Grado en Ingeniería Informática

Trabajo Final de Grado

Reconocimiento de mensajes con *Dejavu* y detección de palabras durante una llamada con Freeswitch

Autor:
Abraham Benítez Sabao

Supervisor:
Oriol Flors Mas
Tutor académico:
Rafael Berlanga Llavorí

Fecha de lectura: 25 de junio de 2018
Curso académico 2017/2018

Resumen

En este documento se detalla el desarrollo de dos proyectos dentro del marco formativo de la estancia en prácticas dentro de la empresa Nayar Systems. Ambos proyectos tienen en común el reconocimiento de voz donde, el primero (proyecto Dejavu) reconoce audios que han sido grabados anteriormente y el segundo (proyecto Freeswitch) detecta palabras clave durante una llamada telefónica.

El proyecto Dejavu consiste en modificar un servicio existente en la empresa para no depender del servicio *speech recognition* de Google, mientras que el proyecto Freeswitch consiste en un sistema de atención telefónica por comandos de voz, con tres acciones diferentes.

Para ambos proyectos se ha seguido la metodología Scrum, con la cual se han alcanzado los resultados esperados en el tiempo que se había planificado.

Palabras clave

Freeswitch, sonido, VoIP, telefonía.

Keywords

Freeswitch, sound, VoIP, telephony.

Índice general

Capítulo 1

Introducción

Índice del capítulo

1.1. Contexto y motivación del proyecto

En esta sección se describe la empresa y el proyecto que se ha propuesto, así como la necesidad, utilidad y objetivos del mismo.

1.1.1. La empresa

Nayar Systems es una empresa especializada en ingeniería de las telecomunicaciones. Nayar Systems es la matriz principal de sus tres marcas comerciales: Advertisim, Net4Machines y 72horas. Ésta última es donde reposa el resultado del proyecto que a continuación vamos a definir. Cada marca comercial se dedica a un tipo de servicio diferente, a saber:

- Advertisim: Plataforma de comunicación para la gestión de contenidos multimedia.
- Net4Machines: Servicio de VPN para conectividad entre máquinas.
- 72horas: Servicio de asistencia y programación remota de telealarmas para ascensores.

La empresa tiene dos ubicaciones, la primera en Castellón de la Plana en la Calle Taxida, nº10 y la segunda en CEEI de Castellón de la Plana en Ronda Circunvalación, 188, lugar donde se realizaron las prácticas con el equipo de I+D+I de 10 personas.

1.1.2. El proyecto

El proyecto formativo de la estancia en la empresa, Nayar Systems, consiste en definir una solución a un problema que tiene actualmente y ofrecer una funcionalidad extra. En primer lugar, la marca 72horas realiza llamadas diarias a los dispositivos instalados en los ascensores a los que proporcionan servicio, con el fin de comprobar su estado (conectado/no conectado). Para ello, se realiza una llamada telefónica a dicho dispositivo y se graba el audio de la llamada en formato *wav*. Esta grabación se envía al servicio de reconocimiento de voz de Google, que devuelve una transcripción del audio, la cual, se analiza para conocer el estado de la línea, ya que normalmente, cuando no está disponible, la operadora reproduce un mensaje de buzón de voz u otro indicando el problema.

Más adelante nos referimos a parte "asíncrona" y parte "síncrona" las que corresponden a cada subproyecto, Dejavu y Freeswitch respectivamente.

En el caso que Google decida hacer este servicio de trascripción de textos de pago, puede suponer un problema para la empresa, ya que el servicio se usa de forma intensiva y supondría un coste de miles de euros al mes.

Como se puede ver en la figura 1.1, se requiere de una aplicación intermediaria entre el servicio de tratamiento del audio grabado y el servicio de Google. Esta aplicación debe ser capaz de reconocer audios analizados previamente, para así reducir al mínimo el uso del actual servicio de Google. Para hacer esto, el *product owner* propone la librería de audio *fingerprinting* Dejavu.

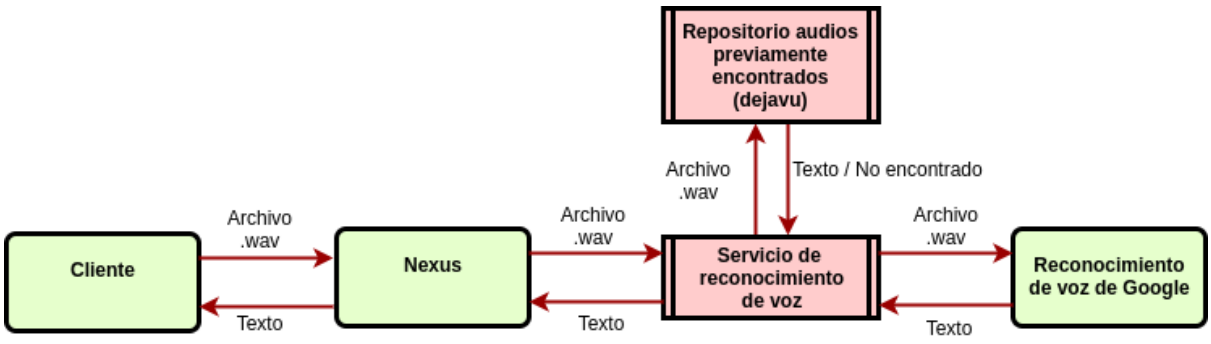

Figura 1.1: Diagrama del sistema del proyecto Dejavu.

En segundo lugar, esta misma marca pretende investigar la viabilidad en que un servicio intermedio entre un cliente y un operador sea capaz de ofrecer diferentes tipos de asistencia durante una llamada en vivo o en *streaming*. En la figura 6.4 se muestra el diagrama de dicha solución.

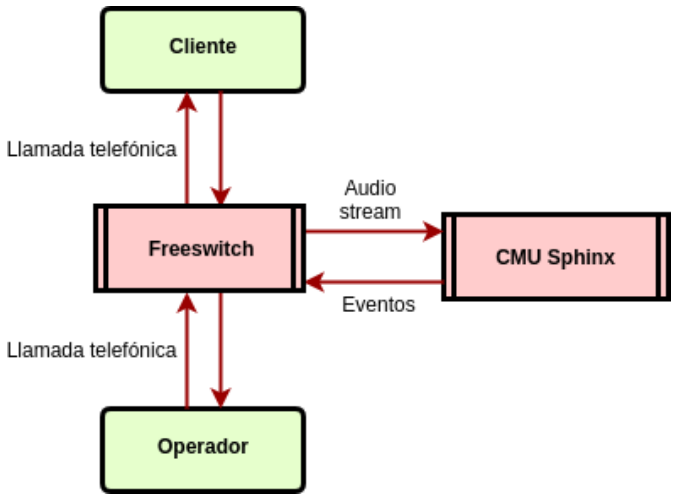

Figura 1.2: Diagrama del sistema del proyecto Freeswitch.

Esta funcionalidad añadida permitirá a Nayar Systems activar distintos servicios para sus clientes dependiendo de ciertas palabras clave, lo cual no sólo mejorará el tiempo de respuesta del mismo, si no que podrá mejorar la eficiencia de las respuestas de los servicios de 72horas.

El alcance de este sistema (proyecto Dejavu) incluye el desarrollo de un sistema asíncrono junto con una base de datos para almacenar los audios grabados, con el fin de detectar qué mensaje de voz se ha obtenido comparándolos con los grabados en la base de datos, antes de tener que usar el servicio de reconocimiento de voz de Google.

Se incluye la implementación de un sistema síncrono (proyecto Freeswitch) capaz de recibir llamas telefónicas y reconocer ciertas palabras clave durante la misma que permitan diferenciar casos.

1.2. Objetivos del proyecto

Para que los proyectos se consideren completados se deben cumplir los siguientes objetivos:

El objetivo del proyecto Dejavu es crear un sistema que, al recibir un archivo de audio, sea capaz de reconocer su contenido comparándolo con otros audios recibidos con anterioridad y, en el caso que no encuentre coincidencia, solicitar al servicio *speech* de Google la transcripción del mismo y almacenarlo.

Para el proyecto Freeswitch se pretende desarrollar un sistema que durante una llamada telefónica sea capaz de detectar ciertas palabras clave y poder ejecutar distintas acciones dependiendo del comando detectado.

Capítulo 2

Descripción del proyecto

Índice del capítulo

2.1. Estado inicial del proyecto

Nayar Systems dispone de un servicio de llamadas de voz ip[31] implementado con Freeswitch[10]. Se emplea un proveedor de telefonía que proporciona un primario voz IP para que estas llamadas puedan salir a las redes de telefonía habituales en caso de ser necesario, para poder contactar con la telealarma (en adelante máquina) que viene incorporada con una tarjeta SIM.

Respecto al proyecto correspondiente a la parte asíncrona (proyecto Dejavu), al iniciarse el mismo, 72horas dispone de un servicio de test para detectar si una máquina tiene conexión. Esta prueba se realiza llamando de forma automática o manual a la máquina y se detecta el tipo de tono que se efectúa durante esta llamada. Si el tono es el correspondiente al de marcado, se cuelga y se reporta el correcto funcionamiento de la máquina, si se detecta el de comunicar, se cuelga la llamada y se reporta que la línea no está disponible. En el caso que no poder detectarse ninguno de estos dos tonos, se graba hasta 20 segundos de audio y al expirar, se corta la llamada y se analiza el audio para saber si la línea funciona o no. En el caso anterior, se accede al servicio *speech* de Google [7], el cual responde con la transcripción de dicha grabación que, con ella se detecta lo que ha sucedido.

Respecto a la parte síncrona del proyecto (proyecto Freeswitch), no disponen de ningún servicio parecido y por ello pretenden investigar la viabilidad del mismo.

2.2. Funcionalidad

Como comentamos anteriormente, el proyecto consiste en detectar y reconocer audios y palabras, un objetivo para cada parte del proyecto formativo, Dejavu y Freeswitch.

2.2.1. Proyecto Dejavu

Para a este subproyecto, se espera modificar y añadir comportamiento al servicio de test de conexión que ya posee la empresa, como comentamos en el anterior apartado 2.1.

El servicio, antes de solicitar la transcripción del audio grabado, ha de consultar en una base de datos si el audio ya fue reconocido anteriormente, para que, en caso afirmativo, devuelva el mismo resultado que se devolvió en aquel momento. En caso negativo, solicitará la transcripción a Google y almacenará su resultado en su base de datos. De esta forma, Nayar Systems dejará de depender tanto del servicio *speech* de Google, que empezaría a ser de pago en caso que la empresa solicite más transcripciones de las estipuladas. De esta forma se sortea el límite impuesto por Google, y se puede utilizar este servicio de test sin grandes repercusiones económicas.

2.2.2. Proyecto Freeswitch

Respecto a este subproyecto se plantea experimentar con el reconocimiento de palabras durante una llamada de voz IP con Freeswitch empleando la tecnología de CMUSphinx. Se espera que, al detectar ciertas palabras clave, el servicio realice diferentes acciones, que se decidirán a lo largo de la implementación de esta fase.

2.3. Alcance

El alcance del proyecto también lo debemos dividir en los dos subproyectos de las que consta este proyecto formativo.

2.3.0.1. Proyecto Dejavu

Los requisitos de este servicio estipulan que se implemente una base de datos en PostgreSQL que, junto con la librería *Dejavu*, permita reconocer los audios analizados anteriormente. Los

que no sean reconocidos por *Dejavu* se solicitarán al servicio de transcripción de audios de Google para almacenar su resultado.

2.3.0.2. Proyecto Freeswitch

Como alcance de esta parte del proyecto, se espera experimentar con Freeswitch y CMUSphinx y realizar una aplicación interactiva como máximo con cuatro palabras clave y sus respectivas acciones.

2.4. Restricciones

Las restricciones del proyecto, que determina el *product owner*, son las siguientes:

- Para el proyecto *Dejavu*:
  - El lenguaje de programación del servicio debe ser Python.
  - Se ofrezca el servicio en un contenedor (Docker).
  - Que el servicio utilice una base de datos en PostgreSQL ubicada en otro contenedor.
- Para el proyecto Freeswitch:
  - Los costes de mantenimiento del sistema deben ser mínimos.
  - Las palabras se deben reconocer con una efectividad del 80 % como mínimo.
  - Se debe poder realizar al menos tres acciones diferentes.
  - El código que se implemente para realizar esta funcionalidad debe ser escrito en el lenguaje de programación Go.

Capítulo 3

Tecnologías y herramientas

Índice del capítulo

3.1. Lenguajes

3.1.1. Python

Python es un lenguaje de programación interpretado que apareció por primera vez en 1991, diseñado por Guido van Rossum. Este lenguaje permite la programación multiparadigma, lo cual permite optar por distintos estilos de programación. [24]

Este lenguaje se ha empleado para el proyecto asíncrono, debido a que la librería *Dejavu* está escrita en este lenguaje y se necesita un script en su mismo lenguaje para darle uso. Además, permite la conexión a Nexus a través de un socket con la librería pynexus.

3.1.2. Go

Go es un lenguaje de programación cuya sintaxis recuerda mucho al lenguaje de programación C. Fue desarrollado por Google y apareció en 2009. Go es un lenguaje de programación compilado, concurrente, imperativo, estructurado y, pese a estar diseñado principalmente para la administración de sistemas, posee un recolector de basura, cosa que le da una gran potencia.[21]

Este lenguaje se ha empleado para la parte síncrona del proyecto donde el *product owner* exigía el uso de este lenguaje en su programación.

3.2. Frameworks y Librerías

3.2.1. Nexus

Nexus [16] es un sistema de llamadas remotas distribuido de código abierto y desarrollado por José Luis Aracil Gómez del Campo para la empresa Nayar Systems. Dicho sistema, escrito en Go, permite la comunicación de microservicios ubicados en distintos sitios.

En el proyecto se emplea para el proceso Dejavu, el cual recibe y transmite mensajes utilizando Nexus.

3.2.2. *Dejavu*

Dejavu es una librería de reconocimiento de patrones de audio, diseñada por el usuario Worldveil de GitHub [33] en la cual, la comunidad de esta página también ha colaborado en su mejora y crecimiento. El proyecto como tal comenzó en septiembre de 2014 y su último aporte se realizó en enero de 2017 tras haber estado "inactivo" desde abril de 2015, cuando presuponemos se completó el proyecto.

Esta librería ofrece un método de reconocer canciones o audios realizando comparativas con otras guardadas en una base de datos, en principio MySQL aunque otros usuarios de GitHub cambiaron ligeramente el código para adaptarlo a PostgreSQL.

3.2.3. Freeswitch

Freeswitch[10] es una herramienta de telefonía que permite interconectar protocolos usando audio, vídeo, texto o cualquier otro tipo de comunicación. Su primera aparición fue en 2006, y desde entonces se ha hecho popular en el mundo de las telecomunicaciones como la plataforma de comunicación de código libre más grande del mundo.

En el proyecto se ha empleado para el subproyecto Freeswitch donde se necesitaba un método de comunicación entre dos partes.

3.2.4. CMUSphinx

CMUSphinx es el término general para determinar un conjunto de proyectos de reconocimiento de voz desarrollado en la Universidad de Carnegie Mellon. Incluye varios programas de reconocimiento de voz y un programa de entrenamiento de modelos acústicos.

En el proyecto Freeswitch se ha empleado el programa PocketSphinx.

PocketSphinx es una versión de Sphinx para sistemas embebidos (móviles, RaspberryPi, etc). Se emplea para el reconocimiento de voz. Requiere menos recursos que su antecesor Sphinx[18].

Para el proyecto Freeswitch se ha utilizado este programa porque la propia librería de Freeswitch daba soporte y facilidad de uso de la misma.

3.3. Control de versiones

3.3.1. Git

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

Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Proyectos de mucha relevancia usan Git, en particular, el grupo de programación del núcleo Linux [20].

En el proyecto se ha utilizado para llevar un control de versiones de todo el código de ambos subproyectos.

3.4. Bases de datos

3.4.1. MySQL

MySQL[8] es un sistema de gestión de base de datos relacional creada por Oracle Corporation[22]. Al tener licencia de código abierto para los proyectos de este mismo tipo, y su gran potencia, representa una buena utilidad para almacenar las canciones que la librería Dejavu muestrea.

En el proyecto solo se empleó para realizar las pruebas oportunas sobre la librería Dejavu. Una vez probado su correcto funcionamiento y utilidad se pasó al SGBD PostgreSQL.

3.4.2. PostgreSQL

Al igual que MySQL, PostgreSQL es una base de datos relacional, con la diferencia que es totalmente de código abierto.

Se ha empleado esta base de datos en el proyecto Dejavu porque la empresa la utiliza para almacenar más información, y al tenerla ya implementada resultaba más sencillo crear una nueva tabla en la misma en lugar de utilizar otro SGBD.

3.5. Despliegue

3.5.1. Kubernetes

Kubernetes[6] es un sistema de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores Docker. Creado por Google y lanzado al público el 7 de junio de 2014.

En la empresa utilizan Kubernetes para la administración de todos sus contenedores Docker, para tener un administrador de los mismos y poder tener un control sobre ellos.

En el proyecto se ha usado para albergar el contenedor Docker del proyecto Dejavu.

3.5.2. Google Cloud

Google Cloud es una plataforma que reúne todas las funcionalidades que ofrece Google, únicamente se emplea una máquina virtual con Debian como sistema operativo para correr Freeswitch en dicho Servidor. Tiene un coste mensual dependiendo de los recursos que se destinene para ese espacio virtual reservado, que se detallan más adelante en el apartado 4.3.

3.6. Entornos de desarrollo

3.6.1. Editor de textos Vi

Vi[29] es un editor de textos que ofrece una gran potencia y funcionalidad a la edición del texto, por lo que a la hora de escribir código es realmente rápido.

Este editor de textos se utilizó mediante la consola de Linux en los ordenadores de prácticas y el servidor de Google Cloud antes de utilizar Visual Studio Code. Se empleó para el desarrollo del proyecto Dejavu y la implementación de Freeswitch, así como la edición de sus *dialplan*, es decir, los ficheros de configuración del flujo de la llamada.

3.6.2. Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft. Incluye soporte para *debugging*, control de GIT integrado, entre otras características que lo convierten en un IDE gratuito muy completo. Además, tiene gran cantidad de opciones de personalización así como de plugins para aumentar su funcionalidad[30]. Contamos con la licencia del software de estudiante por lo que tenemos la aplicación extendida de forma gratuita.

En el proyecto se ha utilizado como entorno de desarrollo para el código en el lenguaje de programación Go.

3.7. Gestión de proyectos

3.7.1. Trello

Trello es un software de administración de proyectos con interfaz web, cliente para iOS y Android para organizar proyectos.

Emplea el sistema *Kanban* para el registro de actividades con tarjetas virtuales. Permite agregar listas, adjuntar archivos, etiquetar eventos, agregar comentarios y compartir tableros [28]. En el proyecto, se ha utilizado para llevar un control personal del avance del proyecto.

3.7.2. Google Sheets

Google Sheets es una herramienta de Google para la edición y visualización de hojas de cálculo en tiempo real almacenadas en la nube.

Para gestionar las horas y las labores que se han ido realizando en ellas se ha empleado Google Sheets que con su potencia en cálculos y operaciones con horas y días se ha llevado un estricto

control sobre la duración de los sprints y las horas diarias dedicadas tanto al trabajo presencial en la empresa, como el trabajo no presencial sobre los informes quincenales y preparación de este trabajo.

También se ha empleado para el seguimiento del proyecto Dejavu tanto en integración como en producción, cuyos resultados podrá observarse en el apartado 4.4.

3.7.3. Google Docs

Google Docs es una herramienta de Google, que permite la edición y visualización de textos colaborativa en tiempo real, como Google Sheets, almacenada en la nube.

En Google Docs se ha llevado un detallado diario sobre la estancia en prácticas (qué se ha hecho en el día, que se pretende realizar, qué problemas han surgido, etc). Por lo que ha resultado útil tanto a la hora de realizar el seguimiento del trabajo como para reflexionar sobre qué se ha estado implementando o investigando.

3.8. Diseño

3.8.1. Vertabelo

Vertabelo es una herramienta web, que permite realizar el diseño físico y lógico de bases de datos, adaptándose a los diferentes SGBD.

En el proyecto se ha empleado para diseñar la base de datos del proyecto Dejavu, y su esquema conceptual para plasmarlo en este proyecto.

3.8.2. Draw.io

Draw.io es una herramienta de diseño en la nube que, gracias a su sencillez de uso, capacidad de respuestas rápidas y gran cantidad de funcionalidades, la convierten una buena opción para la creación de diagramas o mockups. Además, como es posible integrarlo con Google Drive o GitHub, facilita la tarea de organización de los ficheros de diseño dentro del proyecto.

En el proyecto se ha utilizado para realizar los diagramas necesarios para su documentación.

Capítulo 4

Planificación del proyecto

Índice del capítulo

4.1. Metodología

En este apartado, se explicará la metodología seguida en Nayar Systems a la vez que se comparará con la metodología Scrum [25] para ver sus similitudes y diferencias.

En Nayar Systems, los proyectos se organizan por líneas de negocio, que en este momento son cuatro: 72horas, Advertisim, Net4machines y GSR. Cada uno de los proyectos tiene:

- *Project manager*: Se encarga de coordinar el proyecto y de responder de éste de cara al cliente y a la empresa.
- *Product owner*: Es el que decide en última instancia qué se debe desarrollar.
- *Tech leader*: Es el que coordina el equipo técnico y diseña la arquitectura del sistema.
- *TeamWork* o Desarrolladores: Lo componen los desarrolladores que implementan la funcionalidad.

En ocasiones los roles de *project manager* y *product owner* los interpreta la misma persona. El *tech leader* siempre es a la vez desarrollador y los equipos suelen estar compuestos por 4 o 5 personas en total.

Respecto a la selección y reparto de las tareas, el *product owner* y el *project manager* deciden qué tareas se deben realizar, consultando al *tech leader* en el caso que se necesite esclarecer la dificultad de las mismas.

Una vez decididas, siguiendo la metodología Scrum, se valoran para saber cuánto costaría desarrollarlas. En algunos casos se utiliza el llamado *planning Poker*. El *planning Poker* es una forma de "votación", en la que todos los miembros del *teamwork* levantan una carta con un coste temporal, y se debate porqué consideran ese coste para esa tarea, con el fin de alcanzar un consenso antes de la asignación del coste definitivo.

Una vez valoradas, el *project manager* y el *product owner* las priorizan, y se establece un esprint de dos semanas cogiendo las tareas en orden hasta completar la cantidad de trabajo que se calcula que puede realizar el equipo.

La diferencia que encontramos respecto a la metodología Scrum es que lo único que entra al *backlog* son historias de usuario, es decir, tareas que aportan valor al usuario. Sin embargo hay otras tareas (bugs, tareas de mantenimiento u otras tareas que no tienen un valor directo para el usuario) que a pesar de ser necesarias, no se valoran ni se incluyen en los sprints. Éstas tareas se plasman en un *Kanban*, similar a un *backlog* de Scrum.

Las tareas se van introduciendo tal como van apareciendo, se priorizan periódicamente por el *project manager* y el *product owner*, y los desarrolladores las resuelven en cualquier momento, normalmente si acaban las tareas del sprint antes de tiempo. Por esta razón se suele estimar a la baja la capacidad de trabajo de los desarrolladores, para tener un margen en el sprint para resolver tareas del *Kanban*.

Para gestionar todo esto se emplea JIRA[1], que permite tener tanto tablas Scrum, con su *backlog*, *sprints*, estimaciones, asignaciones de tareas, orden y progreso, como tablas *Kanban* con funcionalidades similares.

4.2. Planificación

La planificación del proyecto se ha dividido en 2 partes o subproyectos, con el primero (Dejavu) con cuatro fases: investigación, planificación, desarrollo y producción; el segundo (Freeswitch) con tres fases ya que es un proyecto experimental: investigación, planificación y experimentación.

Se ha decidido realizarlo de esta forma porque para la empresa es necesario mejorar el servicio *speech* que tienen para incluir la detección de audios con Dejavu. El proyecto Freeswitch se realizará al terminar el proyecto Dejavu, ya que, lo que se pretende con Freeswitch es experimentar la viabilidad de poseer un servicio con esas características.

Por ello se ha decidido dividir las etapas mencionadas anteriormente en diferentes *sprints* siguiendo la metodología Scrum, que comentamos anteriormente, y un seguimiento semanal (5 días laborables) al finalizar cada esprint.

[1] `https://es.atlassian.com/software/jira`

En total se planifica realizar 11 *sprints* desde el inicio de las prácticas hasta la finalización de las mismas el 3 de mayo de 2018.

Más adelante, se puede ver los diagramas de Gantt para los proyectos de los proyectos Dejavu 4.1 y Freeswitch 4.2 que se utilizaron para un primer reparto de tareas y planificación del flujo de trabajo durante las prácticas, como se puede comprobar, ambos proyectos terminan el día 3 de mayo, fecha que concluye la estancia en prácticas.

4.3. Estimación de recursos y costes del proyecto

El coste del desarrollo de los proyecto Dejavu y Freeswitch no tendremos en cuenta el coste de recursos que ya emplea la empresa los cuales tienen muy poca repercusión económica por lo que no los tendremos en cuenta en los cálculos (ordenadores, auriculares, teclado, internet,...). Para el proyecto Dejavu, tan solo tendremos en cuenta el coste por horas del programador. Se estima el coste del programador a 10€ la hora y la cantidad de horas invertidas en el proyecto tendremos en cuenta hasta que se pasó a producción contando los días del seguimiento de integración a 1h por día (el seguimiento del proyecto se puede verlo en el capítulo 7) en total 77h de implementación más 16h de seguimiento según el seguimiento de horas del programador:

$$Precio_{\text{Dejavu}} = \left(\frac{10\,€}{h} \times (77\,h + 16\,h)\right) = 930\,€$$

Para el coste del proyecto Freeswitch, con el mismo coste por hora del programador, tendremos en cuenta el resto de horas desde el inicio del proyecto (cuando el proyecto Dejavu pasó a integración) contando las horas de implementación mientras el seguimiento se estaba realizando a 5h diarias, en resumen 300h, menos las ya empleadas en el proyecto Dejavu. Además para este proyecto de ha tenido que contratar el servicio de Google Cloud que empezó a utilizarse en marzo, y, como la estancia en prácticas finalizó el día 3 de mayo, este servicio se contrató por dos meses a un precio de 14.56$ que en euros es 12,397820€(comprobado el 21/05/2018 10:59:28):

$$Precio_{\text{Freeswitch}} = \left(\left(\frac{10\,€}{h} \times (300\,h - 93\,h)\right) + \left(3\,mes \times \frac{12.39\,€}{mes}\right)\right) = 2107.17\,€$$

Ya hechos los cálculos de ambos proyectos, el coste total de la estancia equivale a:

$$Total_{\text{Estancia}} = 930\,€ + 2107.17\,€ = 3037.17\,€$$

4.4. Seguimiento del proyecto

El seguimiento del proyecto se ha llevado durante todo el proceso de investigación e implementación. Asimismo, respecto al proyecto Dejavu se ha realizado un seguimiento del desarrollo tanto en integración como en producción. En el capítulo 7 se puede ver cómo se ha desarrollado el proyecto y, respecto al seguimiento y validación de ambos proyectos, en el apartado 7.12 se muestra el progreso de ambos.

4.4.1. Desviaciones de la planificación

En todos los proyectos, la duración de las tareas establecidas en la planificación pueden modificarse debido a muchos factores.

En este proyecto se ha llevado un estricto control horario de cada labor, dedicándole las horas que se estableció en el apartado 4.2, cumpliendo con las fechas de entrega del proyecto Dejavu y alcanzando el objetivo del proyecto Freeswitch satisfactoriamente.

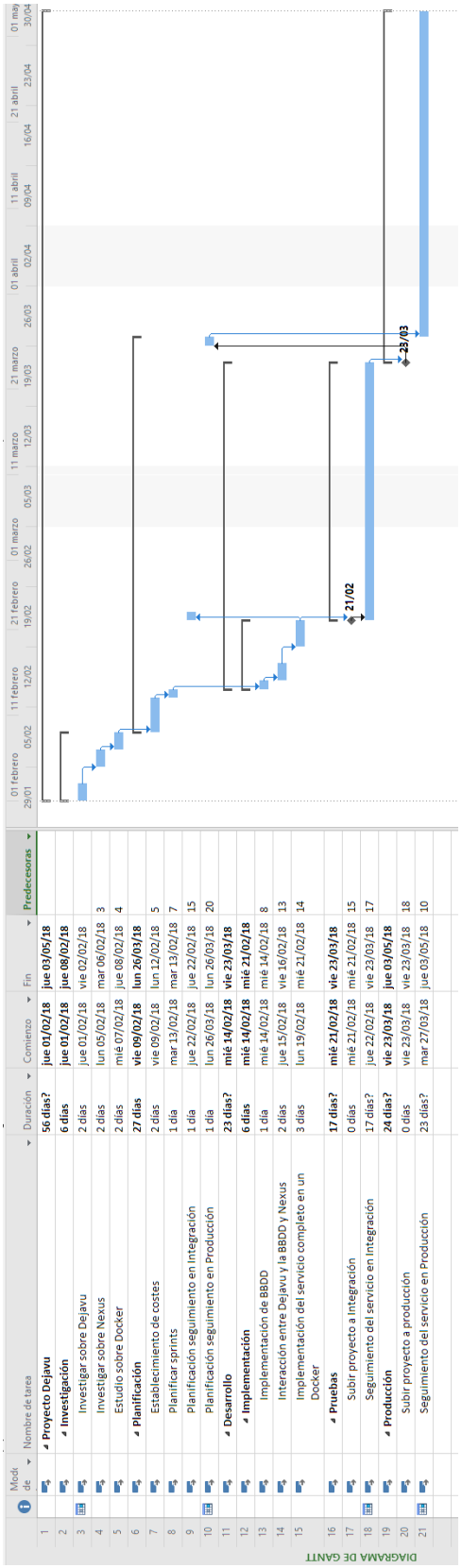

Figura 4.1: Diagrama de Gantt orientativo del proyecto Dejavu.

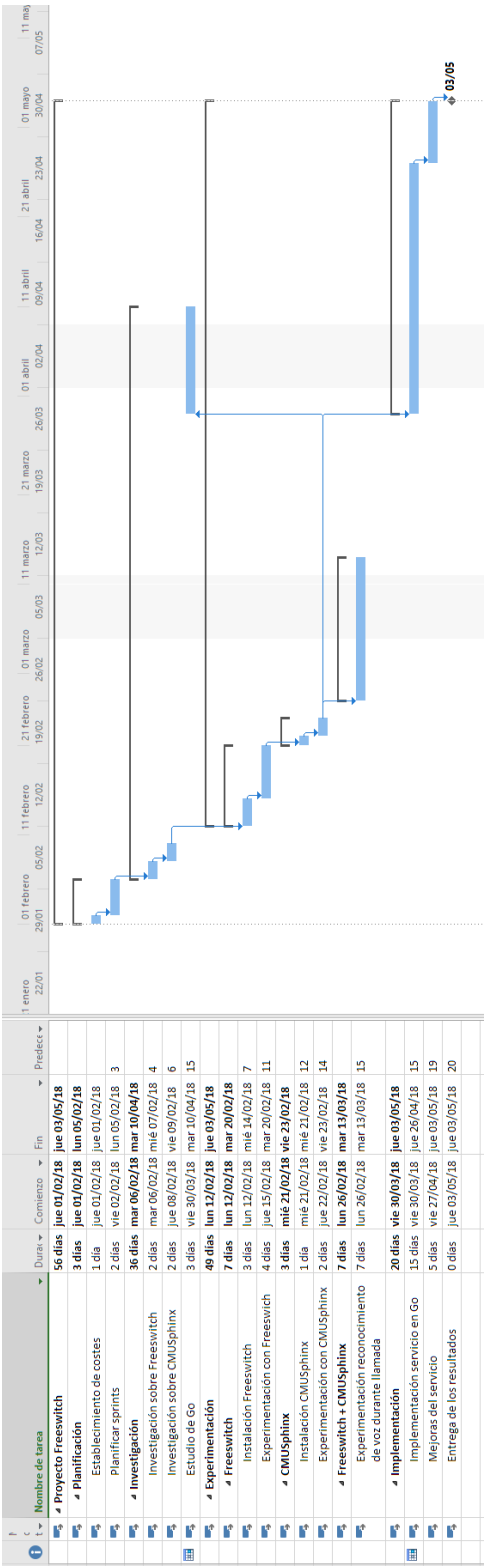

Figura 4.2: Diagrama de Gantt orientativo del proyecto Freeswitch.

Capítulo 5

Definición de requisitos

Índice del capítulo

5.1. Requisitos de los proyectos

Las restricciones del proyecto, que determina el *product owner*, son las siguientes:

- Para el proyecto *Dejavu*:
  - El servicio debe averiguar si el audio recibido es correcto.
  - El servicio debe reconocer audios grabados previamente.

- Si el servicio no reconoce el audio, solicitará la transcripción a Google.
- El servicio debe almacenar el audio junto con la transcripción ofrecida por Google.
- El servicio debe devolver la transcripción ya sea recibida por Google o reconocida en la base de datos.

- Para el proyecto Freeswitch: Las palabras clave que posteriormente se mencionan, no se establecieron en la etapa de planificación del proyecto, en su lugar, se especificaron después de la experimentación con Freeswitch y PocketSphinx.

  - El servicio debe ofrecer una selección de acciones, que si no se han escuchado bien, el usuario pueda volver a escucharlas.
  - Al reconocer una palabra clave (p.e. eco), debe redirigir al servicio *echo*.
  - Al reconocer otra palabra clave (p.e. llamar), debe redirigir la llamada a un teleoperador.
  - Al reconocer otra palabra clave (p.e. números), debe redirigir al servicio de detección de números.
  - El porcentaje de acierto de las palabras deberá ser superior del 80 %.

5.2. Diagrama de casos de uso

5.2.1. Proyecto Dejavu

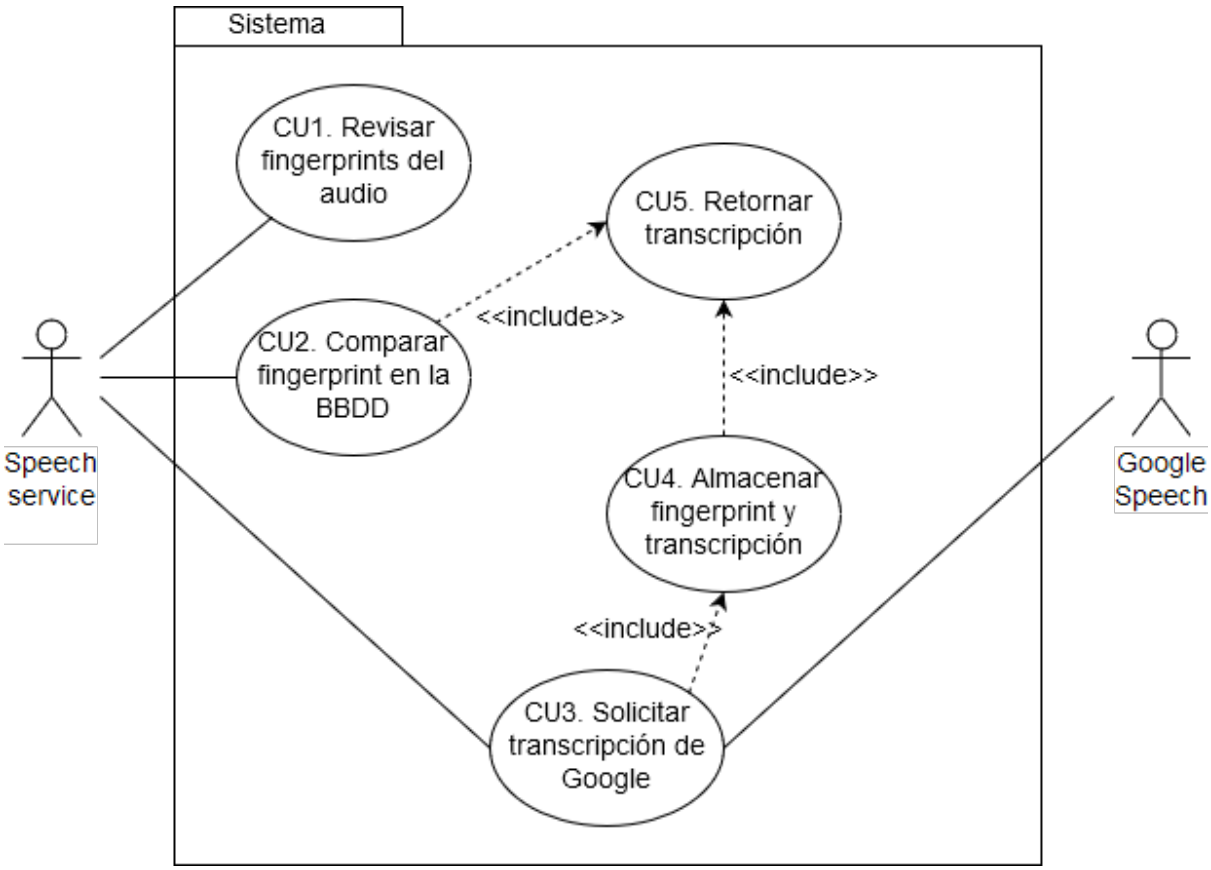

Figura 5.1: Diagrama de casos de uso del proyecto Dejavu.

5.2.2. Proyecto Freeswitch

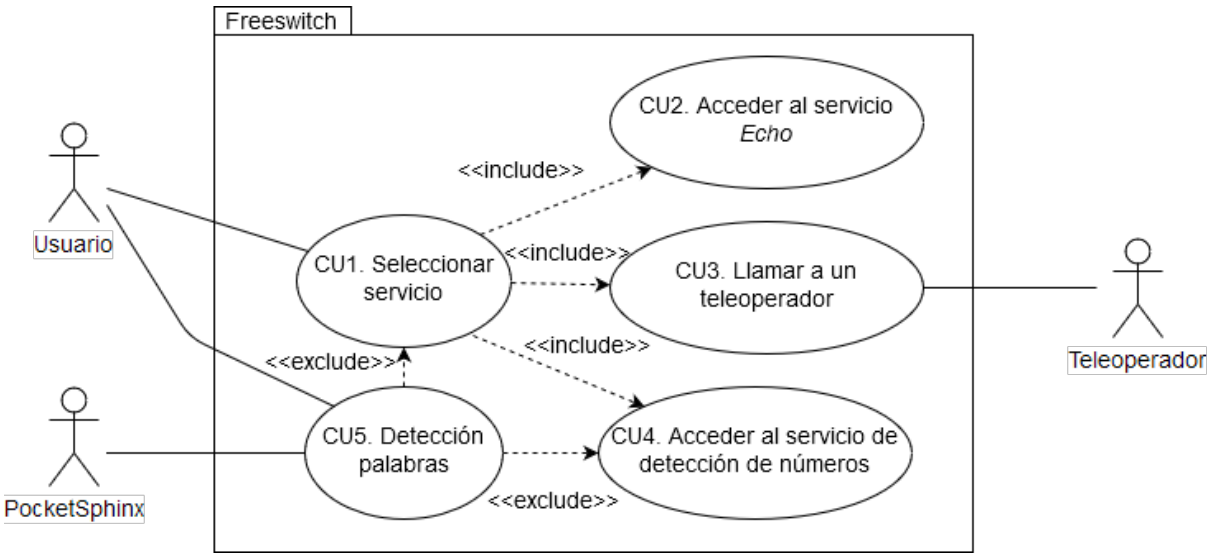

Figura 5.2: Diagrama de casos de uso del proyecto Freeswitch.

5.3. Descripción de casos de uso

5.3.1. Proyecto Dejavu

5.3.1.1. CU1 - Revisar *fingerprints* del audio

Especificación del caso de uso	
ID	CU1
Nombre	Revisar *fingerprints* del audio
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El sistema ha de analizar los fingerprints del audio que recibe.
Alcance	El caso cubre desde el momento que se recibe el audio hasta que se analiza la cantidad de *fingerprints*.
Actor principal	Speech service
Actores secundarios	-
Condición de final con éxito	El sistema ha de retornar al servicio el número de *fingerprints* que se han analizado en el audio.
Condición de final con error	El número que se devuelva será 0.
Trigger	Recibir una petición de análisis de audio.
Secuencia normal	**Acción**
1	El sistema recibe un audio para transcribir.
2	El sistema se asegura que los parámetros de la petición son correctos.
3	El sistema analiza el audio para contar los *fingerprints*.
4	El sistema devuelve el número de *fingerprints* detectados.

Tabla 5.1: CU1 - Revisar *fingerprints* del audio.

5.3.1.2. CU2 - Comparar *fingerprints* en la BBDD

Especificación del caso de uso	
ID	CU2
Nombre	Comparar *fingerprints* en la BBDD
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El sistema ha de comprobar si hay coincidencia entre el audio recibido y los audios grabados previamente, comparando entre sus *fingerprints*.
Alcance	Desde que se ha analizado el número de *fingerprints*, siendo éste mayor que 0 hasta que se encuentra o no coincidencia en toda la base de datos.
Actor principal	Speech service
Actores secundarios	-
Condición de final con éxito	Se encuentra coincidencia con un audio guardado en la base de datos y se prepara el envío de su transcripción como se especifica en el CU5.
Condición de final con error	No se encuentra coincidencia en la base de datos, y se prepara la petición al servicio de transcripción de textos de Google como se especifica en el CU3.
Trigger	Confirmación de más de 0 *fingerprints* en el audio.
Secuencia normal	**Acción**
1	El servicio recoge los *fingerprints* del audio recibido.
2	Compara sus *fingerprints* entre todos los *fingerprints* de la base de datos en conjunto.
3	En caso de encontrar coincidencia prepara su transcripción para ser devuelta. En caso negativo, prepara el audio para solicitar la transcripción a un servicio externo.

Tabla 5.2: CU2 - Comparar *fingerprints* en la BBDD.

5.3.1.3. CU3 - Solicitar transcripción de Google

Especificación del caso de uso	
ID	CU3
Nombre	Solicitar transcripción de Google
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El sistema debe permitir que en el caso de no encontrar coincidencia en la base de datos, se solicite la transcripción a Google.
Alcance	Desde que el audio no se ha encontrado en la base de datos hasta que el servicio de Google devuelve un resultado.
Actor principal	Speech service
Actores secundarios	Google speech
Condición de final con éxito	El audio se ha analizado correctamente y se obtiene su transcripción.
Condición de final con error	Se recibe una excepción por parte del servicio de Google.
Trigger	No se ha encontrado coincidencia del audio en la base de datos.
Secuencia normal	**Acción**
1	Se envía una petición al servicio *speech* de Google.
2	Se recibe y analiza la respuesta.
3	Se prepara el audio con la transcripción recibida para el CU4.

Tabla 5.3: CU3 - Solicitar transcripción de Google.

5.3.1.4. CU4 - Almacenar *fingerprint* y transcripción

Especificación del caso de uso	
ID	CU4
Nombre	Almacenar *fingerprint* y transcripción
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El servicio ha de permitir almacenar en la base de datos los *fingerprints* del audio junto con su transcripción.
Alcance	Desde que se recibe la transcripción hasta que se almacena el audio y su transcripción en la base de datos.
Actor principal	Speech service
Actores secundarios	-
Condición de final con éxito	Los *fingerprints* del audio y su transcripción se han almacenado en la base de datos y se prepara para enviar el resultado para el CU5.
Condición de final con error	No se ha conseguido almacenar en la base de datos los *fingerprints* o la transcripción, y se devuelve un error.
Trigger	Recibir confirmación de transcripción recibida correctamente.
Secuencia normal	**Acción**
1	Se almacena los datos de la transcripción.
2	Se almacena los datos de los *fingerprints*.
3	Se prepara la transcripción para abordar el CU5.

Tabla 5.4: CU4 - Almacenar *fingerprint* y transcripción.

5.3.1.5. CU5 - Retornar transcripción

Especificación del caso de uso	
ID	CU5
Nombre	Retornar transcripción
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El sistema ha de retornar la transcripción al servicio que lo solicitó.
Alcance	Desde que se asegura la transcripción del audio hasta que se envía al solicitante.
Actor principal	Speech service
Actores secundarios	-
Condición de final con éxito	La transcripción se envía con éxito.
Condición de final con error	La transcripción no se envía correctamente.
Trigger	La confirmación de tener la transcripción.
Secuencia normal	**Acción**
1	Se recibe la transcripción.
2	Se envía la transcripción al servicio solicitante.

Tabla 5.5: CU5 - Retornar transcripción.

5.3.2. Proyecto Freeswitch

5.3.2.1. CU1 - Seleccionar servicio

Especificación del caso de uso	
ID	CU1
Nombre	Seleccionar servicio
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El servicio debe comunicar al usuario las acciones que puede realizar que serán los CU2, CU3, CU4, las cuales las deberá elegir por comando de voz.
Alcance	Desde que el usuario realiza la llamada hasta que selecciona uno de los tres servicios.
Actor principal	Usuario
Actores secundarios	-
Condición de final con éxito	Redirige a un servicio en concreto.
Condición de final con error	Repite las opciones al usuario.
Trigger	El usuario llama al servicio.
Secuencia normal	**Acción**
1	El usuario llama al servicio.
2	El servicio le comunica las opciones que debe elegir.
3	El usuario elige una opción mencionando una de las palabras clave.
4	Empleando el CU5, se reconoce la palabra y, se redirige al servicio seleccionado (CU2, CU3 o CU4) o se repite las opciones.

Tabla 5.6: CU1 - Seleccionar servicio.

5.3.2.2. CU2 - Acceder al servicio *Echo*

Especificación del caso de uso	
ID	CU2
Nombre	Acceder al servicio *Echo*
Versión	1.0
Fecha actualización	10/2/2018
Descripción	Se activa el servicio *echo*.
Alcance	Desde que el usuario menciona la palabra clave hasta que el servicio *echo* concluye.
Actor principal	Usuario
Actores secundarios	-
Condición final con éxito	El servicio *echo* se concluye satisfactoriamente.
Trigger	El usuario menciona una palabra clave para el servicio.
Secuencia normal	**Acción**
1	El usuario menciona la palabra clave.
2	Se activa el servicio *echo*.
3	El servicio *echo* concluye.

Tabla 5.7: CU2 - Acceder al servicio *Echo*.

5.3.2.3. CU3 - Llamar a un teleoperador

Especificación del caso de uso	
ID	CU3
Nombre	Llamar a un teleoperador
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El servicio ha de permitir redirigir la llamada a un teleoperador.
Alcance	Desde que el usuario menciona una palabra clave hasta que el usuario o el teleoperador concluyan la llamada.
Actor principal	Usuario
Actores secundarios	Teleoperador
Condición final con éxito	El usuario o el teleoperador concluye la llamada.
Condición final con error	El teleoperador no atiende la llamada.
Trigger	El usuario menciona la palabra clave.
Secuencia normal	**Acción**
1	Al mencionar la palabra clave se redirige la llamada a un teleoperador.
2	El teleoperador puede descolgar o no la llamada.
3	El Usuario o el teleoperador cuelgan la llamada.

Tabla 5.8: CU3 - Llamar a un teleoperador.

5.3.2.4. CU4 - Acceder al servicio de detección de números

Especificación del caso de uso	
ID	CU4
Nombre	Acceder al servicio de detección de números
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El servicio debe repetir al usuario los números que menciona del 0 al 9.
Alcance	Desde que el usuario menciona la palabra clave hasta que éste cuelga la llamada.
Actor principal	Usuario
Actores secundarios	-
Condición final con éxito	El usuario termina la llamada.
Trigger	El usuario menciona la palabra clave.
Secuencia normal	**Acción**
1	El usuario pronuncia un número.
2	El servicio emite un sonido y pronuncia el número detectado.
3	Las acciones 1 y 2 se repiten en este orden hasta que usuario cuelga la llamada.

Tabla 5.9: CU4 - Acceder al servicio de detección de números.

5.3.2.5. CU5 - Detección de palabras

Especificación del caso de uso	
ID	CU5
Nombre	Detección de palabras
Versión	1.0
Fecha actualización	10/2/2018
Descripción	El servicio debe permitir que PocketSphinx detecte las palabras que se pronuncian durante los servicios descritos en los CU1 y CU4.
Alcance	Siempre que el usuario diga una palabra hasta que ésta se detecte por PocketSphinx, consecutivamente dependiendo del servicio.
Actor principal	Usuario.
Actores secundarios	PocketSphinx.
Condición final con éxito	La palabra que se ha detectado es la que se ha mencionado.
Trigger	El usuario comienza a hablar.
Secuencia normal	**Acción**
1	El usuario comienza a hablar.
2	Menciona una palabra o número clave.
2	PocketSphinx detecta la palabra y la comunica al servicio que lo está empleando.

Tabla 5.10: CU5 - Detección de palabras.

Capítulo 6

Análisis y diseño del sistema

Índice del capítulo

6.1. Análisis del sistema

El análisis de sistemas es una de las etapas de construcción de un sistema informático que consiste en mostrar la información actual y proponer en rasgos generales, la solución futura. Para la especificación de esta solución, se utilizan diferentes diagramas según la complejidad y necesidades específicas de cada proyecto.

6.2. Diseño de la arquitectura del sistema

Para este apartado se ha empleado diagramas de componentes UML en lugar de los típicos diagramas de clases, debido a que para ambos proyectos, proporciona más información el diseño de los componentes que las clases en sí además de lo que se desarrolla e implementa es mediante componentes.

6.2.1. Proyecto Dejavu

El sistema del proyecto Dejavu se interconecta entre dos partes: una aplicación que solicitará la transcripción de un audio el cual ella misma proporcionará mediante la librería Nexus y el servicio de transcripción de mensajes de Google.

El diagrama que representa dicha arquitectura lo se puede observar en la figura 6.1. Como se puede observar, los componentes Nexus, Dejavu speech Service y Base de datos están alojados dentro de Kubernetes, por lo tanto, son imágenes de contenedores las que se interconectan mediante conexiones socket. Para acceder al servicio de reconocimiento de voz de Google se emplea una conexión HTTP.

El método de transmisión de la información entre un cliente y Nexus no lo hemos tenido en cuenta puesto que es algo que no nos incumbe para el proyecto pero sí para conocer el funcionamiento del servicio, por ello lo hemos incluido como un componente externo.

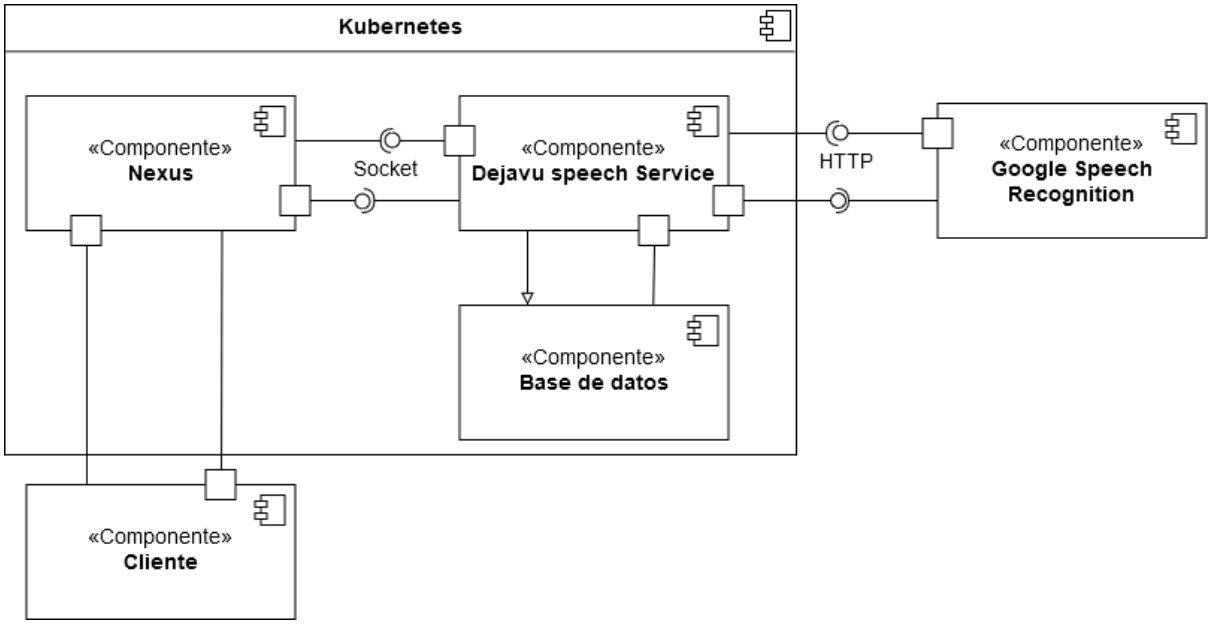

Figura 6.1: Arquitectura del proyecto Dejavu.

Se realiza un rediseño de la base de datos, porque la librería Dejavu original, estaba preparada para trabajar con una base de datos MySQL, cuyos tipo de datos varían ligeramente respecto a PostgreSQL 6.3. En la figura 6.2 se puede ver como sería el diseño en MySQL. Estos ligeros cambios se pueden observar entre ambas figuras. Además a la hora de editar la versión original, el editor se percató que el atributo "file_sha1" no era útil en esta nueva base de datos y se eliminó.

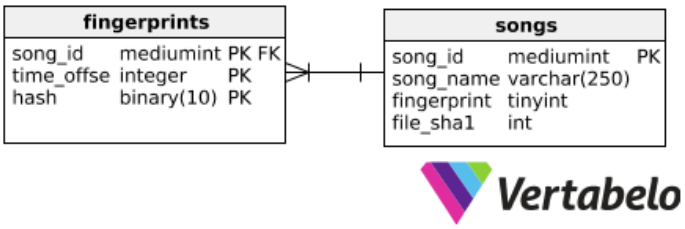

Figura 6.2: Diseño de la base de datos en MySQL.

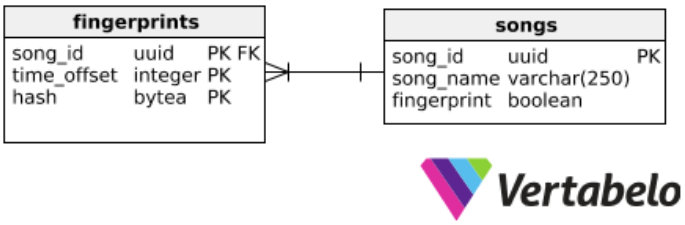

Figura 6.3: Diseño de la base de datos en PostgreSQL.

6.2.2. Proyecto Freeswitch

De forma general la figura 6.4 representa la arquitectura del proyecto Freeswitch. Básicamente, un cliente realiza una llamada telefónica a Freeswitch y este emplea PocketSphinx para reconocer palabras, asimismo, freeswitch puede redirigir la llamada a diferentes servicios que pueden encargarse de otro tipo de reconocimiento de voz.

Como se puede observar, el protocolo de comunicación entre el cliente SIP y Freeswitch es voz IP y la comunicación entre Freeswitch y el Servicio en el lenguaje de programación Go es mediante conexión Socket. Se observa que tanto el componente Freeswitch como el componente del Servicio en Go dependen del componente PocketSphinx, esto es, porque ambos componentes emplean la instancia de dicho servicio en el servidor para reconocer las palabras clave.

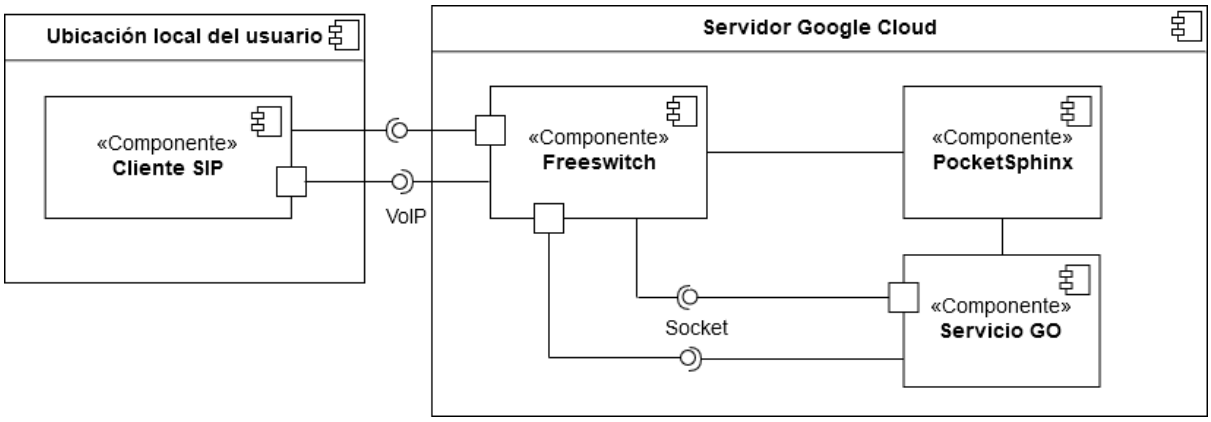

Figura 6.4: Arquitectura del sistema Freeswitch.

Capítulo 7

Implementación

Índice del capítulo

En este capítulo se detallara cómo ha ido desarrollándose el proyecto siguiendo una metodología similar a Scrum. Por ello cada apartado siguiente corresponderá a cada uno de los sprints realizados durante el proyecto, con una semana de duración cada uno.

7.1. Sprint 1

7.1.1. Planificación del sprint

Este sprint, al ser el primero, se pretende definir las necesidades que ha de cubrir el proyecto, así como estudiar las tecnologías principales para llevar a cabo el proyecto de reconocimiento de voz asíncronamente, que en adelante lo nombraremos como la librería que emplearemos: *Dejavu*.

7.1.2. Ejecución del sprint

7.1.2.1. Especificación del proyecto

Dadas las peticiones del *product owner*, se necesita implementar un servicio de reconocimiento de voz con el lenguaje de programación *Python*, el cual dependa lo mínimo del servicio *speech* de Google (del que ahora depende fuertemente) y recurra a otro tipo de servicio, en local a poder ser, para consultar si hay coincidencia entre el audio recién grabado y otros grabados con anterioridad.

Para ello el *product owner* propone el uso de la librería *Dejavu*, como base de datos el uso de PostgreSQL, y exige el uso de Python como lenguaje de programación, para mejorar el script que ya poseen, y el uso de la librería Nexus de su empresa.

7.1.2.2. Estudio del lenguajePython

Al haber desarrollado muy poco en Python, se necesita refrescar los conocimientos de este lenguaje, para ello se accedió a bibliografía sobre Python[1].

7.1.2.3. Estudio de la librería Dejavu

El *product owner* necesita del uso de la librería Dejavu, que se encuentra en el repositorio público del autor en GitHub[32].

Esta librería lo que ofrece es una comparativa entre los *acoustic fingerprints*[17] de un audio reciente con cada audio almacenado en una base de datos MySQL[8].

Como el autor lo implementa con una base de datos MySQL, nuestro objetivo es cambiarlo a PostgreSQL[9].

[1] *Tutorial de Python*[15] y *Python para todos*[14]

Se realizó una aplicación de prueba con lo que aportaba el autor, para comprobar su correcto funcionamiento con un Docker de Linux con MySQL.

7.1.2.4. Estudio de Nexus

Para comprender un poco mejor la interacción entre el servicio a implementar con la librería Nexus de la empresa, se estudia la conexión y transferencia de mensajes entre estos dos servicios.

7.2. Sprint 2

7.2.1. Planificación del sprint

Durante este esprint se pretende cambiar la base de datos que implementa la librería Dejavu, por PostgreSQL e integrarlo con las peticiones de la librería Nexus.

7.2.2. Ejecución del sprint

7.2.2.1. Implementación de librería Dejavu con PostgreSQL

Como comentamos en el esprint anterior, realizamos pruebas sobre la librería Dejavu con un Docker de Linux con MySQL, pues para este se cambió la base de datos del Docker a PostgreSQL y se obtuvo una nueva versión (*branch*) aportada por otro usuario del repositorio de la librería Dejavu con los cambios en el código oportunos para emplear esta base de datos. El resultado fue un éxito.

7.2.2.2. Integración de librería Dejavu con Nexus

Para esta labor, se editó el script correspondiente para que, antes de solicitar la transcripción del audio a Google, comprobase si ésta ya tenía una semejanza en la base de datos.

7.3. Sprint 3

7.3.1. Planificación del sprint

Durante este sprint se estudió cómo crear un Docker propio para almacenar el servicio del proyecto *Dejavu* con el fin de tener un mejor control sobre el mismo gracias a Kubernetes[6], donde la empresa administra todos sus servicios como contenedores.

7.3.2. Ejecución del sprint

7.3.2.1. Almacenar audios con el mismo comunicado

Por necesidades del entorno de trabajo, es posible que hayan casos en que audios comunicando el mismo mensaje tengan *acoustic fingerprints* diferentes, por ello se ha de almacenar cada audio que no se haya encontrado coincidencia.

7.3.2.2. Estudio de Docker

Se estudió la tecnología de contenedores Docker: cómo crear, cómo editar imágenes Docker y contenedores, toda su terminología y metodología de uso. El objetivo es crear un contenedor del servicio óptimo en tamaño y en funcionalidad.

7.3.2.3. Crear un contenedor del servicio y subirlo a Integración

Al estar programado en Python, el contenedor necesita muchas librerías de Python para poder hacer funcionar el servicio debido a las fuertes dependencias de la librería Dejavu; como la empresa ya posee un Docker de PostgreSQL no fue necesario integrarlo junto con este servicio.

Al conseguir una imagen de Docker del servicio consistente, y su tamaño reducido al mínimo, se procedió a subirlo al servidor de integración, donde se realizarán pruebas sobre el servicio para asegurar que cumple su funcionalidad y es estable.

7.4. Sprint 4

7.4.1. Planificación del sprint

Respecto a este sprint se pretende organizar un sistema de seguimiento sobre el servicio en Integración y comenzar a documentarse sobre Freeswitch y CMUSpchinx.

7.4.2. Ejecución del sprint

7.4.2.1. Planificar seguimiento del servicio en Integración

Para realizar el seguimiento del proyecto *Dejavu* en el servidor de integración se desea probar los mensajes de no disponibilidad de diferentes operadoras. Para ello, se procede a realizar 3 llamadas a 10 dispositivos de los usuarios con red móvil en el despacho, incluyendo dos máquinas

que se emplean para realizar pruebas. En la Figura 7.2 se muestra la cantidad de dispositivos por operador a los que podemos llamar para realizar pruebas. Asimismo, se puede ver la probabilidad con la que un operador puede ser elegido en cada prueba.

En el apartado 7.12.1.2 se muestran más detalles sobre este seguimiento en el servidor de integración.

7.4.2.2. Estudiar Freeswitch

Al ser un campo nuevo, se tiene que estudiar muy bien esta herramienta. Freeswitch es una herramienta que ofrece un medio de comunicación entre varios clientes SIP[11], permitiendo un comportamiento concreto determinando en sus *dialplan* qué realizar en caso de llamar a un número o contacto concreto.

7.4.2.3. Estudiar CMUSphinx

Al no haber trabajado tampoco con CMUSphinx, ni con el reconocimiento de voz, se tuvo que realizar una exhaustiva investigación respecto a este tema.

CMUSphinx ofrece varios servicios, entre ellos, reconocimiento de palabras mediante un modelo de audio previamente entrenado y la capacidad de entrenar un modelo de lenguaje. En nuestro caso nos centraremos en los modelos de lenguaje ya entrenados por terceros y ofrecidos a la comunidad.

7.5. Sprint 5

7.5.1. Planificación del sprint

En el esprint anterior, se detectó la necesidad de mejorar el proyecto *Dejavu* para que tomase marcas de tiempos cuando, por ejemplo, detectase audio en la base de datos, o solicitase la transcripción a Google para poder comparar, además, la posible ganancia en eficiencia por usar Dejavu. Durante este esprint se desea actualizar el servicio y proseguir con la investigación de Freeswitch.

7.5.2. Ejecución del sprint

7.5.2.1. Incluir lectura de tiempos del servicio en Integración

Como hemos comentado anteriormente, se necesita tener referencia del tiempo que tarda el servicio en realizar las diferentes acciones. Entre ellas está, detectar si el audio está vacío (no hay sonido), si tiene sonido pero no se detecta ninguna palabra, el audio se reconoce con otros de la base de datos, o el audio no se reconoce en la base de datos y se solicita la transcripción al servicio *speech* de Google.

7.5.2.2. Experimentar con Freeswitch

Para experimentar con el uso de Freeswitch, fue necesario contratar un servidor de Google Cloud[3], con el sistema operativo Debian 8[2] para poder instalar Freeswitch en él y mantenerlo activo.

Se programaron varios *dialplan* para probar su funcionalidad. Por ejemplo: grabación de la llamada, reproducción de audios durante la llamada, detección de DTMF[19], redirección de la llamada, etc. El código escrito en dichas pruebas fue empleado para la versión final de este proyecto.

7.6. Sprint 6

7.6.1. Planificación del sprint

Durante este esprint se desea realizar pruebas sobre el reconocimiento de voz que ofrece Freeswitch utilizando CMUSphinx.

7.6.2. Ejecución del sprint

7.6.2.1. Estudiar CMUSphinx

Siendo poca la información recabada en sprints anteriores sobre esta herramienta, se procede a estudiarla más exhaustivamente.

7.6.2.2. Estudiar integración entre Freeswitch y CMUSphinx

Al ser herramientas completamente distintas pero compatibles entre ellas, nos hemos encontrado con la dificultad de que la documentación referida a la integración entre estas dos utilidades es escasa y poco actualizada.

Entre la documentación encontrada obtenemos dos formas de interconectarlas. En primer lugar emplear un servidor externo utilizando UniMRCP[13] como control entre los dos servidores, uno con Freeswitch y otro con CMUSphinx. La otra opción es utilizado un *mod* de Freeswitch que por debajo interactúa con PocketSphinx[5]. PocketSphinx es un reconocedor de lenguaje, bajo el proyecto CMUSphinx.

Debido a la complejidad y al coste que generaría contratar otro servidor, se descartó la idea de utilizar UniMRCP y se prosiguió utilizando PocketSphinx.

7.7. Sprint 7

7.7.1. Planificación del sprint

Debido a los buenos resultados del proyecto Dejavu en integración lo cuales se pueden ver en el apartado 7.12.1.2, se decide pasar el servicio a producción para probarlo y utilizarlo en un entorno real. Respecto al proyecto Freeswitch se va a continuar con la investigación en la integración entre Freeswitch y PoketSphinx.

7.7.2. Ejecución del sprint

7.7.2.1. Ubicar el servicio del proyecto Dejavu a Producción

Dados los buenos resultados dados durante las pruebas diarias sobre este servicio en Integración, donde en la figura 7.4 se puede observar como el índice de detecciones sin acceder a Google ha aumentado muy rápidamente, tan solo en cuestión de un par de semanas. Recordamos que para las pruebas se elegía al azar entre 10 dispositivos y se llamaba a cada uno 3 veces estando en modo avión o desconectado.

Recordar que se puede consultar el seguimiento del proyecto Dejavu en el apartado 7.12.1.2.

7.7.2.2. Planificar seguimiento del proyecto Dejavu en producción

Como los servicios en producción se pueden reiniciar automáticamente sin previo aviso, por la administración que ofrece Kubernetes, se procede a medir semanalmente, el tiempo de

respuesta ante cada evento mencionado en el apartado 7.5.2.1, además de almacenar la cantidad de resultados ofrecidos en dichos eventos. Aunque esto puede no ser 100 % fiable, sirve para demostrar la eficacia de nuestro nuevo servicio.

7.7.2.3. Reconocer audio durante una llamada voz IP

Se configuró PocketSphinx con un modelo de audio en español. Pese a que tardaba un rato en detectar las palabras, se consiguió detectar con gran acierto las palabras "si", "no" y "casa".

7.8. Sprint 8

7.8.1. Planificación del sprint

Durante este esprint, nos centramos en aprender un nuevo lenguaje de programación: Go. Tras reunir los conocimientos necesarios, realizar un script sencillo para que al conectar vía Socket con Freeswitch, realice algunas de las funciones de prueba que hicimos en *sprints* anteriores.

7.8.2. Ejecución del sprint

7.8.2.1. Estudiar Go

El *product owner* requiere que el script que maneje Freeswitch para realizar la aplicación interactiva por voz se escriba en Go[21]. Al ser un lenguaje nuevo, se tuvo que realizar un serio aprendizaje de la sintaxis del mismo siguiendo el tutorial oficial de Go[2].

7.8.2.2. Realizar script en Go que intervenga en un Dialplan

Tras aprender lo más básico del lenguaje y el uso de algunas de sus librerías, realizamos nuestro primer script en Go, donde realizamos la misma funcionalidad que probamos en un sprint anterior cuando comenzamos a experimentar con Freeswitch. En el apartado 7.5.2.2 se puede recordar lo que hicimos. El código de estos *scripts* se reutilizó para finalizar el proyecto.

7.8.2.3. Comprobar la detección de palabras

Para probar la detección de palabras sin necesidad de estar hablando por teléfono y molestar al resto de compañeros se optó por reproducir los audios mediante un micrófono virtual desde

[2] *Tutorial de Go*[12]

el cliente SIP del ordenador, como si fuese una entrada de micrófono.

Mediante la reproducción de audios, se consiguió detectar algunos números en español (del 0 al 9) con una tasa de éxito de 50 % (no se reconocían los números del 3 al 7) utilizando un modelo de lenguaje en español. Al probar el reconocimiento de palabras en inglés estadounidense con su respectivo modelo de lenguaje, los mismos números en su idioma eran reconocidos un promedio del 60 % (el modelo confundía los números 3, 4, 6 y 8) de los números por llamada. Ambas pruebas se realizaron con mensajes grabados del servicio de reproducción de palabras de *Google translate*[4].

7.9. Sprint 9

7.9.1. Planificación del sprint

Una vez el programa en Go estaba en funcionamiento, el *product owner* sugirió realizar pruebas para observar como se detectaban las palabras de las telealarmas, ya que sería el uso habitual de esta aplicación en fase de experimentación.

7.9.2. Ejecución del sprint

7.9.2.1. Probar la detección de números con los audios grabados de una telealarma

En primer lugar se grabaron las palabras que emite una telealarma mientras se programa, siendo números del 0-9 y pitidos. Una vez grabados estos mensajes se procedió a realizar pruebas con el anterior sistema que teníamos en funcionamiento, que proporcionaba una respuesta con el número que se había detectado. Para tal prueba se realizó una llamada y en ella se reprodujo el audio previamente grabado de programación de una telealarma, de este modo, se comprueba la efectividad del reconocimiento de voz de PocketSphinx frente a estos mensajes.

Los resultados se pueden ver en la figura 7.1 donde se detalla la cantidad de aciertos (u omisiones) por dígito, que es el caso que nos preocupa. Sin embargo, hay ocasiones que detecta algún numero cuando ha sonado uno o varios pitidos, caso que damos por fallo.

Debido a la baja tasa de acierto del test con los audios de la telealarma se decidió buscar otros modelos de audio en español para probarlo de la misma manera. Sin embargo, estos eran anticuados y no pudieron ser ejecutados. Por este motivo se pensó en realizar un modelo de audio personalizado para los comandos que iban a ser emitidos por la telealarma, que son 10 comandos (números del 0 al 9), y que según la documentación[3] de CMUSphinx para desarrollar un modelo de comandos "pequeño", adecuado para nuestro caso ya que cumple estas características. Así, se necesitaría entrenar el modelo con aproximadamente, una hora de mensajes

[3] `https://cmusphinx.github.io/wiki/tutoriallm/#building-a-statistical-language-model`

ESTADÍSTICA	
Aciertos	6
Fallos	25
Omisiones	59
TOTAL	90
Tasa de acierto	6,67%

CANTIDAD DE	
Digitos	82
Pitidos	8
TOTAL	90

Figura 7.1: Tasa de aciertos en el primer test de reconocimiento la telealarma con PocketSphinx.

de audio por cada comando que se desea entrenar. Es decir, un total de 10 horas de mensajes de telealarma.

7.9.2.2. Probar la detección con otros modelos de lenguaje

Debido al escaso tiempo de prácticas que le queda al alumno, se consideró que era más beneficioso para él alumno realizar el programa interactivo que mediante voz natural realizara acciones distintas. Por ello se realizó el mismo proceso de detección de palabras con otros modelos de audio, sin obtener buen resultado.

7.10. Sprint 10

7.10.1. Planificación del sprint

Ya siendo el penúltimo esprint se implementó todo lo aprendido previamente para el servicio interactivo por voz.

7.10.2. Ejecución del sprint

7.10.2.1. Implementación de un servicio interactivo y reactivo por voz

El objetivo de este esprint era realizar un *script* que interactuase como *dialplan* en Freeswitch donde, dadas varias opciones el usuario pudiera realizar una de ellas. Estas acciones fueron:

- Activar el servicio *echo*: este servicio lo usan las telealarmas de 72horas, consiste en grabar durante quince segundos la voz del técnico de ascensores para que, tras este tiempo la telealarma le reproduzca ese mismo audio. Con esto el técnico se asegura del correcto funcionamiento tanto del micrófono como del altavoz.

- Llamar a un agente: este servicio simplemente realiza una redirección de la llamada a otro cliente SIP de Freeswitch. Se intenta simular una centralita de una compañía telefónica.
- Acceder a un servicio de prueba de detección de números: como su nombre indica, detecta los números que se le comuniquen, emplea el reconocedor de voz y repite el número que ha detectado.

El sistema se implementó con el reconocimiento de voz en español que permitía con gran acierto acceder a los servicios "eco", "llamada" y "números" mediante estas palabras clave, sin embargo se pensó en la posibilidad de utilizar el modelo de audio en otro idioma para probar su eficacia frente a este modelo.

7.11. Sprint 11

7.11.1. Planificación del sprint

Para conseguir mejorar la interacción con el usuario se ha establecido que el sistema conteste de diferente manera ante cada tipo de evento que provoca el usuario. Por ejemplo, reconocer una palabra o simplemente escucharla.

7.11.2. Ejecución del sprint

7.11.2.1. Mejorar interacción durante la llamada

Por ello se mejoró el *acknowledgement* (ACK)[1] sustituyendo el anterior sistema de confirmación de acciones por el uso de *Text to Speech* (TTS)[27] de Flite[23].

Flite en Freeswitch por defecto tiene el lenguaje en inglés, y esto no resulta un problema pues sirve como solución para el ACK que se necesitaba.

El idioma por defecto de los servicio TTS y *Automatic Speech Recognition* (ASR)[26] en Freeswitch es el inglés. Por ello se pensó en que posiblemente los resultados fueran mejores en este lenguaje. Para comparar el índice de acierto entre el anterior modelo empleado, español, con este nuevo en inglés, se decide realizar un test de efectividad, el cual se puede ver en el apartado 7.12.2.

7.11.2.2. Seguimiento del proyecto Dejavu en producción

Respecto al seguimiento realizado del proyecto Dejavu en producción, se puede consultarlo en el subapartado 7.12.1.3.

7.12. Validación y seguimiento

En esta sección se presentan las pruebas que se realizaron para verificar y validar los proyectos Dejavu (dividido en dos fases: integración y producción) y Freeswitch.

7.12.1. Proyecto Dejavu

7.12.1.1. Resumen del proyecto Dejavu

Para comprender el proyecto Dejavu, sus respuestas y su método de obtener la transcripción del audio, les hacemos un breve resumen.

El servicio que el proyecto Dejavu proporciona puede responder de tres formas: con un texto vacío, con una excepción o con la transcripción del texto. Para conseguir esto se realizan una serie de comprobaciones, y hay que diferenciar cinco casos por los que la respuesta puede ser una u otra:

- DejavuRecognized: El servicio encuentra una coincidencia en la base de datos y devuelve su transcripción, pudiendo ser una cadena vacía, o un texto.
- InvalidParams: El servicio sufre una excepción que comunica al solicitante de la transcripción.
- NoRecognizedWords: El servicio no encuentra una coincidencia en la base de datos y solicita al servicio de Google la transcripción del mismo, que le devuelve a nuestro servicio una excepción, por lo tanto, Dejavu almacena este audio como audio vacío y retorna una cadena vacía.
- VoidSound: Dejavu detecta que el audio que ha recibido no tiene ninguna *fingerprint* con la que comparar con su base de datos. En este caso se entiende como que el audio no tiene sonido grabado y se devuelve una cadena vacía.
- GoogleRecognized: Al igual que NoRecognizedWords nuestro servicio no encuentra coincidencia en su base de datos, entonces solicita a Google la transcripción del audio y al recibirla la almacena en su base de datos, y devuelve la transcripción que le ha brindado Google.

Más adelante mencionamos los términos “acierto” y “error” a la hora de reconocer un audio. Con “error” nos referimos a que ha sido necesario la intervención del servicio de reconocimiento

de voz de Google para poder transcribir el audio, mientras que con “acierto” nos referimos a no haberlo necesitado para transcribir el audio correctamente. Es decir, que consideraríamos error si el tipo de resultado ha sido “GoogleRecognized” o “NoRecognizedWords” y acierto cuando el tipo de resultado no sean ninguno de los dos anteriores.

7.12.1.2. Seguimiento en integración

Para realizar el seguimiento del servicio en integración se estableció realizar 30 llamadas telefónicas diarias a 10 dispositivos móviles desconectados o en modo avión que se pudieran encontrar en el despacho donde el alumno realizó las prácticas. La cantidad de dispositivos estaba comprendida por 12 teléfonos móviles de compañeros del despacho y 3 telealarmas, en total 15 dispositivos móviles de diferentes operadores a los que realizar las pruebas de reconocimiento de audio. En la figura 7.2 se puede observar tanto el número de dispositivos por compañía telefónica como la probabilidad de ser elegido un teléfono de esta compañía en dichas pruebas.

Operador	Nº de dispositivos	Probabilidad de ser elegido
Amena	1	6,67%
Jazztel	2	13,33%
Movistar	1	6,67%
Movistar M2M	1	6,67%
Orange	2	13,33%
Pepephone	2	13,33%
Simyo	1	6,67%
Vodafone	4	26,67%
Yoigo	1	6,67%

Figura 7.2: Cantidad de dispositivos por operador y su probabilidad de elección en las pruebas.

Al finalizar el seguimiento de este servicio en integración (23/03/18) se obtuvieron la cantidad de errores y aciertos sobre cada operador que se puede observar en la figura 7.3. Asimismo en la figura 7.4 se puede observar cómo ha ido evolucionando la tasa de acierto en cada prueba realizada, dando por validado los sprints desde el número 17.1 al número 37.4.

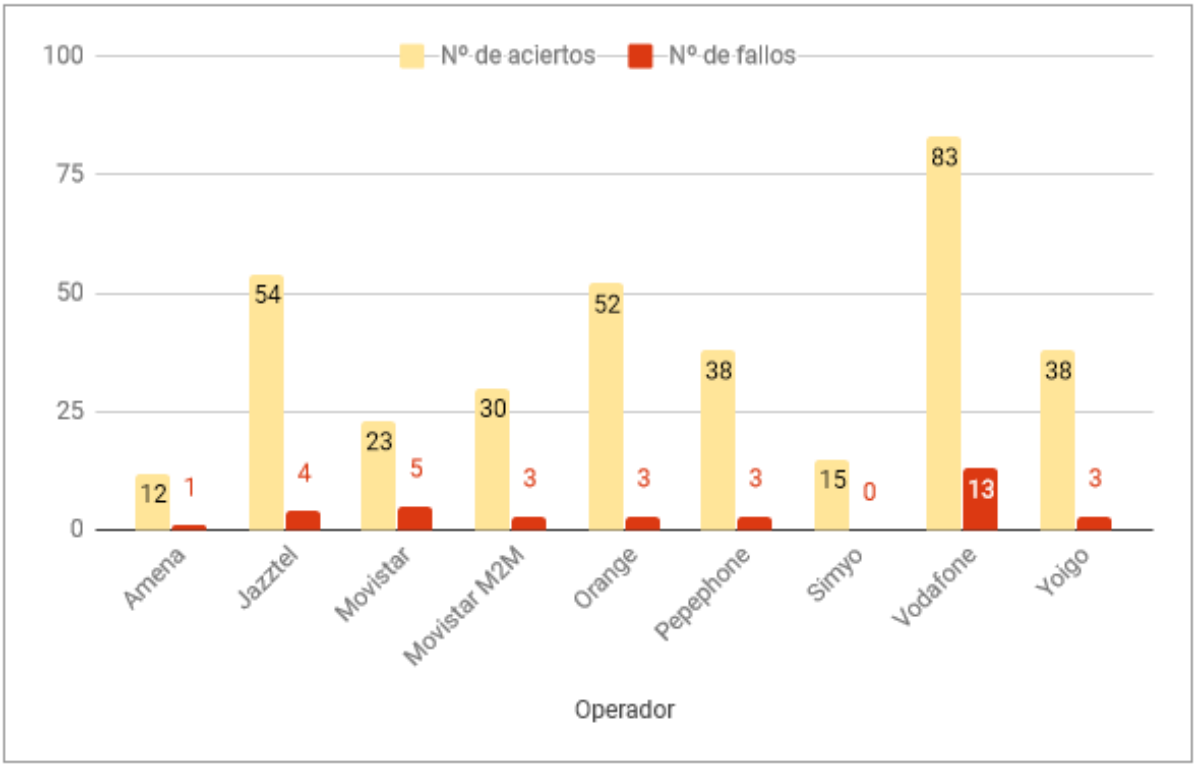

Figura 7.3: Gráfica de aciertos y errores sobre cada operador al finalizar el seguimiento (23/03/18).

7.12.1.3. Seguimiento en producción

El estudio del proyecto Dejavu en producción se ha realizado para demostrar la eficacia del uso del mismo frente a la anterior implementación que se tenía para el servicio de detección del tipo de error ocasionado en una llamada de test a las telealarmas. Para ello, se ha llevado a cabo cinco tomas de marcas de tiempo realizándolas cada viernes desde el inicio de este seguimiento, para demostrar que el uso del actual servicio reduce considerablemente las peticiones al servicio de detección del habla de Google además de que asegura una mayor velocidad de respuesta.

En la figura 7.5 se puede ver las marcas de tiempo tomadas en cada día desde el inicio del seguimiento. Podrán observar que el primer día no hay marca para el tipo de reconocimiento *GoogleRecognized*, esto es porque el servicio en producción emplea la misma base de datos que en integración y que, ya tiene audios con los que poder comparar. En la figura 7.6 se muestra gráficamente estos datos.

Seguidamente, en la figura 7.7 se puede contemplar el desarrollo del servicio, respecto al tipo de reconocimiento que se ha utilizado para la cantidad de veces que se ha solicitado una trascripción del mensaje de audio, es decir, para el día 13/04, se solicitó transcribir 62 audios, de los cuales 17 fueron detectados por el servicio del proyecto Dejavu, 9 de estos audios no contenían o no se reconocían

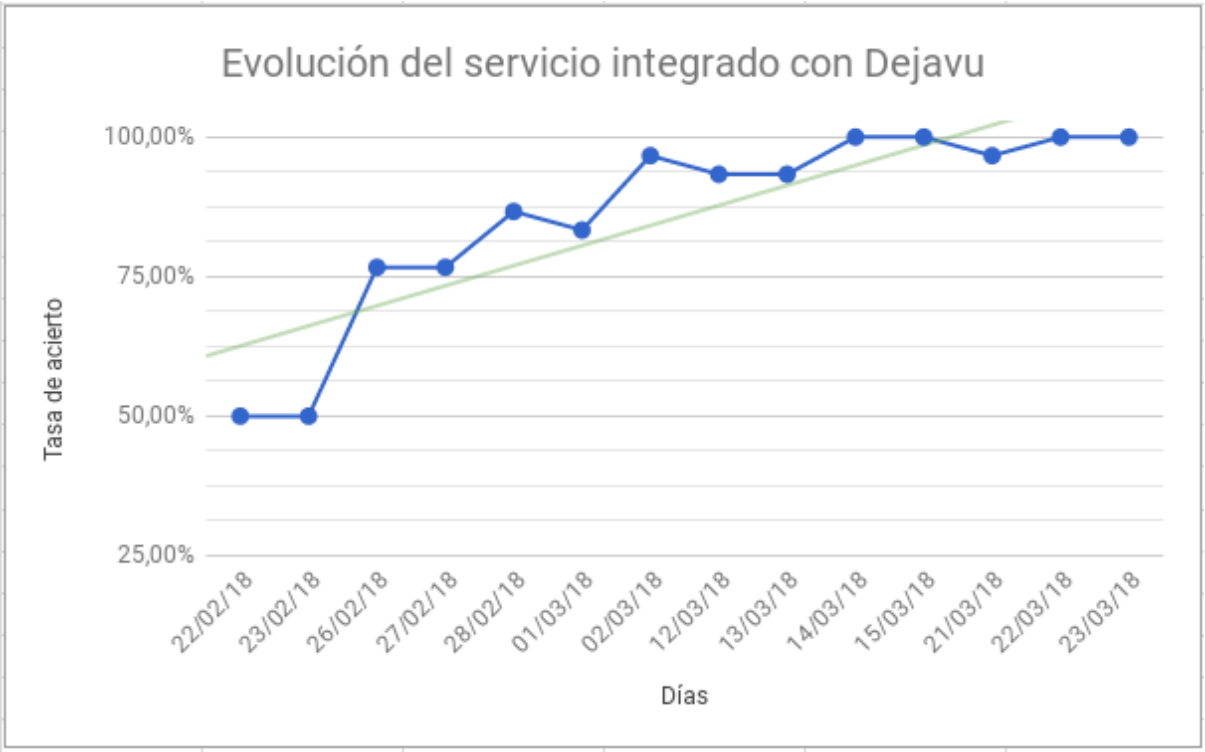

Figura 7.4: Gráfica de la evolución de la cantidad de aciertos en cada prueba (23/03/18).

	DejavuRecognizedMin		NoRecognizedWord		VoidSound		GoogleRecognized	
	T.Mínimo Dejavu	T.Maximo Dejavu	T.Mínimo NoRecognized	T.Maximo NoRecognized	T.Mínimo Void	T.Maximo Void	T.Mínimo Google	T.Maximo Google
13/04	0,8852982521	1,2682809830	4,3119659424	5,9781298637	0,0050818920	0,0144259930		
20/04	0,0885298252	1,2682809830	0,4421582222	6,3923738003	0,0050818920	0,0144259930	2,1671199799	2,8301620483
27/04	0,0885298252	1,8377740383	0,4421582222	6,3923738003	0,0050818920	0,0144259930	2,1671199799	2,8301620483
03/05	0,0885298252	2,4429061413	0,4421582222	7,8158969879	0,0050818920	0,0144259930	2,1671199799	9,9874720573
11/05	0,0885298252	2,4429061413	0,4421582222	7,8158969879	0,0050818920	0,0144259930	1,9774649143	9,9874720573

Figura 7.5: Marcas de tiempo mínima y máxima en evolución de los resultados ofrecidos por el servicio de reconocimiento de audios grabados, Dejavu.

A continuación se puede observar gráficamente en la figura 4.1 cómo se ha desarrollado las marcas de tiempo mínimo y máximo del servicio del proyecto Dejavu.

13/04	
Nº de detecciones	
DejavuRecognized	17
InvalidParams	0
NoRecognizedWords	9
VoidSound	36
GoogleRecognized	0
TOTAL	62

20/04	
Nº de detecciones	
DejavuRecognized	55
InvalidParams	0
NoRecognizedWords	31
VoidSound	36
GoogleRecognized	2
TOTAL	124

27/04	
Nº de detecciones	
DejavuRecognized	78
InvalidParams	0
NoRecognizedWords	34
VoidSound	36
GoogleRecognized	2
TOTAL	150

03/05	
Nº de detecciones totales	
DejavuRecognized	131
InvalidParams	0
NoRecognizedWords	42
VoidSound	36
GoogleRecognized	3
TOTAL	212

11/05	
Nº de detecciones totales	
DejavuRecognized	211
InvalidParams	0
NoRecognizedWords	94
VoidSound	36
GoogleRecognized	8
TOTAL	349

Figura 7.6: Evolución del número de detecciones según el tipo en cada toma de marcas.

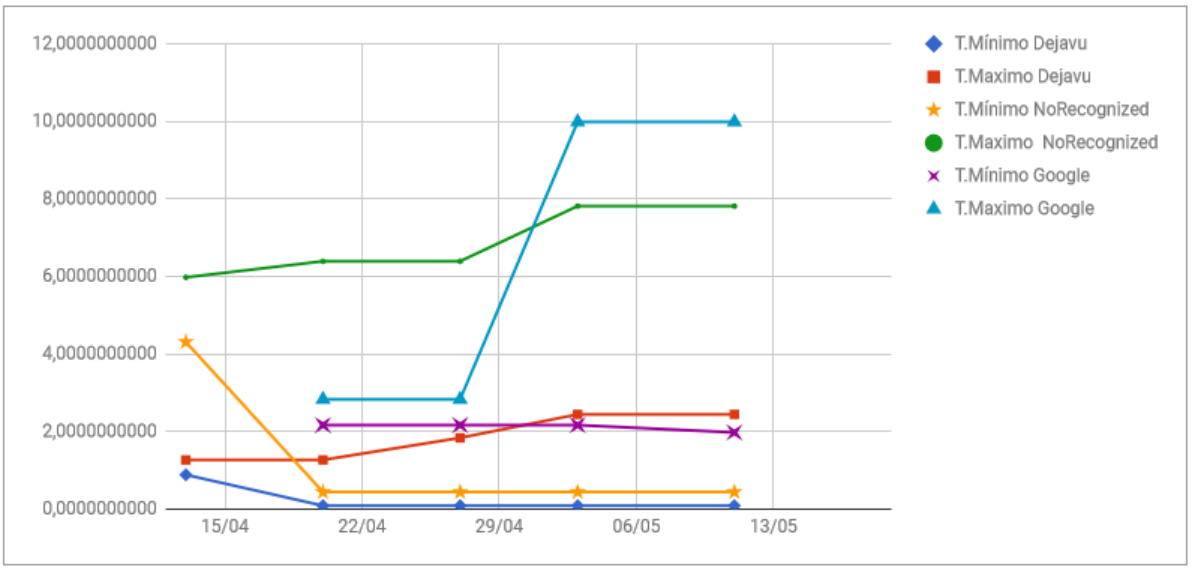

Figura 7.7: Evolución general de las marcas de tiempo de los diferentes resultados por el servicio de reconocimiento de audios grabados, Dejavu.

7.12.2. Validación del proyecto Freeswitch

Para validar el proyecto Freeswitch se realizaron tres pruebas, las cuales se puede ver los detalles en la figura 7.8. Como se puede observar la tasa de acierto es idéntica, sin embargo donde se diferencian estos dos modelos de audio es en la cantidad de intentos necesarios para reconocer una palabra, es decir, que el sistema detecta que se está hablando, pero no es capaz

de reconocer ninguna palabra. Por ello se optó por emplear el modelo de lenguaje en e inglés a falta de uno en español mejor entrenado ya que el *product owner* solicitó en el apartado 5.1 se pedía que este servicio reconociese, al menos el 80 % de los casos. Dando por validados los sprints del 57.5 al 117.11.

		Español			Inglés		
		eco	*telefono*	*números*	*echo*	*telephone*	*numbers*
Prueba 1	Nº de intentos antes de reconocer	9		8	1	1	2
	Nº intentos antes de fallar		9				
Prueba 2	Nº de intentos antes de reconocer	5	11	8	2		2
	Nº intentos antes de fallar					1	
Prueba 3	Nº de intentos antes de reconocer	8	9		1	3	
	Nº intentos antes de fallar			12			1
Intentos mínimo a ser reconocido		5	9	8	1	1	2
Intentos máximo a ser reconocido		9	11	8	2	3	2
MEDIAS	**Intentos total**	7,33	9,67	9,33	1,33	1,67	1,67
	Intentos con aciertos al final	7,33	10,00	8,00	1,33	2,00	2,00
	Intentos con fallo al final	0,00	9,00	12,00	0,00	1,00	1,00
% ACIERTO		100,00%	66,67%	66,67%	100,00%	66,67%	66,67%
% ACIERTO CASOS		73,42%			85,71%		

Figura 7.8: Tabla con los resultados de las tres pruebas utilizando los comandos en el idioma correspondiente.

7.12.3. Conclusiones del seguimiento y la validación

7.12.3.1. Proyecto Dejavu

En pocos días se obtuvieron muy buenos resultados del servicio implementado con la librería Dejavu. Además, gracias a estas pruebas se consiguió captar una buena cantidad de audios de distintas operadoras por lo que a la hora de pasarlo a producción ya se tenía una buena base de audios con los que poder comparar.

Debido a la pequeña base de datos creada a raíz de las pruebas en el servidor de integración, cuando se pasó a producción, el servicio comenzó dando muy buenos resultados, sin necesidad de consultar a Google. Más adelante vemos que hay ocasiones que se necesita la transcripción y que gracias a esto, vemos lo eficiente que es evitar estas solicitudes a un servicio externo como es el *speech recognition* de Google.

Poco hemos hablado del estado de la base de datos, a día 23/05/18 alberga los datos de 301 audios de los cuales 246 son audios sin palabras reconocibles o pitidos. Se configuró de esta manera porque dado este tipo de casos, el servicio siempre solicitaba a Google la transcripción del audio que, si cumple las características descritas anteriormente devolvía una cadena vacía por el método "NoRecognizedWords", pero el hecho es que siempre pasaba por Google, y para remediar esto, se decidió también almacenar este tipo de audios para prevenir el uso del servicio de Google.

En general, el sistema funciona perfectamente, con la salvedad de los audios que no se detectan palabras, estos, pese a que tienen *fingerprints*, no se consigue reconocer con total certeza, pues al parecer inunda la base de datos con audios de este tipo. Esto a la larga podrá resultar un problema y se debe estudiar más a fondo para solucionarlo.

7.12.3.2. Proyecto Freeswitch

Respecto al proyecto Freeswitch, hemos de comentar que no se ha conseguido los resultados esperados con el lenguaje en español como pensamos. Esto se debe a que cada modelo de audio ha sido entrenado de manera diferente y con un tamaño del conjunto de entrenamiento diferente, lo que conlleva a un bajo nivel de *fitness* del modelo de lenguaje, cosa que dificulta al programa (PocketSphinx) reconocer los fonemas. Esto se podría solucionar con más tiempo, donde se consiga muchas más grabaciones en lengua española.

Capítulo 8

Conclusiones

Índice del capítulo

8.1. Conclusiones técnicas

En cuanto a conclusiones técnicas, cabe indicar que se han alcanzado los objetivos que se propusieron al principio de la estancia en prácticas y que se consiguió un muy buen resultado con el proyecto Dejavu en poco tiempo.

Respecto al proyecto Freeswitch, el resultado no era el esperado, pero debido a los problemas a la hora de encontrar un modelo de audio en español, se optó por emplear un modelo de lenguaje inglés, que cumplió con los requisitos que se esperaban.

8.2. Conclusiones personales

Respecto al proyecto Dejavu sí que se tenía conciencia sobre los componentes a tratar y cómo interrelacionarlos, personalmente, resultó una buena toma de contacto con la empresa y la metodología de trabajo.

El proyecto Freeswitch parecía muy ambicioso, sobretodo por la nula experiencia y conocimiento sobre redes VoIP, reconocimiento de voz y el lenguaje de programación Go. Personalmente, estoy muy satisfecho con los resultados obtenidos tras la estancia en prácticas, indagar e investigar sobre Freeswitch y CMUSphinx ha sido una experiencia muy grata, y gracias a esto me he dado cuenta de lo potentes que son estas herramientas, que por suerte o desgracia no

había tenido noticias de su existencia. Esto me ha enseñado la puerta del inmenso mar que me queda descubrir en todo el ámbito de la Informática.

8.3. Posibles mejoras

Las posibles mejoras que creo que pueden resultar útiles para el proyecto Dejavu es poder asegurar que un audio no contiene palabras grabadas o no son audibles, con el fin de evitar almacenarlas en la base de datos, ni consultar a Google sobre los mismos.

Respecto a Freeswitch, las mejoras que se le pueden hacer son innumerables por ejemplo, se le pueden incluir muchos más servicios y no sólo eso, sino reconocer ciertas palabras durante una llamada de auxilio desde un ascensor, para determinar el estado de los ocupantes.

Bibliografía

[1] Acknowledgement. https://es.wikipedia.org/wiki/ACK. [Consulta: 16 de Mayo de 2018].

[2] Debian 8. https://www.debian.org/releases/jessie/. [Consulta: 15 de Mayo de 2018].

[3] Google cloud. https://cloud.google.com/?hl=es. [Consulta: 16 de Mayo de 2018].

[4] Gooogle translate. https://translate.google.com/?hl=es. [Consulta: 16 de Mayo de 2018].

[5] Información sobre pocketsphinx. https://cmusphinx.github.io/2017/03/pocketsphinx-as-standalone-app-on-android-wearables/. [Consulta: 15 de Mayo de 2018].

[6] Kubernetes.io main page. https://kubernetes.io/. [Consulta: 15 de Mayo de 2018].

[7] Librería *speech* de google. https://cloud.google.com/speech-to-text/. [Consulta: 10 de Mayo de 2018].

[8] Mysql main page. https://www.mysql.com/. [Consulta: 15 de Mayo de 2018].

[9] Postgresql main page. https://www.postgresql.org/. [Consulta: 15 de Mayo de 2018].

[10] Página principal de freeswitch. https://freeswitch.com/. [Consulta: 21 de Mayo de 2018].

[11] Sip. https://es.wikipedia.org/wiki/Protocolo_de_iniciaci%C3%B3n_de_sesi%C3%B3n. [Consulta: 15 de Mayo de 2018].

[12] Tutorial de go. https://tour.golang.org/welcome/1. [Consulta: 16 de Febrero de 2018].

[13] Unimrcp main page. http://www.unimrcp.org/. [Consulta: 15 de Mayo de 2018].

[14] Raúl González Duque. Pyton para todos. http://mundogeek.net/tutorial-python/. [Consulta: 5 de Febrero de 2018].

[15] Jr. Guido van Rossum Editor: Fred L. Drake. Tutorial de python 2.7.0. http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf. [Consulta: 3 de Febrero de 2018].

[16] José Luis Aracil Gómez del Campo. Nexus. https://github.com/jaracil/nexus/wiki. [Consulta: 21 de Mayo de 2018].

[17] Wikipedia. Acoustic fingerprint. https://en.wikipedia.org/wiki/Acoustic_fingerprint. [Consulta: 15 de Mayo de 2018].

[18] Wikipedia. Apartado sphinx. https://es.wikipedia.org/wiki/CMU_Sphinx#Sphinx. [Consulta: 21 de Mayo de 2018].

[19] Wikipedia. Dtmf. https://es.wikipedia.org/wiki/Marcaci%C3%B3n_por_tonos. [Consulta: 15 de Mayo de 2018].

[20] Wikipedia. Git. https://es.wikipedia.org/wiki/Git. [Consulta: 16 de Mayo de 2018].

[21] Wikipedia. Go. https://es.wikipedia.org/wiki/Go_(lenguaje_de_programaci\C3%B3n). [Consulta: 16 de Mayo de 2018].

[22] Wikipedia. Oracle corporation. https://es.wikipedia.org/wiki/Oracle_Corporation. [Consulta: 21 de Mayo de 2018].

[23] Wikipedia. Proyecto festival. https://es.wikipedia.org/wiki/Festival_(TTS)#Flite. [Consulta: 16 de Mayo de 2018].

[24] Wikipedia. Python. https://es.wikipedia.org/wiki/Python. [Consulta: 16 de Mayo de 2018].

[25] Wikipedia. Scrum. https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software). [Consulta: 10 de Mayo de 2018].

[26] Wikipedia. *Automatic Speech Recognition*. https://es.wikipedia.org/wiki/Reconocimiento_del_habla. [Consulta: 16 de Mayo de 2018].

[27] Wikipedia. *Text to Speech*. https://es.wikipedia.org/wiki/Conversor_texto-voz. [Consulta: 17 de Mayo de 2018].

[28] Wikipedia. Trello. https://es.wikipedia.org/wiki/Trello. [Consulta: 16 de Mayo de 2018].

[29] Wikipedia. Vi. https://es.wikipedia.org/wiki/Vi. [Consulta: 21 de Mayo de 2018].

[30] Wikipedia. Visual studio code. https://en.wikipedia.org/wiki/Visual_Studio_Code. [Consulta: 16 de Mayo de 2018].

[31] Wikipedia. Voz ip. https://es.wikipedia.org/wiki/Voz_sobre_protocolo_de_internet. [Consulta: 21 de Mayo de 2018].

[32] Worldveil. Dejavu. https://github.com/worldveil/dejavu. [Consulta: 5 de Febrero de 2018].

[33] Worldveil. Repositorio de worldveil. https://github.com/worldveil. [Consulta: 5 de Febrero de 2018].