que se han hecho para el envío del audio al servidor, siguiendo la metodología TDD (Test Driven Development). El tamaño del buffer del audio no viene establecido por el servidor, el número de segmentos que ha de enviar tampoco está establecido, por lo que se ha tenido que recurrir a una metodología de prueba/error para obtener la transcripción deseada. (Las pruebas realizadas se han hecho con la librería Speex para la compresión).

1. **Enviar un segmento de audio.**

Dependiendo del tamaño del buffer (potencia de 2; comenzado en 256 hasta 8192), se enviará un segmento de audio cada (tamaño del buffer/44100) segundos.

Tamaño de buffer	Tiempo de envío (ms)	Resultado de transcripción
256	5.81	Ninguna respuesta del servidor
512	11.61	Ninguna respuesta del servidor
1024	23.22	Ninguna respuesta del servidor
2048	46.44	Ninguna respuesta del servidor
4096	92.88	Ninguna respuesta del servidor
8192	185.76	Ninguna respuesta del servidor

Tabla 1: Pruebas realizadas con 1 segmento

Como se puede apreciar en la tabla 1, independientemente del tamaño del buffer enviando sólo un segmento de audio, el servidor no responde con ningún mensaje.

2. **Enviar 10 segmentos de audios.**

En esta prueba, lo que se hace es implementar una cola, donde se van acumulando segmentos de audio, cuando se llega a 10 segmentos se envían al servidor.

Tamaño de buffer	Tiempo de envío (ms)	Resultado de transcripción
256	50.81	Ninguna respuesta del servidor
512	110.61	Ninguna respuesta del servidor
1024	230.22	Ninguna respuesta del servidor
2048	460.44	Ninguna respuesta del servidor
4096	920.88	Ninguna respuesta del servidor
8192	1850.76	Ninguna respuesta del servidor

Tabla 2: Pruebas realizadas con 10 segmentos diferentes

En la tabla 2, se puede ver que enviando 10 segmentos de audio se sigue sin obtener respuestas del servidor.
Se vuelven a enviar 10 segmentos, pero esta vez, se envía el segmento de audio repetido 10 veces cada (tamaño del buffer/44100) segundos.

Tamaño de buffer	Tiempo de envío (ms)	Resultado de transcripción
256	5.81	Respuesta del servidor errónea
512	11.61	Respuesta del servidor errónea
1024	23.22	Respuesta del servidor errónea
2048	46.44	Respuesta del servidor errónea
4096	92.88	Respuesta del servidor errónea
8192	185.76	Respuesta del servidor errónea

Tabla 3: Pruebas realizadas con 10 segmentos iguales

En la tabla 3, se observa que enviando el segmento duplicado 10 veces, el servidor responde aunque como es de esperar, no con la transcripción correspondiente al audio enviado.

3. Enviar un conjunto de segmentos de audio

En esta prueba, se ha optado por enviar un conjunto de segmentos de audio, conseguidos en un período de tiempo entre 100 y 200 ms empleando un timer.

a. Cada 100 ms

Tamaño de buffer	Nº de segmentos	Resultado de transcripción
256	20	Respuesta del servidor errónea
512	9	Respuesta del servidor errónea
1024	4	Respuesta del servidor errónea
2048	2	Ninguna respuesta del servidor
4096	1	Ninguna respuesta del servidor
8192	0	Ninguna respuesta del servidor

Tabla 4: Pruebas realizadas con un conjunto de segmentos (100 ms)

El número de segmentos reflejados en la tabla 4, corresponde al número de segmentos capturado cada 100 ms. Como se puede ver en esa tabla, a partir de 4 segmentos el servidor envía una posible transcripción aunque no la correcta.

b. Cada 200 ms

Tamaño de buffer	Nº de segmentos	Resultado de transcripción
256	40	Respuesta del servidor errónea
512	18	Respuesta del servidor errónea
1024	8	Respuesta del servidor errónea
2048	4	Respuesta del servidor errónea
4096	2	Ninguna respuesta del servidor
8192	1	Ninguna respuesta del servidor

Tabla 5: Pruebas realizadas con un conjunto de segmentos (200 ms)

En la tabla 5, se obtiene la misma conclusión obtenida de la tabla 4, a partir de 4 segmentos el servidor responde pero no con la transcripción que corresponde al audio enviado.

Con ninguna de las diferentes pruebas mostradas arriba, se ha obtenido una transcripción óptima. Usando Flash se obtiene un segmento de audio cada 20 ms con frecuencia de 16KHz y

comprimido con Speex. Sin embargo en el caso de Javascript, en el mejor de los casos, se obtiene un segmento de audio cada 23,22ms, a eso se le añade el tiempo de remuestreo más el tiempo de compresión y además el tiempo de codificación a base64, por lo que se acaba obteniendo un fragmento de audio de 4-7 segmentos en un intervalo de tiempo entre 100 y 200 milisegundos.

La librería de Speex [18] sólo permite usos de banda estrechos (NarrowBand) y al reproducir el audio comprimido por esta librería a una frecuencia de 16KHz se escuchaba con mucho ruido con lo que las transcripciones no eran las adecuadas, además el servidor sólo acepta el modo de banda ancho (WideBand). Ante este resultado se optó por probar el otro codec aceptado por Vocali, Opus, este codec es el sucesor de Speex y consigue un pequeño aumento de calidad a costa de un mayor consumo de ancho de banda. La librería elegida para usar éste último codec, permite elegir la frecuencia de muestreo, un muestreo de 8 KHz a 48 KHz (FullBand). Eligiendo los 16KHz requeridos por Vocali, el audio comprimido, si se reproduce antes de enviarlo, se escucha perfectamente.

Por otra parte, se planeó utilizar Web Workers, para ejecutar la compresión en segundo plano mientras se capta el audio, así se logra mejorar los tiempo de ejecución. Estas alternativas no se han llegado a comprobar por surgir en los últimos días de la estancia en prácticas.

Capítulo 6

6. Conclusiones

6.1 Conclusiones técnicas

El objetivo de este proyecto era eliminar la dependencia de Adobe Flash en el proceso de reconocer la voz en la dictación médica a través de una interfaz web, sustituyendo Flash por HTML5 y Javascript.

Se ha construido una página web que permite recoger el audio del micrófono del sistema donde se carga la página web, cambiar el muestreo, comprimirlo, codificarlo y enviarlo a la plataforma de reconocimiento de voz INVOX de Vocali.

La principal herramienta empleada ha sido el Web Audio API, que permite realizar la grabación en formato RAW. Por los requerimientos de la plataforma INVOX, el audio debe comprimirse mediante Speex u Opus [19] y enviarse a Invox con un determinado ritmo.

La compresión del audio ha sido la principal dificultad del proyecto en tanto que la Web Audio API no provee funciones para la compresión de audio. Ha sido necesario buscar librerías *opensource* para tal fin.

Los resultados de la codificación con la librería *speex* no han sido los esperados, mientras que la librería *opus* obtiene mejores resultados, aunque su validación final completa depende de la respuesta del servidor INVOX.

Los objetivos planteados en el proyecto se han conseguido parcialmente o casi todos. La validación pendiente responde principalmente al funcionamiento incorrecto de la librería *speex* que es la primera por la que se apostó, y a que no se ha dispuesto de tiempo suficiente para completar las pruebas con la librería *opus* que parece dar mejores resultados.

6.2 Conclusiones personales

A nivel personal, realizar este proyecto ha tenido diversas consecuencias de aprendizaje profesional y propio. El proyecto ha permitido aprender la actual API de Audio de Javascript, el manejo de Websockets, conceptos de compresión de audio y conceptos de TLS entre otras cosas. Con ello se complementan los conocimientos adquiridos en los estudios de grado.

El paso por ActualMed no fue mi primera experiencia laboral en el ámbito informático. Comparada con las anteriores, la empresa es de reducido tamaño y con una organización muy jerarquizada. La empresa suministró los medios físicos para realizar el proyecto, pero escaso apoyo tecnológico. Así, la realización del proyecto ha supuesto, ante todo, un gran esfuerzo personal, poniendo a prueba la capacidad de buscar soluciones a problemas técnicos, tanto los especificados como algunos inesperados.

Bibliografía

[1] Wikipedia. Códec de audio.
https://es.wikipedia.org/wiki/C%C3%B3dec_de_audio
[Consulta: marzo de 2017]

[2] Wikipedia. Speex.
https://es.wikipedia.org/wiki/Speex
[Consulta: marzo de 2017]

[3] Wikipedia. Adobe Flash.
https://en.wikipedia.org/wiki/Adobe_Flash
[Consulta: marzo de 2017]

[4] Wikipedia. Adobe Flash Player.
https://en.wikipedia.org/wiki/Adobe_Flash_Player
[Consulta: marzo de 2017].

[5] Wikipedia. Internet Explorer.
https://es.wikipedia.org/wiki/Internet_Explorer
[Consulta: julio de 2017].

[6] Ecma International. ECMAScript® 2015 Language Specification
http://www.ecma-international.org/ecma-262/6.0/ (en inglés)
[Consulta: julio de 2017].

[7] Mozilla Wiki. Audio API.
https://wiki.mozilla.org/Audio_Data_API (en inglés)
[Consulta: julio de 2017].

[8] W3C Wiki. Web Audio API.
https://www.w3.org/TR/webaudio/ (en inglés)
[Consulta: julio de 2017]

[9] Wikipedia. Codificación digital.
https://es.wikipedia.org/wiki/Codificaci%C3%B3n_digital
[Consulta: marzo de 2017].

[10] MDN. Usando Web Workers.
https://developer.mozilla.org/es/docs/Web/Guide/Performance/Usando_web_workers
[Consulta: junio de 2017]

[11] MDN. WebSockets.
https://developer.mozilla.org/es/docs/WebSockets-840092-dup (en inglés)
[Consulta: marzo de 2017]

[12] Echo Test. WebSockets.
https://www.websocket.org/echo.html (en inglés)
[Consulta: julio de 2017]. [Consulta: marzo de 2017]

[13] MDN. JSON.
https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/JSON
[Consulta: marzo de 2017]

[14] JSON. Introducing JSON.
http://www.json.org/ (en inglés)
[Consulta: marzo de 2017]

[15] redHat. Using Stunnel.
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/sec-Using_stunnel.html (en inglés)
[Consulta: abril de 2017]

[16] PMBOK. Guía de PMBOK quinta edición.
https://www.gob.mx/cms/uploads/attachment/file/79535/PMBOK_5ta_Edicion_Espanol__1_.pdf
[Consulta: marzo de 2017]

[17] TDD. Diseño Ágil con TDD,
http://www.carlosble.com/downloads/disenoAgilConTdd_ebook.pdf
[Consulta: marzo de 2017]

[18] GitHub. Librería Speex
https://github.com/jpemartins/speex.js/
[Consulta: abril de 2017]

[19] Wikipedia. Opus.
https://es.wikipedia.org/wiki/Opus_(c%C3%B3dec)
[Consulta: junio de 2017]