Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo e integración de nuevos servicios en el sistema de reconocimiento de voz de una plataforma de generación de informes médicos radiológicos

Autor:
Marc Amposta Pérez

Supervisor:
Rafael Forcada
Tutora:
María José Aramburu Cabo

Fecha de lectura: 7 de Noviembre de 2016

Curso académico 2015 /2016

Resumen

En esta memoria de Proyecto Final de Grado, se detallan el conjunto de procedimientos y recursos que han sido utilizados para la realización del proyecto de desarrollo e integración de servicios en el actual sistema de reconocimiento de voz de ActualTec Innovación Tecnológica, S.L., la empresa donde se ha realizado el proyecto.

Entre otras funcionalidades, este sistema es capaz de transcribir un dictado de voz en tiempo real y está enfocado a la transcripción informes radiológicos. Para ello, dispone de una serie de servicios, tales como diccionarios, macros y comandos diseñados específicamente para facilitar la actividad de los clientes, en este caso radiólogos. El proyecto consiste en implementar e integrar dichos servicios, así como otras funcionalidades varias, a la plataforma web.

Palabras clave

Aplicación web, servicio Web, reconocimiento de voz, radiología, PHP, HTML, JS, CSS, Laravel.

Keywords

Web application, web service, speech recognition, radiology, PHP, HTML. JS, CSS, Laravel.

Índice general

Capítulo 1. Contexto y motivación del proyecto

ActualTec Innovación Tecnológica, es una empresa de base tecnológica, situada en Castellón de la Plana, dedicada a proporcionar soluciones informáticas para empresas y particulares. Disponen de 2 ramas de negocio: ActualWeb (servicios web) y ActualMed (servicios médicos).

Desde ActualWeb, se ofrecen servicios de Internet: diseño y programación de páginas Web, registro de nombres de dominio, correo electrónico, alojamiento web, posicionamiento en buscadores, tiendas virtuales y copias de seguridad remotas y de almacenamiento on-line. Más de 150 clientes en la provincia de Castellón avalan su trabajo, profesionalidad y experiencia.

Desde ActualMed se ofrecen servicios informáticos a centros radiológicos. Entre dichos servicios se encuentra el sistema Actualpacs que permite alojar, editar y acceder a estudios radiológicos. El sistema de edición de los informes radiológicos implementa el servicio de dictado de voz, suministrado por el API Vocali. Mi proyecto consistirá en ampliar y mejorar dicho servicio, pues sus usuarios han destacado varios aspectos a mejorar, así como varias funcionalidades a añadir.

Para este proyecto nos centraremos en el editor de informes de Actualpacs, el cual permite al usuario dictar, via la aplicación Vocali, un informe en base a un máximo de 2 radiografías, que son visualizadas desde el mismo editor. Seguida la finalización de los informes, Actualpacs permite la descarga en formato PDF del mismo, así como la emisión de la versión definitiva a la base de datos. A continuación, se mostrará una traza de los pasos a seguir para editar un informe, para ello es necesario introducir brevemente los tipos de usuario de la plataforma Actualpacs.

El entorno de trabajo en el área de la medicina acostumbra a ser ajetreado, por ello, cualquier herramienta que permita agilizar la actividad en estos centros es más que bienvenida. Esto constituye la mayor motivación para este proyecto, pues este proyecto tiene como principal objetivo abastecer al sistema con recursos intuitivos que aspiran a ser de utilidad para los clientes.

1.1 Descripción de la aplicación actual

Actualpacs dispone de los siguientes grupos de usuarios, ordenados jerárquicamente de mayor a menor: administrador, radiólogo y auxiliar administrativo. Para acceder a la plataforma es necesario acceder previamente como uno de dichos usuarios. Por pertenecer a uno de estos grupos el usuario hereda los permisos de su respectivo grupo y existen permisos comunes a todos los grupos. Empezaremos citando estos y a continuación los permisos particulares a cada grupo de usuarios. Además, los permisos de cada usuario pueden ser modificados individualmente por el administrador.

Permisos:

Comunes:

- Consultar informes asignados.
- Descargar informe en formato PDF.
- Filtrar informes.
- Generar plantillas.

Radiólogo:

- Informar estudios asignados.

Administrador:

- Informar estudios (asignados y no asignados).
- Consultar informes (asignados y no asignados).
- Asignar estudios.
- Borrar estudios.
- Borrar plantillas.
- Asignar permisos.

Como primera toma de contacto con la aplicación, nos encontramos con la pantalla de inicio de sesión (figura 1.0)

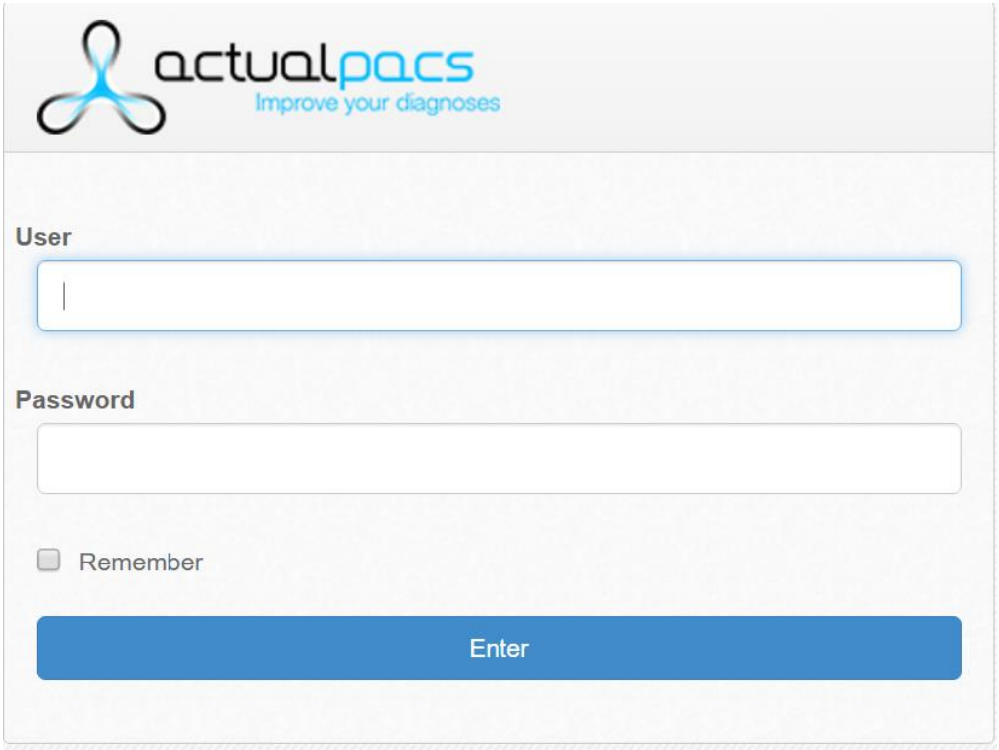


Figura 1.0 Pantalla de inicio de sesión.

Una vez se ha accedido con el nombre de usuario y la contraseña, la página de inicio de la aplicación web Actualpacs presenta la siguiente distribución (fig 1.1).

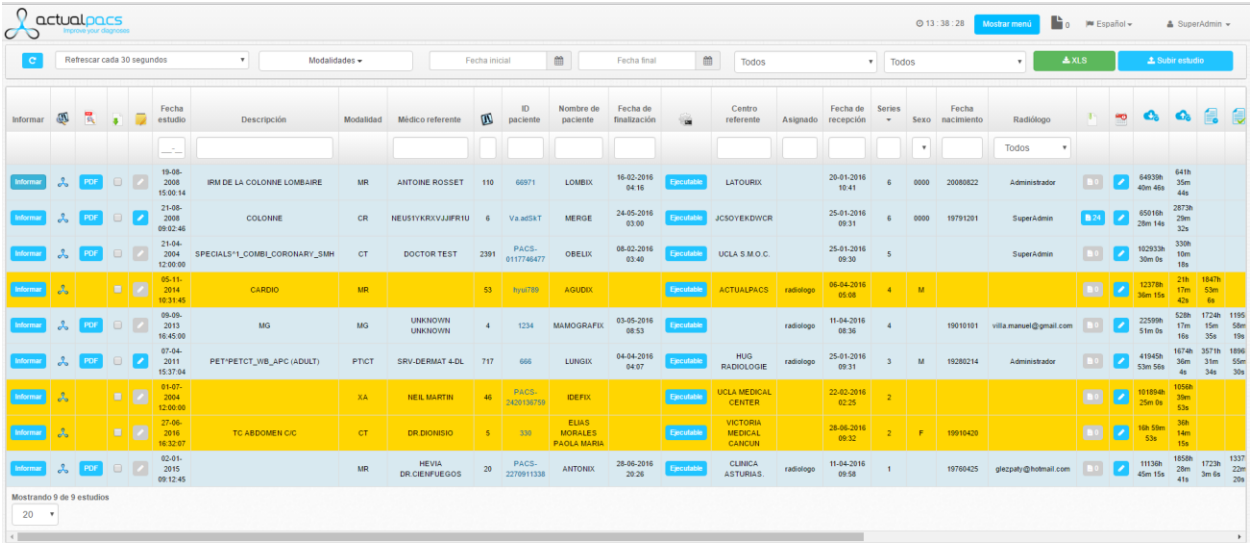


Figura 1.1 Pantalla de inicio, como administrador.

Como se puede apreciar en la figura 1.1, aparecen listados todos los informes asignados al usuario conectado. Pulsando en cualquiera de ellos accedemos al editor del informe, que presenta la interfaz de la figura 1.2.

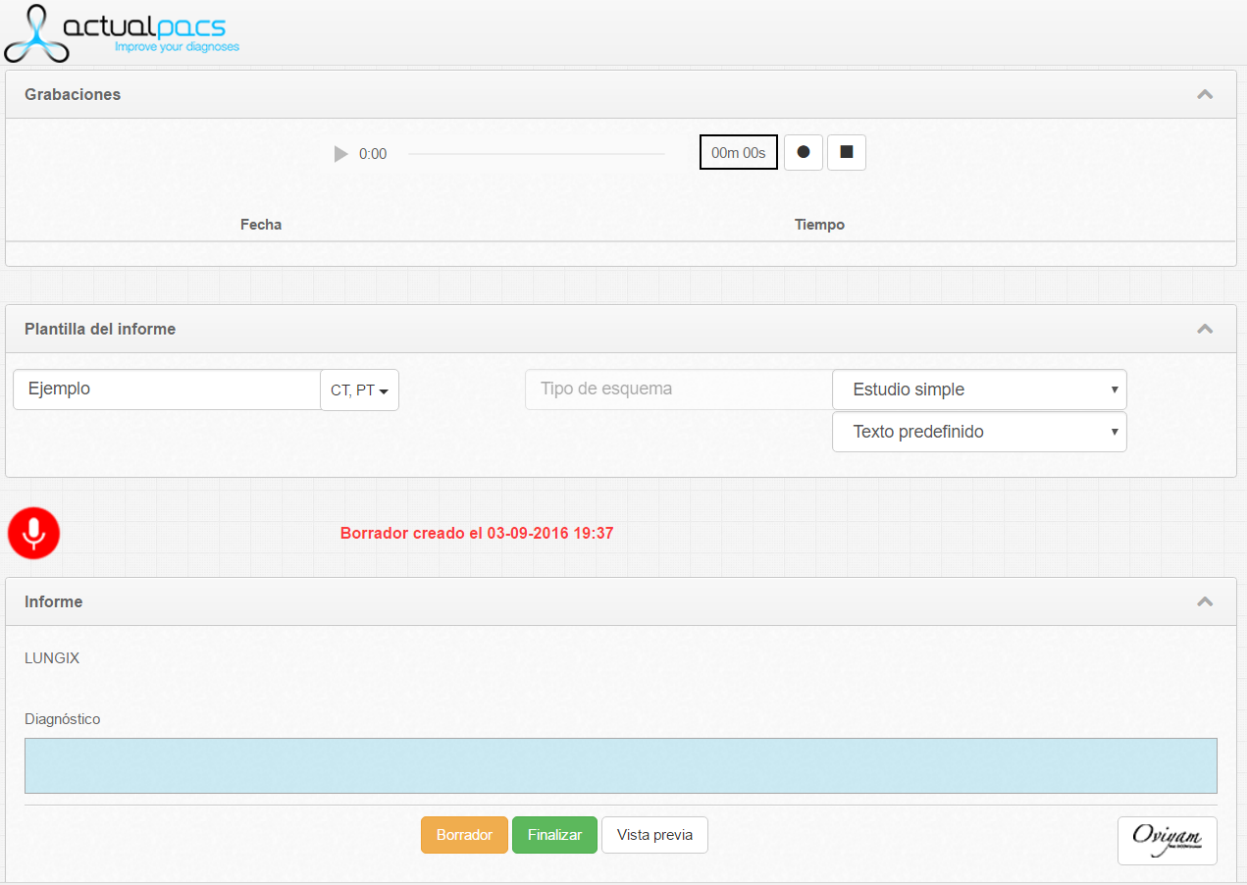


Figura 1.2 Pantalla de edición de informes.

Como se aprecia en la figura 1.2, la pantalla de edición de informes presenta el selector de plantillas en la parte superior. Una plantilla no es más que una configuración predefinida del editor de estudios radiológicos. Una vez cargada una de las plantillas, podemos empezar el dictado pulsando en el icono del micrófono. Asimismo, el sistema permite la visualización de un máximo de dos radiografías, para realizar el informe en base a estas.

Otra funcionalidad de interés para este proyecto es la fase de entrenamiento que consiste en un conjunto de frases que el usuario debe dictar, de esta forma Vocali es capaz de adaptar el dictado al usuario. Como punto de partida, el sistema utiliza el SDK de Vocali para abordar el entrenamiento, dicho sistema presenta un estilo simplista que contrastaba con el estilo del resto de la plataforma web (fig 1.3).

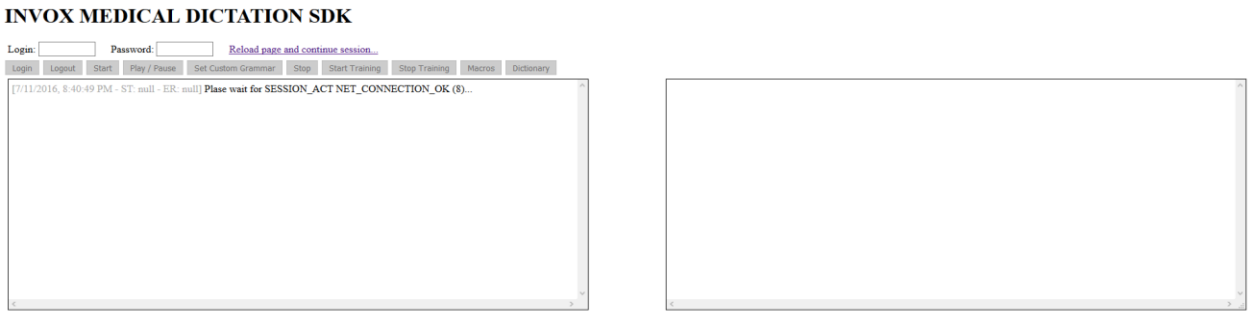


Figura 1.3 SDK Vocali.

En su lugar, como resultado de la ejecución de este proyecto, se utilizará una nueva implementación del entrenamiento del dictado (fig 1.4).

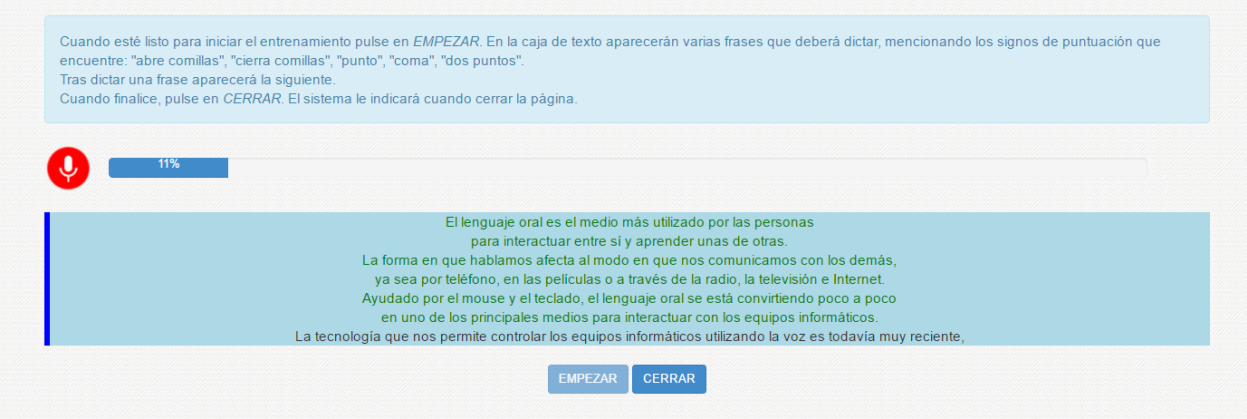


Figura 1.4 Pantalla de entrenamiento.

Para empezar el entrenamiento basta con pulsar en empezar, a continuación, aparecerán una serie de frases con signos de puntuación que el usuario tendrá que dictar, al pronunciar la última frase, se le informará que el entrenamiento ha sido finalizado.

Entre los aspectos a mejorar cabe destacar la fase de entrenamiento. Para ello se implementará el entrenamiento de forma interna, sin usar el SDK de Vocali. Por lo que respecta a las funcionalidades a añadir se ha solicitado el uso de comandos por voz para cargar plantillas y la muestra de resultados parciales (*feedback*) *en la* transcripción del dictado. De todos modos, por iniciativa de la empresa, se añadirán otras funcionalidades no solicitadas por los clientes, como la integración de macros y

diccionarios, así como, otros comandos. En el apartado objetivos entraremos en más detalle cómo abordar cada una de estas tareas.

1.2 Objetivos del proyecto

El principal objetivo de este proyecto es añadir nuevas funcionalidades a la plataforma Actualpacs para agilizar, mejorar o incluir ciertas actividades que los usuarios realizan en esta.

1.2.1 Objetivos específicos del proyecto

A continuación, se definen como una enumeración, los objetivos específicos del proyecto que se corresponden con las funcionalidades a incluir a la plataforma.

1. **Incorporar *feedback* del reconocimiento de voz al usuario.**
   El estado actual de la plataforma plantea el volcado de la transcripción del dictado en un cuadrado de texto. Dicho volcado se efectúa cuando el usuario realiza una pausa en el dictado, lo que presenta problemas como la falta de *feedback* hasta que el usuario realiza una pausa. Para incorporar *feedback*, el sistema extraerá transcripciones de reconocimientos parciales y los mostrará en el cuadrado de texto, al mismo tiempo que se añaden o corrigen las transcripciones tal y como se van reconociendo. De este modo, el usuario será capaz de corroborar que el dictado efectivamente se está transcribiendo correctamente en tiempo real. Los principales contratiempos a la hora de realizar esta tarea vienen dados por la necesidad de que sea compatible con los principales navegadores en el mercado, pues estos siguen políticas muy dispares para aspectos clave como el tratamiento de cursores o la maquetación de los estilos.

2. **Integrar la fase de entrenamiento al sistema.**
   Actualmente el sistema carece de entrenamiento interno, en cambio utiliza el SDK de Vocali. Esta mejora consistirá en crear un estilo web acorde al resto de la plataforma en el que implementar los procedimientos del entrenamiento, tal y como se ha mostrado en la figura 1.4. Las sentencias a dictar en el entrenamiento se registran en un mensaje XML.

3. **Integrar el uso de diccionarios.**
   Vocali ofrece un sistema de diccionarios que permite asignar pronunciaciones a palabras que acostumbren a interpretarse erróneamente. Para ello, el sistema generará un menú desplegable customizado sobre la selección de texto, dicho menú dispondrá de las opciones: “Añadir al diccionario”, “Copiar” y “Pegar”. Una vez el usuario pulse en “Añadir al diccionario” la palabra se registrará en el diccionario.

4. **Integrar el uso de macros.**

Vocali ofrece un sistema de macros que permite reemplazar transcripciones de dictados por textos, ambos predefinidos. Para acceder a esta funcionalidad se añadirá un formulario con un botón con ambos textos como parámetro, al pulsar el botón se enviará la petición para añadir la macro.

5. **Implementar comandos por voz.**
   Vocali ofrece un sistema de comandos por voz que permite crear comandos simples y personalizados. Para los comandos simples los *triggers* están predefinidos. Los *triggers* son lo que el usuario tiene que dictar para activar el comando. Por el contrario, para los comandos personalizados, usaremos un sistema de clases que siguen el patrón de diseño decorador para generarlos. El patrón de diseño decorador responde a la necesidad de añadir dinámicamente funcionalidad a un Objeto. Esto nos permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera. Entre los comandos simples que se añadirán cabe destacar: Seleccionar “texto”, mover a “texto”, mover detrás de “texto” y mayúsculas “texto”. Por otro lado, como comando personalizado se implementará cargar plantilla.

6. **Actualizar a la nueva versión de Vocali.**
   Actualmente el sistema está haciendo uso de una versión desactualizada de Vocali que en breve quedará obsoleta y se cesará su mantenimiento. Para ello, habrá que reemplazar los archivos con los de la nueva versión, a la par que se adapta el código para que funcione con la nueva versión.

7. **Formación equipo Actualmed.**
   Dado que los responsables principales de la implementación de Vocali han dejado la empresa, actualmente no hay nadie que esté familiarizado con Vocali. Mi tarea consistirá en introducir a los empleados involucrados en el desarrollo de Actualpacs, en la medida que se me indique, cómo tratar con la aplicación Vocali.

Capítulo 2. Descripción del proyecto.

2.1 Recursos software

En este apartado Introducimos el software que ha sido utilizado para elaborar el proyecto. Todos los recursos han sido impuestos por la empresa para que sean los mismos que los usado en el proyecto Actualpacs.

- **PHP:** PHP es un lenguaje de lado servidor diseñado para el desarrollo web pero también usado como lenguaje de propósito general. Fue creado originalmente por Rasmus Lerdorf en 1995. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP.

- **Laravel**: Para el desarrollo de la plataforma se usará el entorno de desarrollo Laravel que es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti". Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC.

- **JavaScript:** (abreviado comúnmente JS). Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. En este proyecto, se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

- **Airbnb**: Para mantener una concordancia con los distintos proyectos de la empresa implementados en parte o en totalidad con JS, se utiliza el standard de estilo. **Airbnb.** Entre otras peculiaridades puramente estilistas, este estándar anima a utilizar prácticas que minimicen la aparición de errores.

- **HTML**: El HyperText Markup Language, hace referencia al lenguaje de marcado para la elaboración de páginas web. A grandes rasgos consiste en un sistema de etiquetas que permite definir distintos elementos en una misma página web, bien sean imágenes, audio, video, secciones etc.

- **CSS**: Hoja de estilo en cascada o **CSS** (siglas en inglés de *cascading style sheets*) es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML.

- **Git**: es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

2.2 Arquitectura y funcionalidades del sistema

En este apartado se hará un inciso en los aspectos de metodología y tecnologías utilizadas en este proyecto. Por otro lado, se presentarán los diferentes componentes del proyecto y cómo estos interactúan entre sí.

Para ilustrar este apartado nos apoyaremos en el diagrama del sistema que se muestra en la figura 2.1.

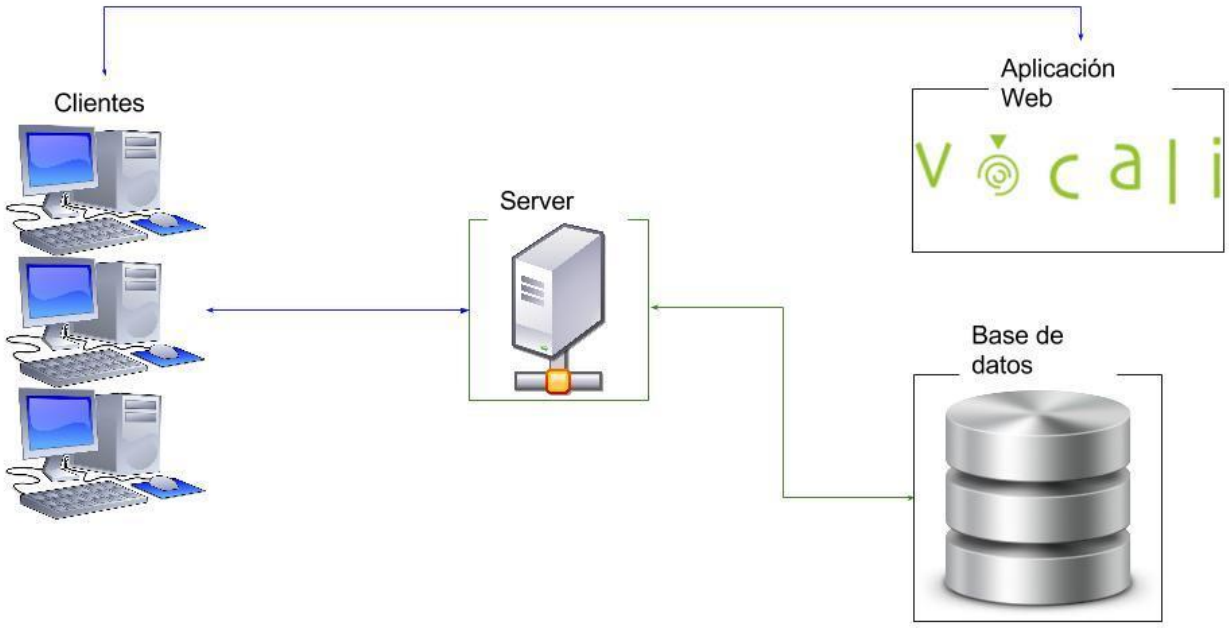


Figura 2.1: Diagrama extendido del funcionamiento del sistema.

En este proyecto podemos destacar dos principales líneas de trabajo:

- Vocali sigue el paradigma cliente-servidor. Nosotros tenemos acceso al cliente, por ello debemos establecer todas las comunicaciones pertinentes con el servidor de Vocali en la parte cliente. Para ello haremos usos de *sockets*, todos los parámetros de conexión son establecidos y detallados en el manual proporcionado por Vocali. Una vez completado este apartado, podemos realizar peticiones vía *sockets*. En esta línea de trabajo haremos uso del lenguaje de programación JS siguiendo el standard Airbnb, así como XML como formato de los mensajes.
- Para el Desarrollo tanto funcional como visual del apartado WEB de Actualpacs, se sigue el sistema modelo-vista-controlador. En cuanto a las herramientas para este apartado, se usa el *framework* Laravel, el entorno de desarrollo XAMPP, el modelo de vistas Blade, el lenguaje HTML, estilos CSS, Bootstrap y el lenguaje PHP utilizando el *framework* Symphony. Al igual que Vocali, Actualpacs sigue el paradigma cliente-servidor. La parte cliente se ejecuta en el mismo navegador y envía peticiones a los servidores de Actualpacs y Vocali, estos atienden dichas peticiones y actúan en consecuencia. Los navegadores soportados son: Firefox, Chrome, Safari y IE. Aunque no se descarta, no se asegura el correcto funcionamiento en otros navegadores.

Las tecnologías necesarias para este proyecto no distan de las básicas para cualquier proyecto estándar, un equipo informático, micro y servidores cubren todas las necesidades.

A continuación, se explica el funcionamiento del sistema. Nuestro cliente tiene acceso al cliente de Vocali y desde este se realizan peticiones directamente al servidor de Vocali (GET y POST habitualmente). Dichas peticiones se envían vía *socket*, Vocali responde con el resultado de dicha petición. Una vez el cliente recibe el resultado, este puede mostrar el resultado como crea conveniente. En el caso de la transcripción, se vuelca el resultado en un cuadrado de texto que iremos actualizando tal y como se vayan reconociendo nuevas transcripciones.

Como veremos más adelante, en ciertas tareas es necesario acceder a estructuras de datos alojadas en la base de datos de Actualpacs. En dichas situaciones el cliente solicita las estructuras pertinentes al servidor de Actualpacs, a su vez el servidor recogerá la información solicitada y la enviará al cliente en forma de respuesta.

2.2.1. Cliente

Para el cliente se ha decidido dar soporte a los navegadores Firefox, Chrome, Internet explorer y Safari. Todas las comunicaciones con los distintos componentes de Vocali, así como, las funcionalidades a añadir, se implementarán en JavaScript. Por lo que respecta al apartado visual se utilizan las herramientas básicas HTML y CSS. Además, se usará Bootstrap pues ofrece gran variedad de estilos que mantienen un comportamiento uniforme sin importar el dispositivo.

2.2.2 Servidor

En este proyecto realizaremos peticiones a servidores de Vocali y Actualpacs. Dado que no tenemos ningún tipo de control sobre el server de Vocali, no debemos preocuparnos de este. Por lo que respecta al apartado servidor de Actualpacs, desde este se enrutan todas las peticiones a las distintas URLs de la plataforma web a la par que se administran todas las consultas a la base de datos.

2.2.3 Base de datos.

Actualpacs hace uso de una base de datos relacional SQL. En nuestro caso nos centraremos en una pequeña abstracción de la base de datos de Actualpacs, pues para el caso que nos concierne no necesitamos la totalidad de las tablas sino un subconjunto reducido. En dicha abstracción haremos uso de dos tablas, usuario y plantilla. 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. Todos estos elementos están definidos en código HTML en la columna configuración de la tabla plantilla. La relación entre ambas es de muchos a muchos, es decir, un usuario puede disponer de varias plantillas y una plantilla puede ser usada por varios usuarios. Para abordar esta relación mediante los estándares de normalización, se generará una tabla auxiliar con llave doble siendo las claves id_usuario e id_plantilla, de esta forma se mantiene una relación usuario - plantilla.

Figura 2.2 Diagrama UML base de datos simplificado

2.2.4 Vocali.

Vocali es la aplicación utilizada para transcribir texto, sin embargo, como veremos más adelante, ofrece muchísimas más posibilidades. Actualpacs dispone del cliente de Vocali y a través de él realiza todas las operaciones. Las peticiones se harán directamente desde el cliente y vía *socket*, asimismo se harán uso de archivos XML en ciertos mensajes.

Aunque cuando entremos en materia de implementación incidiremos más en este aspecto, cabe destacar a groso modo como funciona Vocali. Vocali presenta una serie de ámbitos y desde cada uno puedes realizar unas tareas u otras. En cualquier momento puedes cambiar de ámbito, sin embargo, debes tener en cuenta que no se esté llevando a cabo ningún proceso.

En la tabla de la figura 2.3 se muestra en la primera columna los ámbitos y en la primera fila los eventos que puede disparar cada ámbito.

	Dictation Event	Command Event	Keyboard Event	Instance Selection Event	NewCorrection Event	EndCorrection Event	Phonetic Event
NONE							
CUSTOM							
PAUSED							
RUNNING							
INSTANCE_ SELECTION							
CORRECTION							
PHONETIC_ TRANSCRIPTION							
SPELLING							

Figura 2.3 Ámbitos y eventos asociados

En este proyecto se usarán los siguientes ámbitos: paused, running y phonetic transcription.

A continuación, se introducen los eventos usados en el proyecto:

- Dictation Event: evento que se dispara en el ámbito *running* cuando quiera que se transcribe texto, el evento contiene el texto transcrito.
- Command Event: evento que se dispara en el ámbito *running* cuando se reconoce un comando configurado, el evento contiene el identificador del comando disparado.
- Phonetic Event: evento que se dispara que se dispara en el ámbito phonetic transcription cuando se reconoce texto, el evento contiene la transcripción fonética del texto reconocido.

Capítulo 3. Planificación

Este capítulo pretende exponer la distribución de recursos que se realizó en la fase inicial del proyecto. Su objetivo es especificar los costes temporales y los recursos materiales y humanos necesarios para abordar las tareas del proyecto.

3.1 Metodología

Para este proyecto se seguirá la metodología ágil SCRUM. Las metodologías ágiles no están diseñadas como un conjunto de mandamientos que hay que seguir al pie de la letra, sino que, la utilidad de dichas metodologías reside en la capacidad de adaptación de las mismas al proyecto y a las circunstancias concretas.

SCRUM es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.
- Reuniones diarias para la planificación del *sprint*.
- El *Sprint* es el período en el cual se lleva a cabo el trabajo en sí. Es recomendado que la duración de los *sprints* sea constante y definida por el equipo con base en su propia experiencia. Se puede comenzar con una duración de *sprint* en particular (2 o 3 semanas) e ir ajustándolo con base en el ritmo del equipo, aunque sin relajarlo demasiado. Al final de cada *sprint*, el equipo deberá presentar los avances logrados, y el resultado obtenido es un producto que, potencialmente, se puede entregar al cliente.
- Así mismo, se recomienda no agregar objetivos al *sprint* a menos que su falta amenace al éxito del proyecto. La constancia permite la concentración y mejora la productividad del equipo de trabajo.

Dado que soy el único responsable del proyecto y no se requiere coordinación con el resto de proyectos de la empresa, no se realizarán reuniones diarias para planificar el sprint como aconseja la metodología SCRUM, sino que en su lugar se propondrán dichas reuniones cuando haya avances considerables en el proyecto.

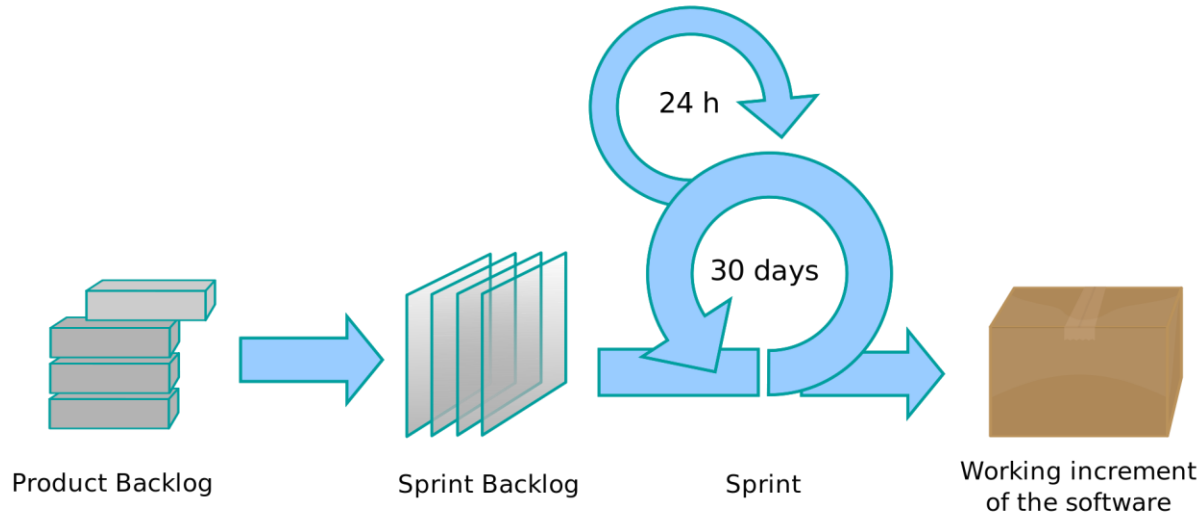


Figura 3.1 Esquema de SCRUM

3.2 Control de versiones

Como se indicó en el apartado 'Recursos Software' para el control de versiones se usará GIT. La metodología que se seguirá al largo del proyecto respecto al uso de GIT se centrará en el uso de Ramas. Una rama representa una línea de desarrollo en particular dentro del marco del proyecto en cuestión. En el proyecto Actualpacs existen 2 ramas principales *master* y *develop*. *Master* representa la línea de trabajo de mayor jerarquía, esta línea de desarrollo contiene la versión actual del proyecto y solo se actualizará para subirla a una nueva versión. *Develop* es la rama a la que todos los desarrolladores involucrados en el proyecto aplican su desarrollo. Cuando esta rama contiene todas las características necesarias que conforman la nueva versión, es fusionada con *master*, de este modo master pasaría a formar la versión actual. Para asegurar que todos los desarrolladores tienen en cuenta posibles cambios en el código realizados en cualquier iteración del desarrollo, se crearán las ramas particulares a partir de *develop*, de este modo se arrastran todos los cambios realizados por los distintos miembros del equipo. Para asegurar que dos miembros del equipo no machacan código mutuamente, a la hora de fusionar, GIT detecta si fragmentos de código han cambiado en diferentes ramas desde que se originaron dichas ramas. En ese caso, ambos miembros del equipo deberán poner su trabajo en común, de todos modos, esto no debería suceder si se planifica correctamente el Sprint.

3.3 Tareas del proyecto

Previamente en el apartado objetivos hemos dado una introducción a los objetivos específicos del proyecto, considerando cada uno de estos objetivos específicos como una tarea independiente. En cambio, en este apartado entraremos en más detalle citando peculiaridades, herramientas y decisiones tomadas para llevarlas a cabo, pero sin entrar en materia de implementación, pues se reserva para el correspondiente apartado.

1. **Incorporar *feedback* del reconocimiento de voz al usuario.**

   En esta tarea se utilizarán los ámbitos *Running* y *Paused* (figura 2.3), para iniciar y pausar el dictado. Se ha decidido mostrar el texto parcial en un color a elección del usuario, desafortunadamente la transcripción se realiza sobre un cuadrado de texto tipo *textarea* y éste solo soporta texto plano. Como alternativa se ha decidido abordar el problema apoyándonos en un *div content editable*. Esto no es más que una sección HTML editable, gracias a esto podemos utilizar distintos marcados de texto que provee HTML en secciones específicas del texto. Para definir la sección sobre la que actúa el marcado de texto, envolvemos el texto en cuestión en etiquetas HTML. Por defecto tenemos acceso a la etiqueta HTML *span* que resulta adaptarse a nuestras necesidades, pues está diseñado para agrupar secciones de texto distintas en un mismo texto.
   Las incidencias más habituales a la hora de realizar esta tarea vienen protagonizadas por la implementación del cursor de texto en los distintos navegadores, pues genera varios conflictos y dependencias. Como ejemplo, Safari e Internet explorer reinician la posición de cursor si se oculta la visibilidad. Por ello se ha tenido que corregir dicho caso crítico. En todo momento se ha intentado no sacrificar la legibilidad y cohesión del código.

2. **Integrar la fase de entrenamiento al sistema.**

   Esta tarea se ha centrado en el diseño de la interfaz a la par que implementar los procesos de entrenamiento. Para implementar el entrenamiento se capturan eventos, donde cada evento trata una frase del entrenamiento. El último evento contiene un valor especial que indica el fin del entrenamiento. Dichos eventos son escuchados por un *listener* que actúa en consecuencia. La dinámica en cada evento consiste en actualizar la vista, es decir, lo que se muestra al usuario. Los elementos más destacables en la pantalla de entrenamiento son la sección donde se muestran las imágenes tratadas hasta el momento y la barra de progreso (figura 1.4).
   Para empezar el entrenamiento el usuario debe pulsar en empezar, de forma análoga, el usuario debe pulsar en terminar para terminar el dictado.

3. **Integrar el uso de diccionarios.**

   Para esta tarea haremos un uso del ámbito de dictado *phonentic transcripción* (figura 2.3). Este ámbito genera un evento a partir del cual se permite extraer la transcripción fonética de una palabra a partir de la pronunciación del usuario. Dicho valor será usado como parámetro a la hora de

generar la palabra a añadir al diccionario. La construcción de la palabra se realizará en base a una clase suministrada por Vocali llamada Word y que tiene como atributos la transcripción de la pronunciación de la palabra y la palabra en términos gramaticales. Por ejemplo, para el caso de la palabra "hola" la clase contendría como parámetros: "ola" y "hola". Una vez generada la palabra se envía en una petición vía *socket* para que se incluya en el diccionario alojado en los servidores de Vocali. Cabe puntualizar que se puede vincular la palabra exclusivamente al usuario que la generó o a la totalidad de usuarios.
Para añadir la palabra al diccionario el usuario debe seleccionar la palabra y abrir el menú contextual, pulsando en añadir al diccionario.

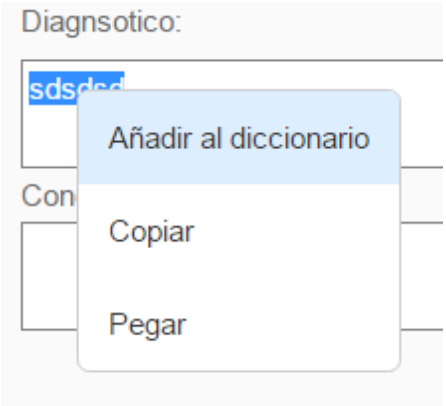


Figura 3.2 Menú contextual diccionario.

Para generar dicho menú se evita la aparición del menú por defecto y se reemplaza por este, sin embargo, este bloqueo sólo se producirá en los campos del dictado.

4. **Integrar el uso de macros.**

Para acceder a la gestión de diccionarios, se incorpora un formulario con dos campos de texto como parámetros: el texto y el reemplazo de la macro. La forma de generar la macro comparte muchas similitudes con la forma de crear el diccionario. Para generar la macro haremos uso de una clase suministrada por Vocali que tiene como atributos el texto y el reemplazo de la macro. Una vez generada la macro se le enviará una petición para incluirla en el conjunto de macros alojado en los servidores de Vocali.

5. **Implementar comandos por voz.**

Como se introdujo en el capítulo uno, Vocali sigue un sistema de clases para generar comandos personalizados, este sistema se denomina gramática de comandos.

Una gramática de comandos queda definida internamente por un objeto CommandSpec que, en sí mismo, es un árbol de objetos CommandSpec según el patrón de diseño Composite.
En el siguiente diagrama de clases UML se ven los tipos de nodos que puede contener el árbol.

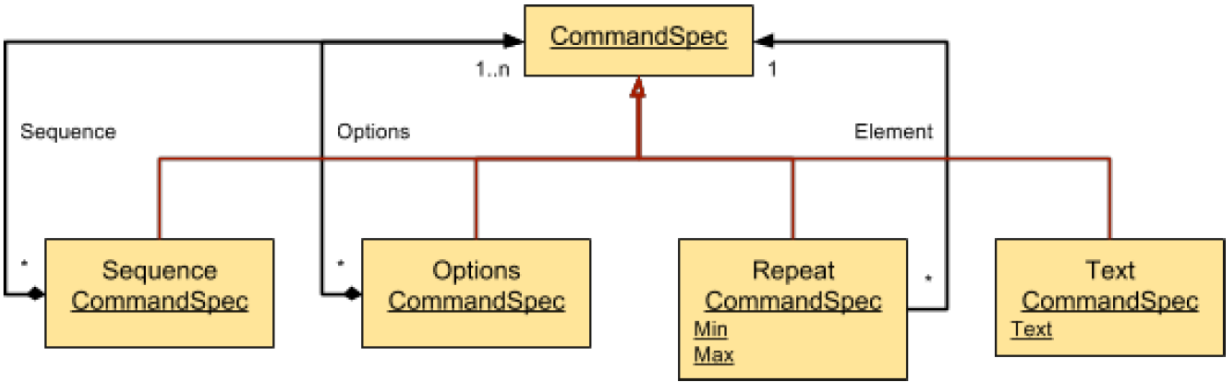


Figura 3.3 Arbol sintáctico Command spec.

Estas composiciones de objetos en forma de árbol se pueden encontrar en las definiciones sintácticas de prácticamente todos los lenguajes estructurados. Es muy habitual que se representen con diagramas sintácticos y documentos BNF. En nuestro caso particular haremos uso de Text CommandSpec y Options CommandSpec. El comando consistirá en una parte invariable, es decir *Text* CommandSpec con el contenido "Plantilla", seguidamente se presenta un option command Spec englobando todas las plantillas disponibles en la base de datos. Por lo tanto, la invocación del comando presentaría el siguiente formato: "Plantilla [Tipo de Plantilla]".

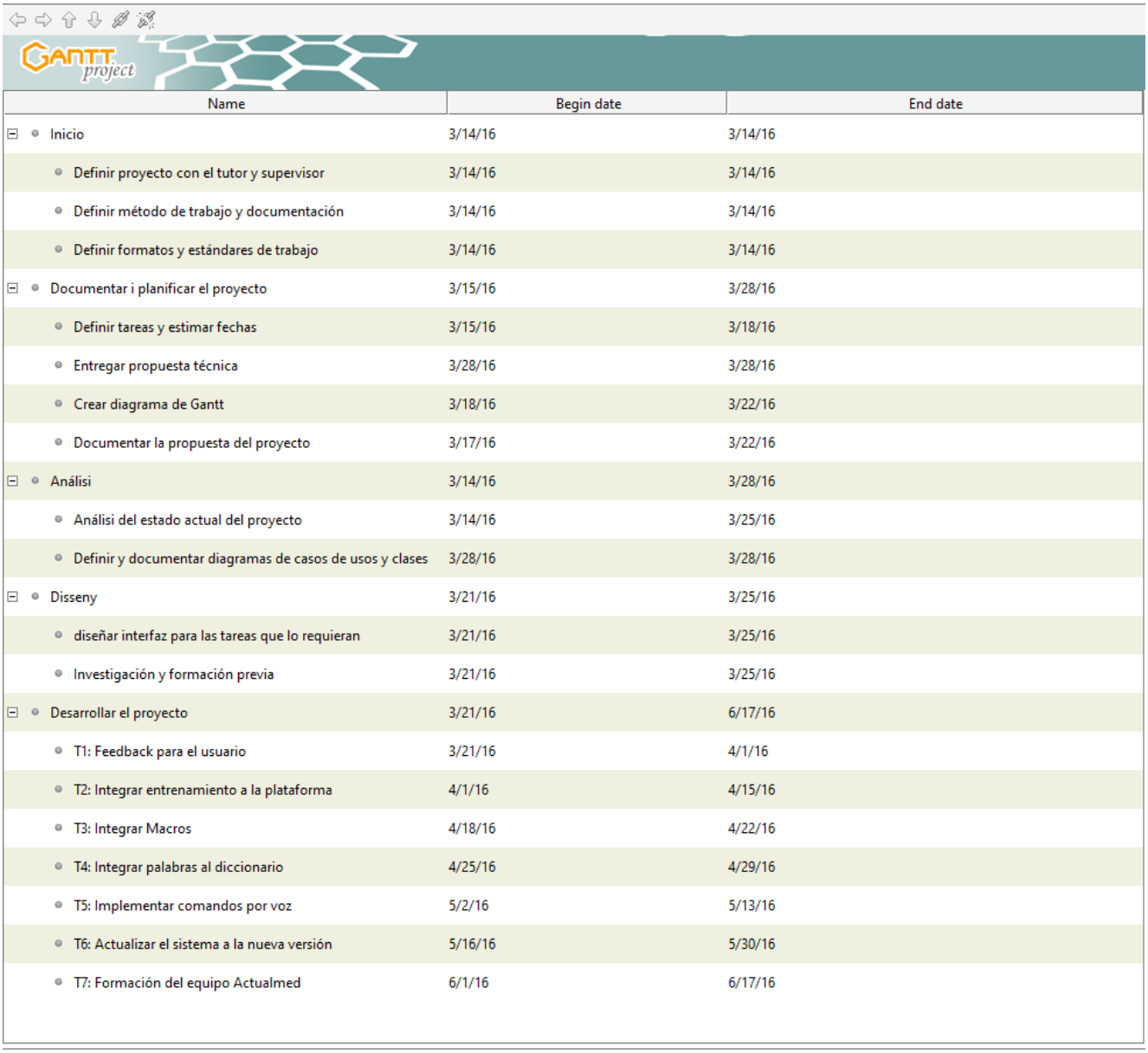
6. **Actualizar a la nueva versión de Vocali.**

   Esta tarea es la más mecánica, la dificultad radica en identificar en el código de Vocali, todos aquellos métodos obsoletos y actualizarlos siguiendo las indicaciones del manual de Vocali.

7. **Formación equipo Actualmed.**

   Esta tarea se concreta en una presentación, repasando todas aquellas características de Vocali que se consideren de interés común al resto del equipo.

3.4 Estimación temporal



Name	Begin date	End date
Inicio	3/14/16	3/14/16
Definir proyecto con el tutor y supervisor	3/14/16	3/14/16
Definir método de trabajo y documentación	3/14/16	3/14/16
Definir formatos y estándares de trabajo	3/14/16	3/14/16
Documentar i planificar el proyecto	3/15/16	3/28/16
Definir tareas y estimar fechas	3/15/16	3/18/16
Entregar propuesta técnica	3/28/16	3/28/16
Crear diagrama de Gantt	3/18/16	3/22/16
Documentar la propuesta del proyecto	3/17/16	3/22/16
Análisi	3/14/16	3/28/16
Análisi del estado actual del proyecto	3/14/16	3/25/16
Definir y documentar diagramas de casos de usos y clases	3/28/16	3/28/16
Disseny	3/21/16	3/25/16
diseñar interfaz para las tareas que lo requieran	3/21/16	3/25/16
Investigación y formación previa	3/21/16	3/25/16
Desarrollar el proyecto	3/21/16	6/17/16
T1: Feedback para el usuario	3/21/16	4/1/16
T2: Integrar entrenamiento a la plataforma	4/1/16	4/15/16
T3: Integrar Macros	4/18/16	4/22/16
T4: Integrar palabras al diccionario	4/25/16	4/29/16
T5: Implementar comandos por voz	5/2/16	5/13/16
T6: Actualizar el sistema a la nueva versión	5/16/16	5/30/16
T7: Formación del equipo Actualmed	6/1/16	6/17/16

Figura 3.4 Tabla de tareas

Figura 3.5 Diagrama de Gantt

3.5. Herramienta para seguimiento y control de la gestión temporal del proyecto

En Actualmed se exige el seguimiento preciso y efectivo de la situación de las tareas en relación con el resto del proyecto. Para ello, disponen de una herramienta de gestión de proyectos llamada Redmine.

Redmine es una herramienta para la gestión de proyectos que incluye un sistema de seguimiento de incidentes y errores. Otras herramientas que incluye son calendario de actividades, diagramas de Gantt para la representación visual de la línea del tiempo de los proyectos, wiki, foro, visor del repositorio de control de versiones, RSS, control de flujo de trabajo basado en roles e integración con correo electrónico, entre otras opciones.

Está escrito usando el *framework* Ruby on Rails. Es software libre y de código abierto, disponible bajo la Licencia Pública General de GNU v2.

El diseño de Redmine está significativamente influenciado por Trac, otra herramienta con características similares.

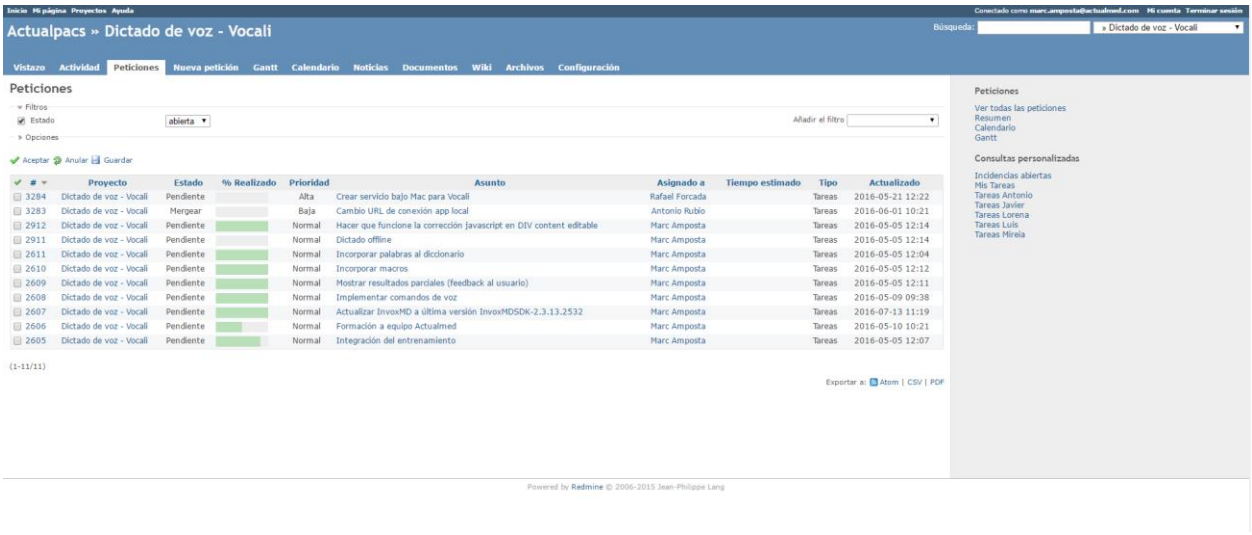


Figura 3.6 Pantalla inicio.

Como podemos apreciar en la figura 3.6, aparecen listadas las tareas asignadas al usuario registrado. Entre las características más interesantes de Redmine cabe destacar la posibilidad de añadir anotaciones acerca el estado del proyecto que pueden ser visualizadas por todos los compañeros involucrados en el proyecto, además de establecer las fechas de inicio y fin estimadas para la tarea. También permite obtener de forma gráfica si ha habido desviaciones en la planificación. En el caso de este proyecto las desviaciones no han sido significativas por lo que se han obviado.

Otra característica a puntualizar es el nombre de las tareas. Como podemos observar, en la primera columna aparece el código único referente a la respectiva tarea. Dicho código será usado para identificar la rama de su tarea en el control de versión GIT de la siguiente forma: feature-[Código tarea].

Capítulo 4. Análisis

En esta apartado se describe todo el estudio referente al análisis de la aplicación. Partiendo de los requisitos del sistema, se describen todas las partes que hacen que la aplicación funcione correctamente.

4.1. Requisitos del sistema

Dadas las características del proyecto podemos categorizar los requisitos dentro de tres tipos:

- **Requisitos de datos**: expresan una necesidad referente a la información que debe almacenar el sistema.
- **Requisitos funcionales**: expresan una necesidad que se cubre añadiendo una funcionalidad al software.
- **Requisitos no funcionales**: un requisito que especifica criterios que pueden usarse para definir la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales. Por tanto, se refieren a todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento. En nuestro caso particular sólo el diseño de la interfaz se ajusta a estas características.

4.1.1. Requisitos de datos

A continuación, Indicaremos los requisitos de datos para las tareas que lo requieren.

Dictado

El sistema tiene que devolver una cadena específica por cada *stream* de audio reconocido en el dictado. Dicha cadena está contenida en el evento del dictado activado por parte de Vocali. No obstante, hay diferentes tipos de eventos, y el sistema debe contener la información para poder activar el evento correcto. Internamente serán almacenados como un objeto enumeración. Dichos tipos son Reconocimiento aceptado, Reconocimiento parcial, Reconocimiento denegado y Reconocimiento desconocido.

Tipo evento (Requisito de datos)	*Activación*	*Resultado*
Aceptado	Pausa en el dictado	Cadena correspondiente a los reconocimientos efectuados hasta el momento.
Parcial	Emisión ininterrumpida del dictado.	Cadena de reconocimiento parcial (sujeto a modificaciones).

Denegado	Emisión ininterrumpida del dictado (este evento cortará la emisión).	Cadena que se ha reconocido, pero no ha superado el umbral para considerarse correcta.
Desconocido	Emisión ininterrumpida del dictado.	El *stream* de audio se considera ininteligible por lo tanto devuelve cadena vacía.

Tabla 4.1 Tipos de eventos de dictado.

Entrenamiento

Como se ha mencionado en anteriores apartados, se almacenarán las frases que se dictarán en la fase de entrenamiento en un fichero XML, que representa la siguiente distribución de etiquetas.

Entrenamiento

Etiqueta	**Descripción**
Cabecera	Indica la versión de mensaje XML.
Frases entrenamiento	Inicia y cierra la secuencia de frases
Frase	Inicia y cierra cada una de las frases del entrenamiento.

Tabla 4.2 Estructura de frases entrenamiento.

A continuación, se muestra un ejemplo del fichero XML.

```
<?xml version="1.0" encoding="utf-8" ?>
<FrasesEntrenamiento>
<Frase>Frase1</Frase>
<Frase>Frase2</Frase>
</FrasesEntrenamiento>
```

Diccionarios/Macros

Las peticiones a enviar via *socket* representa la siguiente estructura:

Macros:

```
imdSession.Macros.addMacro(new invoxmd.Macro($('#macro_name').val(),
$('#macro_replacement').val(), false, invoxmd.macroType.TEXT));
```

Diccionarios:

```
imdSession.Dictionary.addWord(new invoxmd.Word(dicWord,[new
invoxmd.Pronunciation(msg.Transcription,0)], false));
```

Como se indicó en el apartado 3.3 puntos 3 y 4 y como se puede apreciar en la estructura de las peticiones, los diccionarios y macros hacen uso de ciertas clases para representar el contenido a añadir en las colecciones. El sistema debe ser capaz de generar dichas instancias de clases.

A continuación, se detalla el contenido de dichas clases y cómo se almacena en el sistema.

- Diccionarios.
  La petición para añadir una palabra al diccionario precisa de la palabra en cuestión. Además, cada palabra contiene una pronunciación como atributo que a su vez contiene una serie de atributos. La estructura de palabras y pronunciaciones es la representada en las tablas 4.3 y 4.4 respectivamente:

Palabra

Dato	**Descripción**
Texto	Cadena que representa la palabra a añadir.
Pronunciación	Pronunciación correspondiente a la palabra a añadir.
Compartido	Booleano que indica si se desea que se comporta entre los usuarios o no.

Tabla 4.3 Estructura de “Palabra”.

Pronunciación

Dato	**Descripción**
Símbolo	Cadena que representa la transcripción fonética de la palabra a añadir.
POS	Modalidad del discurso, esta funcionalidad no se utiliza por lo que el valor siempre será NULL.

Tabla 4.4 Estructura de “Pronunciación”.

- Macros.
  La petición para añadir una macro a la colección precisa de la macro en cuestión, la estructura de las macros es la representada en la tabla 4.5:

Macro

Dato	Descripción
Nombre	Cadena sobre la que se aplica el reemplazo de la macro
Reemplazo	Cadena que se reemplazará por Nombre
Compartida	Booleano que indica si desea que se comporta entre los usuarios o no.
Subtipo	Indica el tipo de macro, en nuestro caso utilizaremos solo el subtipo texto, no obstante, los subtipos son: TEXT, KEYBOARD, COMPOSITE, PAUSE, MOUSE, UNKNOWN.

Tabla 4.5 Estructura de "Macro".

Comandos

Como vimos en el capítulo dos, los comandos se representan mediante una estructura de clases siguiendo el patrón de diseño *composite* (Fig 2.5). En nuestro caso particular haremos uso de las subclases *Text* y *Option*, que forman parte de la secuencia del comando. Asimismo, dicha secuencia será un *Array* de *CommandSPec* (Clase padre), de este modo se permite que formen parte de ella todas las clases hijo.

Text

Dato	Descripción
Text	Cadena que dispara el comando

Tabla 4.6 Estructura de "CommandSpec Text"

Option

Dato	Descripción
Option	*Array* de objetos "Text" que representan las opciones del comando.

Tabla 4.7 Estructura de "CommandSpec Option"

4.1.2 Requisitos funcionales

Casos de uso

El propósito de esta sección es mostrar cómo el usuario interactúa con el sistema. Nuestro caso particular no presenta gran complejidad pues no interviene una jerarquía de usuarios como tal para hacer uso del dictado de voz. Por ello desempeñamos un mayor énfasis en las funcionalidades más que en los usuarios que la desempeñan. A continuación, se muestra el diagrama de casos de uso del sistema.

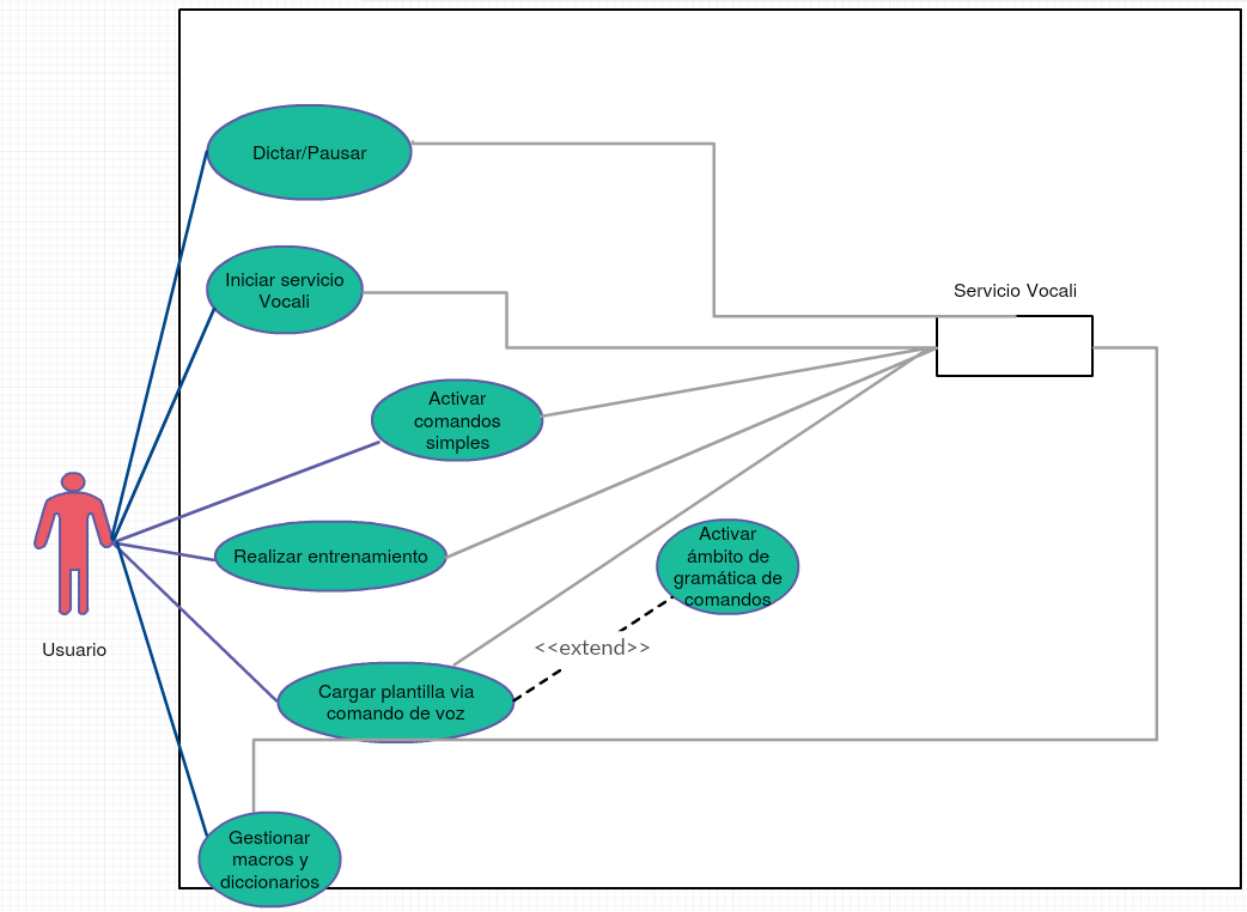


Figura 4.1 Diagrama de casos de uso.

Dictar/Pausar

ID: UC01
Nombre: Dictar/Pausar
Autor: Marc Amposta Pérez

Pasos:

1. En la sección de redacción de informes, el usuario clica en el icono del micrófono para iniciar/pausar.
2. El dictado cambia el estado de pausado a iniciado y viceversa dependiendo del estado actual

Excepciones:

Precondiciones: Iniciar sesión en Vocali

Iniciar servicio Vocali

ID: UC02
Nombre: Iniciar servicio Vocali
Autor: Marc Amposta Pérez

Pasos:

1. Cuando la página HTML esté cargada, el sistema iniciará la sesión de Vocali automáticamente

Excepciones:

- En caso de que ya exista una sesión del usuario Vocali abierta, el sistema esperará hasta que sólo haya una única solicitud de la sesión.

Precondiciones: Documento HTML debe estar listo

Activar comandos simples

ID: UC03
Nombre: Activar comandos simples
Autor: Marc Amposta Pérez

Pasos:

1. En la sección de redacción de informes, el usuario pronuncia una de las siguientes graníticas:
   Borrar eso
   Mayúsculas eso
   Ir al final del documento
   Ir detrás de xxx
   Ir a xxx
   seleccionar xxx
2. El sistema ejecuta el comando indicado.

Excepciones: Si el sistema no lo reconoce como comando se transcribe el texto como si se tratara de un dictado. En dicho caso el usuario será el responsable de eliminar dicha transcripción si así lo desea.

Precondiciones: Iniciar sesión en Vocali

Realizar entrenamiento

ID: UC04
Nombre: Activar comandos simples
Autor: Marc Amposta Pérez

Pasos:

1. En la sección de Entrenamiento, el usuario inicia o reanuda el entrenamiento clicando en el botón empezar o en el icono del micrófono de forma indiferente.
2. Para avanzar en el entrenamiento, el usuario debe pronunciar la frase i-esima de la iteración.
3. Al finalizar el entrenamiento se enviará el resultado a los servidores de Vocali.

Excepciones: En caso de que se obtenga una excepción por parte del servidor de Vocali, el sistema bloqueará cualquier tipo de interacción del usuario con el sistema.
Si el usuario cierra la ventana sin terminar el entrenamiento, el sistema no envía ningún resultado a los servidores de Vocali, por lo tanto, es como si nunca se hubiera realizado a efectos prácticos.

Precondiciones: Iniciar sesión en Vocali

Comentarios: Durante el transcurso del entrenamiento el cliente puede pausar el dictado clicando el botón detener, en ese caso podrá reanudar el dictado siguiendo nuevamente las indicaciones del Paso 1.

Activar comando custom

ID: UC05
Nombre: Activar comandos simples
Autor: Marc Amposta Pérez

Pasos:

1. En la sección de redacción de informes, el usuario pronuncia la siguiente gramática: Menú > Plantilla XX.
2. El sistema carga la plantilla seleccionada.
3. El sistema notifica al usuario que la operación se ha completado con éxito.

Excepciones: En caso de ausencia de plantillas el sistema mostrará un mensaje notificando al usuario.

Precondiciones: Iniciar sesión en Vocali

Gestionar macros y diccionarios

ID: UC06
Nombre: Activar comandos simples
Autor: Marc Amposta Pérez

Pasos:
Diccionario

1. En la sección de redacción de informes, el usuario selecciona una palabra con el cursor, acto seguido aparecerá un menú desplegable con la opción ‘Añadir al diccionario’. 2. Tras clicar en Añadir al diccionario el sistema mostrará un mensaje con la información que el usuario debe seguir para añadir la palabra al diccionario. 3. Tras pronunciar la palabra, esta será añadida al diccionario Macros 1. En la sección de gestión de Macros, el usuario debe rellenar el formulario de añadir macro. 2. Una vez rellenado el formulario, el usuario debe clicar en registrar macro.
Excepciones: Si al enviar la petición para añadir contenido a las colecciones el sistema recibe un mensaje de error de Vocali, el sistema notificará dicho mensaje al usuario.
Precondiciones: Iniciar sesión en Vocali

Diagrama de secuencia.

La fase de entrenamiento es tal vez la más intrincada, pues presenta varias ramificaciones en cuanto al comportamiento se refiere. Para representar de forma visual se presenta el diagrama de secuencia de la figura 4.2.

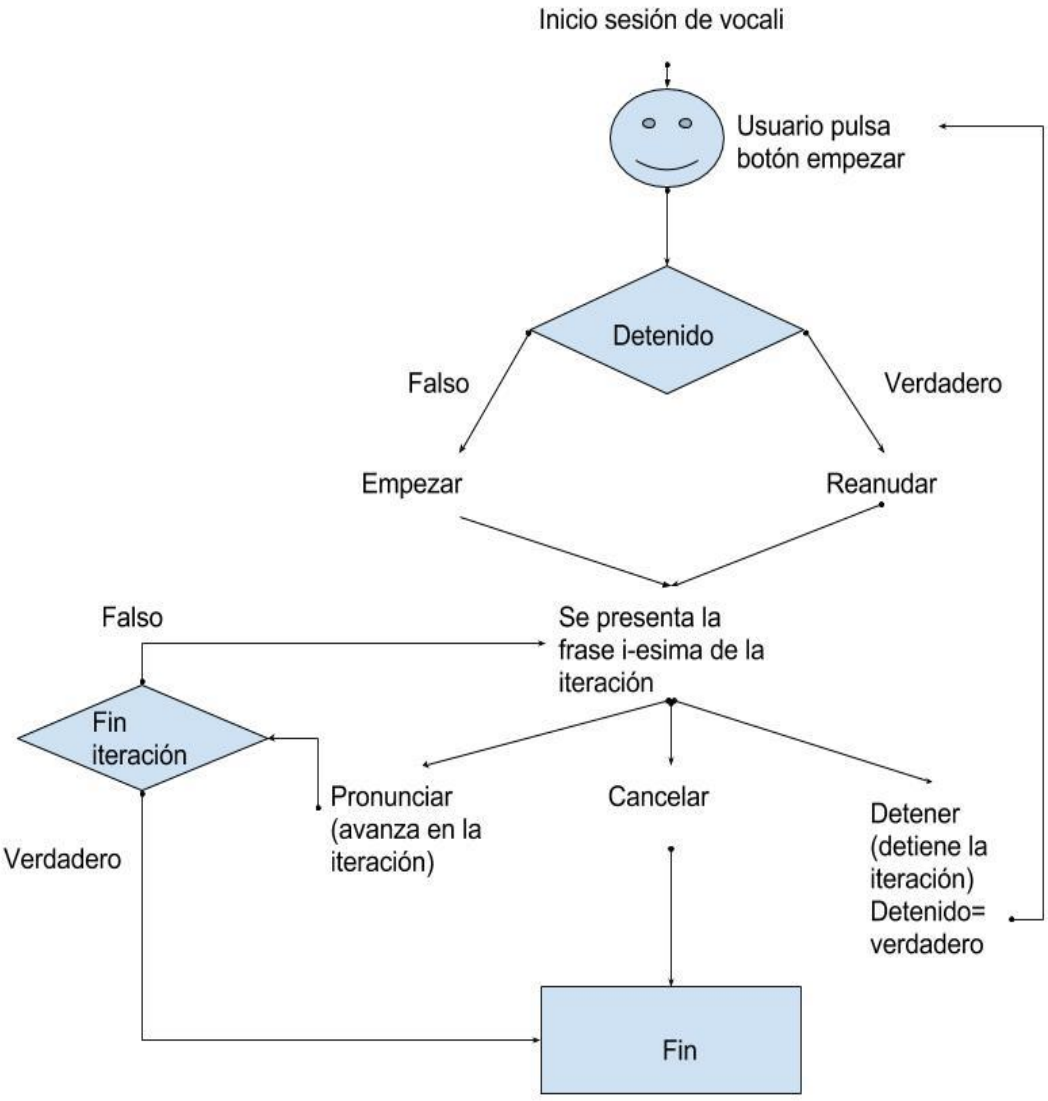


Figura 4.2 Diagrama de secuencia, fase de entrenamiento.

4.1.3 Requisitos no funcionales

Este proyecto solo presenta interfaces como requisitos no funcionales. A continuación, haremos un inciso en cómo se ha decidido implementar las interfaces y cómo el usuario interactúa con ellas.

El estado inicial de la aplicación web Actualpacs, presenta la siguiente interfaz:

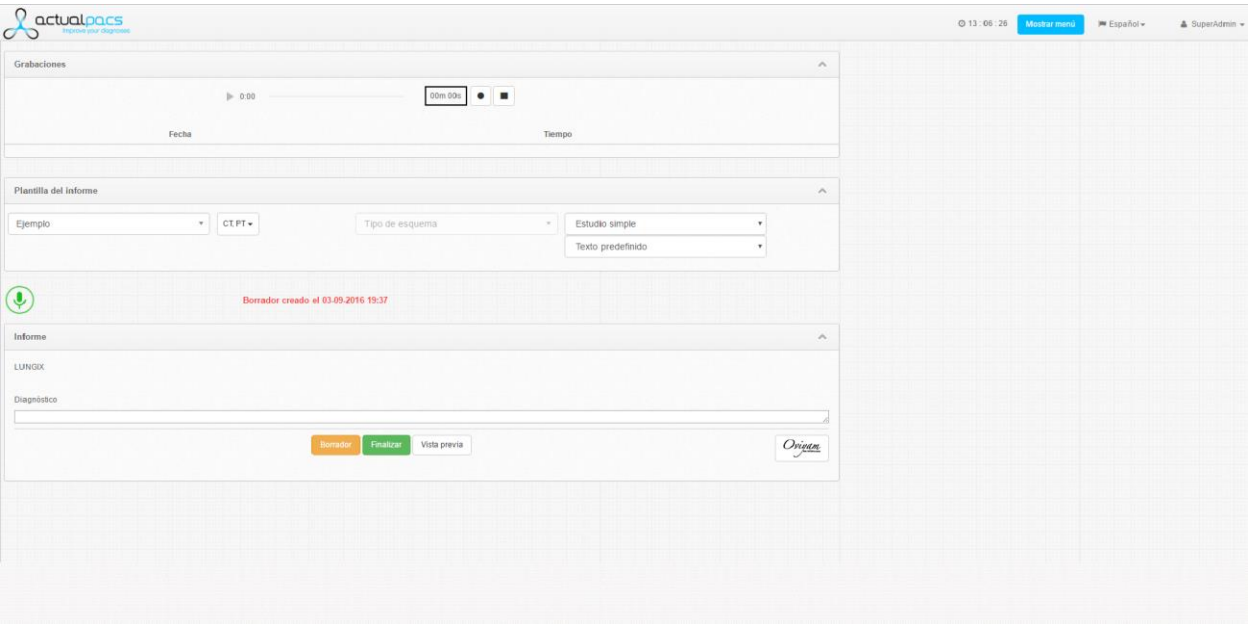


Figura 4.3 Pantalla dictado

La pantalla dispone de numerosos elementos mediante los cuales el usuario puede interactuar con la aplicación. Sin embargo, nos centraremos en un subconjunto, tal que contiene exclusivamente los elementos que tienen relación con nuestro proyecto, estos elementos son:

- Micrófono. Permite iniciar y pausar el dictado. Además, se muestra verde si el dictado está pausado y rojo si el dictado está activado, de este modo presenta de forma visual e intuitiva el estado del dictado.
- Caja de texto. Permite al usuario introducir texto y muestra el texto transcrito en caso de estar haciendo uso de reconocimiento de dictado. El sistema permite insertar varias cajas de texto via plantillas.
- Desplegable de plantillas. Permite cargar la plantilla seleccionada.

Mis aportaciones han sido en gran medida complementarias a la ya existente. A continuación, citaremos las aportaciones.

Para informar al usuario de cualquier error relacionado con Vocali, se ha implementado una caja de alerta que aparece y se desvanece en la esquina inferior de la ventana de forma estática, es decir sin importar la posición del *scroll*. La caja de alerta aparece y se desvanece en un segundo pues se ha considerado tiempo más que suficiente para leer el mensaje, que es en todo caso de una longitud no mayor de 50 caracteres. Se debe implementar de este modo para evitar la interacción del usuario para cerrar la caja de alerta, pues puede resultar tedioso en algunos casos.

El siguiente código se encarga de mostrar y ocultar el mensaje pasado un segundo:

```
function popUpMsn(msn){
  $('#alert > p').get(0).innerHTML = msn;
  $('#alert').fadeIn('slow',function(){


     setTimeout(function(){
        $('#alert').fadeOut('slow');
     },1000);
  });
}
```

El siguiente código CSS define el aspecto visual del mensaje:

```
<div class="alert" id='alert' style="
position:fixed;
display:none;
bottom: 0;
right: 0;
width: 300px;
border: 3px solid #73AD21;">
```

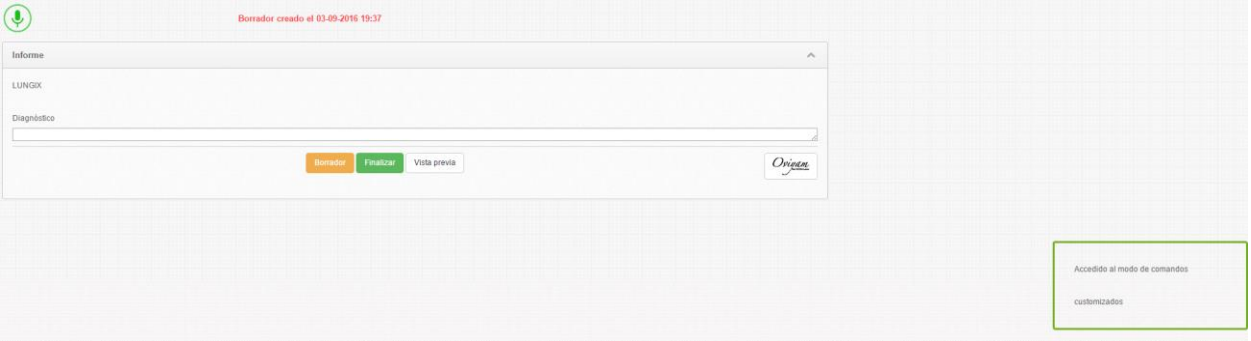


Figura 4.4 Mensajes alerta

Para insertar una palabra al diccionario se ha generado una caja de instrucciones. Dicha caja contiene la palabra a añadir y un botón “pronunciar”, que activa el ámbito pertinente para añadir una palabra al diccionario. Una vez se reciba el mensaje de que la palabra se ha añadido con éxito, la caja de instrucciones se ocultará automáticamente, una vez más para evitar interacciones tediosas con el usuario.

El siguiente código se encarga de ocultar la caja de instrucciones una vez se ha añadido la palabra al diccionario.

```
imdSession.ActionHandlerTable[invoxmd.wsMessage.DICTIONARY_ADD_WORD] = function (response, session) {
    if (response.ExceptionType == invoxmd.exceptionType.NONE) {
        popUpMsn(message_added_word);
        $("#pronunBox").finish().hide(100);
    }
}
```

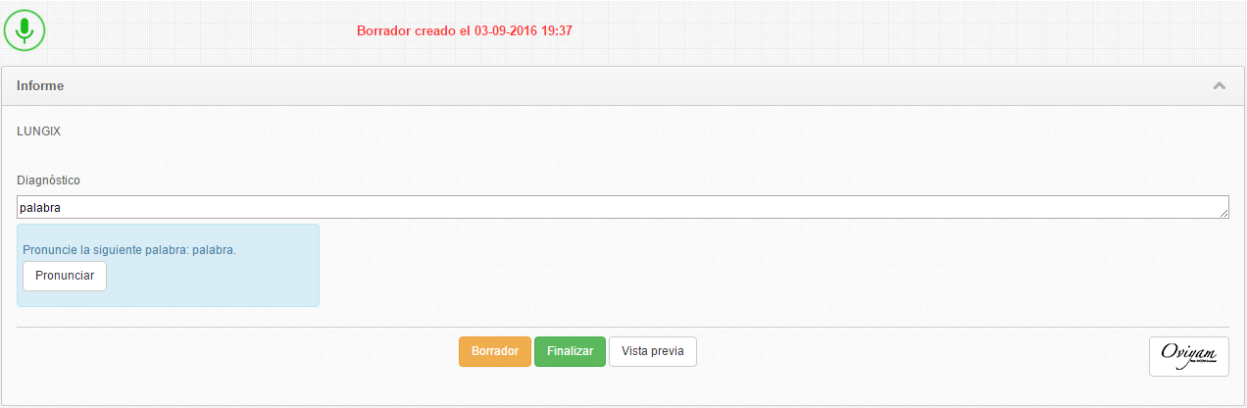


Figura 4.5 Caja de instrucciones

Para introducir las credenciales de Vocali se ha generado un formulario simple en la sección de configuración

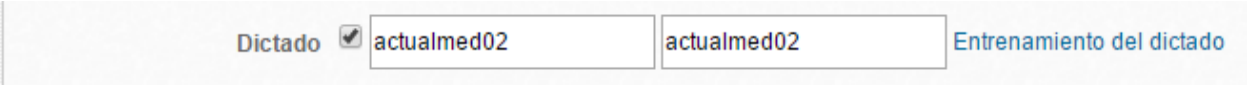


Figura 4.6 Formulario credenciales Vocali

La pantalla de dictado ha sido generada desde cero, dado que no existía en el estado inicial del proyecto. A continuación, comentaremos los elementos más destacables:

- Sección de instrucciones: en la parte superior se ubica la caja de información que enumera una serie de pautas para realizar el entrenamiento.
- Micrófono: se comporta de forma similar al micrófono de la pantalla del dictado, gran parte del código ha sido aprovechado.
- Sección del entrenamiento: sección en la que aparecen las frases a pronunciar.
- Botón empezar: Inicia el entrenamiento y reanuda en caso de que esté detenido.
- Botón detener: Detiene el entrenamiento.
- Botón cerrar: Cancela el entrenamiento.

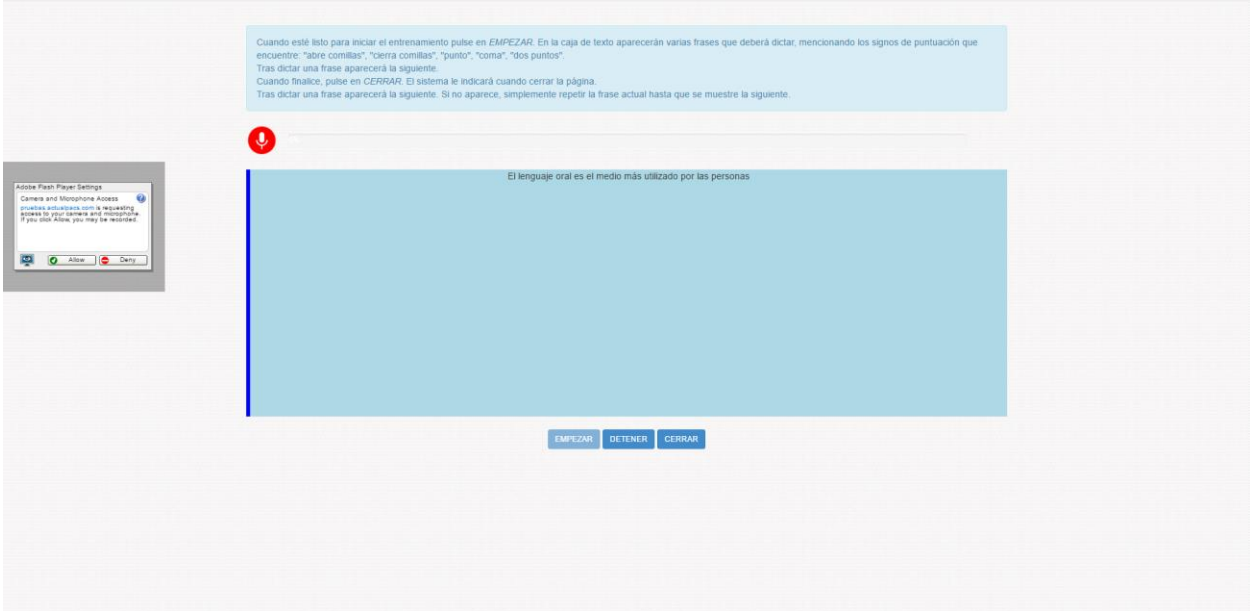


Figura 4.7 Pantalla dictado

Capítulo 5. Implementación y pruebas

En este capítulo, detallaremos cómo se ha implementado cada una de las tareas del proyecto. Además, trataremos aspectos tales como qué estructuras de datos se han usado y el porqué, así como que tipo de dificultades en materia de implementación han surgido durante el transcurso del proyecto.

Los parámetros de conexión se extraerán de la sesión de usuario, para no tener que acceder a la sesión continuamente se compartirá con todas las vistas:

```
View::share('uses_dictation',          (bool)          Session::get('user.dictation'));
View::share('dictation_user',                        Session::get('user.vocali_user'));
View::share('dictation_pwd', Session::get('user.vocali_pwd'));
```

Antes de profundizar en cada tarea en particular se expondrá cómo el sistema trata las sesiones de Vocali. El sistema es capaz de captar si dos instancias de sesiones de Vocali están intentado iniciar sesión con el mismo usuario. Dado este caso, el sistema evita el inicio de sesión y sólo será procesado mientras haya una sesión por usuario.

Para escuchar los eventos referentes a las sesiones de Vocali haremos uso del *listener* SESSIONACTEVENTHANDLER , proporcionado en la librería de Vocali. El siguiente fragmento de código es el encargado de detectar sesiones duplicadas y actuar en consecuencia.

```
session.SessionActEventHandler = function (msg) {
    if (imd_reconnect == true && msg.Act == invoxmd.sessionAct.NET_CONNECTION_DUP)
    {
        imd_reconnect = false;
        $('#micro_status').hide();
    }

}
```

Para abordar la incorporación de varios idiomas introduciremos el texto en el fichero PHP de idiomas, que a su vez contiene un mapa donde la clave es un texto identificador y el valor el texto traducido al idioma correspondiente. Hay tantos ficheros de idiomas como queramos incorporar, en nuestro caso inglés y castellano que ya existían en Actualpacs. Para referenciar el texto que deseamos introducir haremos uso de la siguiente función propia del *framework* Laravel:

```
trans('language.{mensaje}')
```

A continuación, se muestra un ejemplo del fichero de idiomas.

```
<?php

return array(

        'mensaje'   =>      'New register',

);
```

5.1 Tarea 1. Incorporar *feedback* del reconocimiento de voz al usuario.

En el apartado Objetivos específicos del proyecto del capítulo 1, se expuso el estado inicial de la funcionalidad del dictado y cómo éste requería de mejoras respecto al *feedback*. Por otro lado, en el capítulo 3.3, se planteó el dictado desde un punto de vista más general, pero sin entrar en demasiados detalles de programación. En este punto cubriremos dichos detalles que han quedado por desarrollar.

Variables

Las siguientes variables corresponden al código JS init.js encargado de implementar los servicios de Vocali.

VARIABLE	DESCRIPCIÓN
CARETPREHIDDEN	Posición del cursor la primera vez que se reconoce texto.
PRERECOGLENGTH	Longitud del texto transcrito.
TARGET	Caja de texto de tipo textarea sobre el que se está transcribiendo. Cambia su valor al clicar en una caja distinta, por defecto el sistema asigna el campo diagnóstico o el primero que recoja de la jquery en su defecto. El siguiente código será el encargado de asignar el valor de target: `function findActiveField() { if(target == undefined) target = $('[name=diagnostic]').get(0); if(target == undefined) target = $('#report_wrapper textarea').get(0);. if(target == undefined) target = $('#report_wrapper div[contenteditable]').get(0); $('#report_content').on('click', 'textarea, div[contenteditable]', function(){ target = $(this); }); return target; }`

Implementación

Como se introdujo en el capítulo 3.3, esta tarea hace uso de dos etiquetas HTML que representan secciones de texto: Textarea y div content editable. Aunque en el capítulo 3.3 se expuso porque era necesario el uso de dichas etiquetas, en este capítulo detallaremos qué características de éstas son de interés.

Etiqueta textarea

- la etiqueta textarea define una entrada de texto multi-línea, necesaria para escribir.
- Textarea permite un número ilimitado de caracteres, el texto se renderiza en función a una anchura definida, lo que nos facilita dar formato mediante CSS.
- Las proporciones del textarea se pueden especificar en los atributos fila y columnas de la misma o incluso mejor via CSS

Etiqueta div content editable

- La etiqueta div define una sección en el documento HTML.
- La etiqueta agrupa elementos y permite darles formatos via CSS.
- El atributo content editable permite la entrada de texto en una sección div, necesario para escribir.

Para llevar a cabo la integración de *feedback*, se volcará el texto parcial transcrito en la variable *target*. Además, dado que textarea no admite marcado de texto, se mostrará el contenido de dicho textarea en una sección HTML div.

Esta tarea requiere eliminar y añadir las transcripciones de texto reiterativamente. Para ello haremos uso de las funciones insertAtCursorPosition y removeTestRecog, que en combinación con las variables que hemos mencionado nos permitirán eliminar y añadir la transcripción en la posición de cursor deseada:

```
  function insertAtCursorPosition(field, text,startPos,endPos) {
    if ('selectionStart' in field) {    // Standard.
        var scrollTop = field.scrollTop;
        field.value = field.value.substring(0, startPos) + text +
field.value.substring(endPos);
        field.focus();
        field.selectionStart = startPos + text.length;
        field.selectionEnd = startPos + text.length;
        field.scrollTop = scrollTop;
    } else if (document.selection) {    // IE.
        field.focus();
        var sel = document.selection.createRange();
        sel.text = text;
        field.focus();
    }
}
function removeTextRecog(){
    var endPos = caretprehidden+preRecogLength;
    var startPos = caretprehidden;
    insertAtCursorPosition($(idfieldpar).get(0), '',startPos,endPos);
}
```

Las acciones a tomar cada vez que se transcribe texto dependen del tipo de transcripción. Vocali distingue 3 tipos: Aceptado, Parcial, Rechazado (tabla 4.1). Obtendremos el tipo de la transcripción como parámetro del *listener* del evento:

```
session.Recognizer.DictationEventHandler = function (msg) {
```

```
        if (msg.Subtype == invoxmd.dictationEventType.ACCEPTED) {
        }
        if (msg.Subtype == invoxmd.dictationEventType.REJECTED) {
        }
        if (msg.Subtype == invoxmd.dictationEventType.PARTIAL) {
        }

        }
```

Los tipos de transcripción denegado y desconocido no requieren de ningún tipo de acción por parte del sistema. Sin embargo, contemplaremos el caso de denegado, para mostrar la palabra que no se ha llegado a reconocer por consola. En el caso de desconocido no podemos extraer ningún tipo de información, por lo tanto, no será contemplado.

Parcial:

En una transcripción parcial cabe diferenciar si ya se ha transcrito texto previamente.

Si no hay texto transcrito el sistema iniciará las variables necesarias para llevar a cabo el volcado de la transcripción, de lo contrario, eliminará las transcripciones generadas hasta el momento. A partir de este punto el sistema actúa de forma idéntica en ambos casos, pues sólo procede actualizar la caja de texto textarea y la sección div con la transcripción actual.

Rechazada:

Si la transcripción ha sido rechazada, el sistema borrará la transcripción realizada hasta el momento y reiniciará las variables del dictado.

Aceptada:

En este caso la transcripción ha sido aceptada y ya es inmutable, por lo tanto, se cohesiona el texto con el actual. Para ello se tendrán en cuenta aspectos gramaticales, tales como, mayúscula después de puntos, espacio después de coma, etc. Para abordar este cometido disponemos de una función suministrada en la librería de Vocali.

5.2 Tarea 2. Integración de la fase de entrenamiento.

Enrutado

Previa a la implementación de esta tarea procede montar el enrutado para que el usuario sea capaz de acceder a la funcionalidad.
La URL y el controlador a través de los cuales accederemos se definen en el archivo de rutas de la siguiente forma:

```
Route::get('account/users/{id}/training/{user}/{pwd}',                    array(
'as'                              =>                          'account.users.training',
'uses'                             =>                        'UsersController@training'
));
```

El controlador se encarga de invocar la vista, así como facilitar los parámetros necesarios. El contenido de la función *training* es el siguiente:

```
public function training($user_id, $dictation_username, $dictation_password)
{
   $data = ['username' => $dictation_username, 'pwd' => $dictation_password];
   return View::make('account::users.training', $data);
}
```

Variables

Las siguientes variables corresponden al código JS training.js encargado de implementar los servicios de Vocali.

Variable	*Descripción*
trainingsentences	Frases que el usuario ha pronunciado hasta el momento.

Implementación

La funcionalidad de entrenamiento mostrará iterativamente las frases que el usuario debe pronunciar. Cada vez que se complete una frase, las frases pronunciadas hasta el momento aparecerán de color verde, aparecerá una nueva frase a pronunciar y se actualiza la barra de progreso. Además, durante el transcurso del entrenamiento, el usuario puede cancelar o pausar el dictado. Cuando se complete la pronunciación de las frases el sistema notificará al usuario que el entrenamiento ha sido finalizado con éxito.

La frase a dictar es extraída del evento, a su vez, el evento es disparado de forma iterativa hasta que se complete el entrenamiento.

```
session.Trainer.TrainingSentenceEventHandler = function (evt) {…}
```

Dado que el usuario puede haber cancelado y reanudado el dictado, siempre que tratemos una frase debemos tomar la precaución de reiniciar las frases de entrenamiento, así como la caja donde se muestran.

```
    if (evt.NextSentence) {
        if(trainingsentences.length>0){

$('#container').get(0).innerHTML="<span>"+trainingsentences.fontcolor("green")+"</span
>";
        }
        if(evt.Index==1){
            $('#container').get(0).innerHTML="";
           trainingsentences="";
        }
        var sentence=evt.NextSentence;
        $('#container').get(0).innerHTML+=sentence;
```

```
        trainingsentences+=sentence+"<br>";
        var max = evt.NumSentences;
        var value = evt.Index;
        $('#prog').get(0).value=(value*100)/max;
    }
```

Cuando termine el entrenamiento el sistema mostrará un mensaje de éxito.

```
else {
        trainingsentences ="";
        $('#container').get(0).innerHTML="Entrenamiento finalizado, puede cerrar la
ventana";
        doStopTrainer();
    }
}
```

5.3 Tarea 3 y 4. Integración de Macros y Diccionarios.

Variables

Las siguientes variables corresponden al código JS init.js encargado de implementar los servicios de Vocali.

Variable	*Descripción*
dicWord	Palabra a añadir al diccionario

Implementación

El usuario accede a la funcionalidad de diccionarios, pulsando añadir en el menú desplegable que aparece seleccionado la palabra que desea añadir al diccionario. El siguiente fragmento de código se encargará de mostrar el menú. Además, asegura que solo una palabra está seleccionada y que la palabra seleccionada corresponde a una palabra de la caja de texto de dictado.

```
$(document).on('select', '#report_content textarea, #report_content
div[contenteditable]', function(event){
    dicWord=getSelectionText();

    while(dicWord.charAt(0)==' ' && dicWord.length>0){
        dicWord=dicWord.substring(1);
    }
    while(dicWord.charAt(dicWord.length-1)==' ' && dicWord.length>0){
        dicWord=dicWord.substring(0,dicWord.length-1);
    }

    if(!dicWord.includes(' ') && dicWord.length>0 ){
        $('#addToDict').finish().toggle(100);
```

```
    }
});
```

Una vez se ha mostrado el menú desplegable, el usuario puede pulsar en la opción deseada o bien ocultar el menú pulsando en cualquier lugar salvo el menú. Para ello, el sistema extraerá de un evento de pulsado, si el elemento pulsado cuelga del menú desplegable.

```
// si el documento es clicado fuera del menu
$(document).bind('mousedown', function (e) {
    // si el elemento clicado no es el menu
    if ($(e.target).get(0).id!='addToDict') {
        //esconderlo
        $('#addToDict').hide(100);
    }
    if (!$(e.target).parents('#pronunBox').length > 0) {
            var buttonswitch = ( $('#addWord').get(0).style.display == 'none' ?
$('#addWord').get(0) : $('#cancelarpron').get(0));

            if( buttonswitch == $('#addWord').get(0) ){
                $('#addWord').get(0).style.display='block';
                $('#cancelarpron').get(0).style.display='none';
            }
            var isphoneTrans = (imdSession.Recognizer.CurrentScope ==
invoxmd.dictationScope.PHONETIC_TRANSCRIPTION);
            if(isphoneTrans){
                imdSession.Recognizer.setScope(invoxmd.dictationScope.PAUSED);
            }
        $('#pronunBox').finish().hide(100);
        //session.Recognizer.setScope(invoxmd.dictationScope.PAUSED);
    }
});
```

Una vez el usuario pulsa en la opción añadir al diccionario, el sistema activará el ámbito phonetic *transcription* de donde el sistema extraerá la trascripción fonética a añadir al diccionario. Para ello atraparemos el evento y crearemos los objetos *word* y *pronunciation* (tablas 4.3 y 4.4) para añadir la palabra de la siguiente forma:

```
imdSession.Recognizer.PhoneticEventHandler= function(msg) {
    imdSession.Dictionary.addWord(new invoxmd.Word(dicWord,[new
invoxmd.Pronunciation(msg.Transcription,0)], false));
    imdSession.Recognizer.setScope(invoxmd.dictationScope.RUNNING);
    $("#pronunBox").finish().hide(100);
}
```

El procedimiento para añadir macros es similar, salvo el detalle que para las macros haremos uso de un formulario sencillo, ubicado en la sección de configuración.

5.4 Tarea 5. Integración de comandos por voz

En el capítulo 1, en los objetivos específicos del proyecto, diferenciamos entre comandos simples y *custom*, a la par que indicando las particularidades de cada uno. A continuación, se expondrán los aspectos de implementación de cada uno de estos tipos de comandos.

Integración de comandos simples.

Implementación

Los comandos simples son definidos en un *listener* de eventos de comandos, suministrado en la librería de Vocali, cuyo nombre es `CommandEventHandler`. Dependiendo de el comando reconocido que es recibido como parámetro del evento, definiremos un comportamiento. Para ello haremos uso de un *switch* y definiremos en cada sección *case* el comportamiento particular. Además, debemos eliminar cualquier texto transcrito parcial que haya hasta el momento, pues formaba parte del comando reconocido. El siguiente fragmento de código muestra este planteamiento.

```
imdSession.Recognizer.CommandEventHandler = function(msg) {
    console.log('COMMAND RECOGNIZED: ' + msg.CommandId + ' / ' + msg.Hypothesis.Text);
    if(isTextRecog()){
        removeTextRecog();
        showTextArea();
        $(targetPar).get(0).setSelectionRange(caretprehidden,caretprehidden);
        preRecogLength=0;
    }
    switch (msg.CommandId) {
        case invoxmd.commandId.{Comando}:{

            imdSession.Recognizer.setScope(invoxmd.dictationScope.RUNNING);
            break;
        }

    }
}
```

Los comandos simples no presentan gran complejidad, simplemente buscan la cadena de texto sobre la que actúa el comando y efectúa los cambios pertinentes. Para ello separaremos el texto por espacios en un *array* para facilitar la búsqueda.

```
function textSelectionCommand(msgCommandId, rightHS){
    var indiceCurs=indiceCursor();
    if(indiceCurs == -1){
        return null;
    }
    var valuesplit=$(target).get(0).value.split(' ');
    var espacios=rightHS.split(' ').length-1;
    var cursorParc=0;
    var encontradoIzq=false;
    var encontradoDer=false;
    var frase='';
    for (i=0; i < valuesplit.length-espacios; i++) {
        frase=valuesplit[i];
        if(espacios>=1){
```

```
            for(j=i+1; j<=espacios+i; j++){
                    frase+=' '+valuesplit[j];
                }
            }
        if (i<indiceCurs){
            if (frase.toLowerCase()==rightHS.toLowerCase()  && !encontradoIzq ) {
                    if(msgCommandId==invoxmd.commandId.GO_TO){
                        $(target).get(0).setSelectionRange(cursorParc,cursorParc);
                        encontradoIzq=true;
                    }
                    else if(msgCommandId==invoxmd.commandId.GO_TO_AFTER){
                        $(target).get(0).setSelectionRange(cursorParc + rightHS.length
, cursorParc + rightHS.length);
                        encontradoIzq=true;
                    }
                    else if(msgCommandId==invoxmd.commandId.SELECT){
                        $(target).get(0).setSelectionRange(cursorParc, cursorParc +
rightHS.length);
                        encontradoIzq=true;
                    }
                }
        }
        else{
            if(i>indiceCurs){
                if (frase.toLowerCase()==rightHS.toLowerCase()  && !encontradoDer ) {
                    if(msgCommandId==invoxmd.commandId.GO_TO){
                        $(target).get(0).setSelectionRange(cursorParc,cursorParc);
                        encontradoDer=true;
                    }
                    else if(msgCommandId==invoxmd.commandId.GO_TO_AFTER){
                        $(target).get(0).setSelectionRange(cursorParc + rightHS.length
, cursorParc + rightHS.length);
                        encontradoDer=true;
                    }
                    else if(msgCommandId==invoxmd.commandId.SELECT){
                        $(target).get(0).setSelectionRange(cursorParc, cursorParc +
rightHS.length);
                        encontradoDer=true;
                    }
                }
            }
        }
        cursorParc+=valuesplit[i].length+1; //la longitud de la palabra + espacio
    }
}
```

Integración de comandos *custom*.

Implementación

El primer paso para abordar los comandos *custom*, es crear la estructura de objetos Composite que representarán nuestro comando (tabla 4.6 y 4.7). En nuestro caso queremos generar el comando plantilla {nombre_plantilla}. Dado que necesitamos los nombres de las plantillas los pasaremos a la vista desde la tabla plantilla (figura 2.2) de la siguiente forma

```
$this->data['design_templates'] = Design::getDefaultDesignList($user);
```

El *array* de plantillas está almacenado en una variable design_templates extraída de la sesión de usuario. Sin embargo, necesitamos acceso a esa variable desde JS, pues es donde generamos el código de carga de plantilla.

Desde la vista tenemos acceso a la variable design_templates y podemos declarar las variables JS codificadas en JSON dentro de una etiqueta <script>, afortunadamente PHP y JS disponen de numerosas librerías para codificar y decodificar mensajes JSON, de la siguiente forma:

```
<script>
var design_templates = {{json_encode($design_templates)}};
<script>
```

Una vez disponemos de todos los requisitos de datos podemos proceder a crear el comando. Primero se genera la parte invariable del comando es decir “plantilla” de la siguiente forma:

```
var csb = imdSession.RecognizerEngine.CommandSpecBuilder;
var textoComando = csb.buildText('Plantilla');
```

A continuación, especificamos la parte variable del comando en una *array* de la siguiente forma:

```
var arrayOpciones=[];


for (i=0; i<optionVal.length;i++){
    var mad = csb.buildText(optionVal[i]);
    mad.addSemanticInfo('OPCION', new invoxmd.StringSemanticInfo(optionVal[i]));
    arrayOpciones.push(mad);
}
```

Una vez hemos definido ambos partes de comando, podemos crear la secuencia de activación del comando de la siguiente forma:

var command = csb.buildSequence([textoComando, csb.buildOptions(arrayOpciones)]);

Llegados a este punto, hemos definido la semántica del comando. En último lugar debemos definir el comportamiento del comando. Para ello definiremos la función anónima que se asignará al *listener*, el cometido de la función es cargar la plantilla seleccionada en el comando.

```
grammar.CommandEventHandler = function(msg) {
        if (msg.Semantic['OPCION'].StringValue){
            console.log('CUSTOM COMMAND RECOGNIZER: ' +
msg.Semantic['OPCION'].StringValue + ' / ' + msg.Hypothesis.Text);
            loadDesign(optionIndex[msg.Semantic['OPCION'].StringValue]);
            imdSession.Recognizer.setScope(invoxmd.dictationScope.RUNNING);
        }
```

```
    }
imdSession.RecognizerEngine.addCommandGrammar(grammar);
```

Para carga el diseño haremos uso de Ajax, para realizar la petición. Como resultado de la petición recibiremos la representación HTML de la plantilla seleccionada, que será insertada en la sección HTML de la plantilla cuyo identificador es “report content”:

```
loadDesign = function()
{
  var design_template = $('select[name="design_template"]').val();
  var study_iuid = $('input[name=study_iuid]').val();
  var cname = $('input[name=cname]').val();
  // Check if there is a template
  if ( design_template !== undefined )
  {
     $.ajax({
        type : "post",
        dataType : 'json',
        url : base_url + "report/PostReportContent",
        data : {id:design_template, 'cname':cname, 'study_iuid': study_iuid},
        success : function(data)
        {
           $('#report_content').empty();
           $('#report_content').append(data.txt);
           $('#report_content textarea').css('height', '20px');
           //Habilito el select de modalidad
           $('select[name="report_template_modality[]"]').multiselect('enable');
           $('select[name="report_template_modality[]"]').change();
           if (data.attach_images == 1) {
              $( "#attached_images" ).removeClass( "hidden" );
           }else{
              $( "#attached_images" ).addClass( "hidden" );
           };
           checkAddendum(addendum_enabled);
```

```
            }
        });
    }
    else
    {
        //Deshabilito el select de modalidad y de esquema(plantilla)
        $('select[name="report_template_modality[]"]').multiselect('disable');
        $('select[name=report_template_type]').attr('disabled',
true).trigger("chosen:updated");
        $('#report_content').empty();
    }

    target = undefined;
    $(document).on('focusin', '#report_content textarea, #report_content
div[contenteditable]', function(){
        target = $(this).get(0);
    });
}
```

Verificación y validación.

Durante el transcurso del proyecto se han realizado varias pruebas unitarias, es decir en entornos muy concretos y con un alcance limitado a funcionalidades concretas. Para ello se ha usado el *framework* de pruebas PHPunit integrado en Laravel y PHPBrowser para emular el navegador. Las pruebas han sido verdaderamente útiles para comprobar que las distintas funcionalidades siguen funcionando tal y como se añaden nuevas funcionalidades. Además, son interesantes por si en algún momento se decide refactorizar el código o realizar un procedimiento de ingeniería inversa dado que se pueden lanzar estas pruebas cada vez que se efectúe un cambio para comprobar que todo sigue funcionando correctamente.

Sin embargo, estas pruebas en un entorno local no son suficientes y una vez terminado el proyecto se debe probar en un entorno de integración. Para ello Actualmed dispone de servidores de prueba. Las pruebas han sido realizadas por el supervisor del proyecto y tras dar el visto bueno han sido preparadas para incluir en la nueva versión de Actualpacs.

Los navegadores en los que se han realizado las pruebas son: Chrome, Internet Explorer, Safari y Firefox, pues son a los se da soporte.

El alcance de las pruebas no se limita solo a la funcionalidad, también se han contemplado aspectos gramaticales tanto en el inglés como en castellano.

Capítulo 6. Conclusión.

Mi estancia en la empresa durante el transcurso de las prácticas ha sido verdaderamente gratificante. No solo he reforzado conocimientos adquiridos en el grado de ingeniería informática, sino que he desarrollado nuevas competencias tales como desarrollo web en JS, Jquery y PHP. Además, he tenido la oportunidad de integrarme en un equipo de desarrollo real, con todo lo que conlleva, es decir adaptarse a ciertas maneras de trabajar, usar herramientas en común, coordinación, comunicación, usar un sistema de control de versión en común etc. A pesar de que en varias asignaturas se contemplan todos estos aspectos mencionados no se llegan a poner completamente en práctica en un contexto real, por lo que las prácticas de empresa se presentan como un gran reto y oportunidad para cualquier estudiante. Otro aspecto que creo que merece la pena recalcar es el uso intensivo de documentación que ha requerido este proyecto pues se han hecho uso de varias librerías y *frameworks* que desconocía.

El resultado del proyecto ha sido satisfactorio a mi parecer, pues cumple con las expectativas marcadas en la planificación inicial. Si bien algunas funcionalidades sencillas has tomado una cantidad de tiempo considerable, esto se debe al tiempo invertido en investigar la mejor forma de abordar la implementación.

La tecnología utilizada se adapta perfectamente a las necesidades del proyecto, sin bien, en algunas situaciones un equipo más potente hubiera agilizado el proyecto, no supuso un contratiempo destacable.

Desde mi punto de vista, a largo plazo el proyecto está sujeto a varias mejoras. Una funcionalidad interesante a añadir sería la implementación de una sección en la que el propio cliente fuera capaz de eliminar registros tanto en diccionarios como en macros.

Para finalizar, simplemente decir que no encuentro ninguna pega destacable y en su conjunto las prácticas de empresa han sido una gran experiencia.

Bibliografía

[1] http://www.infojobs.net/gen/multisite

[2] ij.xhtml?iCodigoPerfil=98495056525550659911612011143616206188&perfil=actualtec-innovacion-tecnologica-s.l.

[3] https://github.com/airbnb/javascript

[4] http://www.w3schools.com/

[5] https://laracasts.com/

[15] http://www.phptherightway.com/

[6] Getting Stuff Done with Laravel 4 A journey through application design and development using PHP's hottest new framework. Chuck Heintzelman. https://leanpub.com/gettingstuffdonelaravel.

[7] Referencia rápida. Joaquín Ataz López Murcia. Diciembre de 2004. http://es.tldp.org/Tutoriales/doc-tutorial-vim/Guia-Vim.pdf.

[8] Laravel: Code Bright Web application development for the Laravel framework version 4 for beginners. Dayle Rees. https://daylerees.com/codebright/

[9] Beauty Creating Beautiful Web Apps with Laravel 5.1. Chuck Heintzelman. https://leanpub.com/l5-beauty

[10] Laravel Testing Decoded The testing book you've been waiting for. Jeffrey Way. https://leanpub.com/laravel-testing-decoded

[11] INVOX Medical Dictation - Manual SDK.pdf.

[12] INVOX Medical Dictation - Referencia SDK.pdf.

[13] Executable Specifications with Scrum a Practical Guide to Agile Requirements Discovery. Mario Cardinal. ISBN10: 0132776510. ISBN13: 9780132776516.

[14] Código limpio Manual de estilo para el desarrollo ágil del software. Editor: ANAYA MULTIMEDIA; Edición: edición (19 de junio de 2012). ISBN-10: 8441532109.
ISBN-13: 978-8441532106.

APENDICE A Índice de TABLAS