Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

AskTelegram: bot conversacional para traducción voz a voz

Autor: Miguel González Santos

Enero 2023

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Computación

AskTelegram: bot conversacional para traducción voz a voz

Autor: Miguel González Santos

Tutor: Valentín Cardeñoso Payo

A mi madre, mi padre, mi hermana y mi grupo de la carrera

Agradecimientos

Han sido muchas personas las que han ayudado a que este proyecto haya salido adelante.

En primer lugar agradecérselo a mi tutor Valentín, ya que a parte de proponer un tema interesante en el que he disfrutado trabajando, me ha ayudado mucho con las revisiones periódicas y consejos aportados.

De la misma forma hacer un guiño a todos esos profesores que me han enseñado todo lo que sé y me han motivado para elegir el camino en el que estoy ahora.

Ya en un tono más personal gracias a mi familia, especialmente a mis padres y a mi hermana, que siempre han estado apoyándome y creyendo en mi en todo momento.

Gracias a mi grupo de la carrera *Lame9*, con los cuáles he vivido tanto en tan poco tiempo, haciendo de esta carrera una de las mejores experiencias de mi vida.

Gracias.

Resumen

El objetivo de este proyecto es la creación de un bot de Telegram que traduzca mensajes de voz de un idioma a otro cualquiera.

Principalmente, esto permitirá tanto usarlo de modo individual para reproducir los audios traducidos a otra persona, así como para invitar al bot a un grupo de usuarios y que cada uno lo use al mismo tiempo para tener una conversación en un mismo idioma conjunto.

Además, implementa traducciones texto a texto y texto a audio para aportar la mayor versatilidad posible.

Este bot hará más sencillo hablar en diferentes idiomas o simplemente practicarlos por medio de Telegram de una manera gratuita y rápida.

Abstract

The goal of this project is the creation of a Telegram's bot able to translate voice messages from one language to any other.

Mainly, this will allow both to use it individually to play the translated audios to another person, or to invite the bot to a group of users and let each one use it at the same time to have a conversation in the same language together.

In addition, it implements text-to-text and text-to-audio translations to provide the greatest possible versatility.

This bot will make it easier to speak in different languages or simply practice them through Telegram in a free and fast way.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

Hoy en día, todos los que usamos en cierta medida internet hemos empleado alguna vez tecnologías de traducción de texto, así como de detección de voz, la cuál se está volviendo cada vez más común en el día a día de las personas gracias a sistemas innovadores como *Alexa* o *Siri*.

También muchas personas se habrán topado con BOTS en aplicaciones que permiten mantener un chat con otros usuarios como es el caso de DISCORD, TELEGRAM o WHATSAPP. Estos BOTS permiten al usuario acceder a todo tipo de funcionalidades de una forma muy simple, similar al uso habitual que se le da a estas aplicaciones.

Juntar estos dos conceptos es precisamente el tema de este proyecto; hacer posible a un usuario, en este caso de TELEGRAM, traducir un mensaje de voz que envíe en cualquier idioma a otro que éste desee de la manera más simple posible.

1.1 Motivación

La sociedad actual se encuentra en una evolución constante de globalización y digitalización, cada vez es más frecuente que las grandes compañías del mercado incorporen en sus productos nuevas características de tecnología punta, existiendo así una competición para la investigación a gran escala.

Una de estas tecnologías que se encuentra en auge en los últimos años es el uso del reconocimiento y síntesis del habla para el desarrollo de asistentes digitales como ALEXA o SIRI mediante el procesamiento del lenguaje natural y la inteligencia artificial. Estos sistemas aportan al usuario una excelente experiencia debido principalmente a la comodidad de su uso y aprendizaje, así como la sensación de futurismo de algo tan innovador.

Por otro lado, la traducción entre idiomas es una de las funciones más empleadas a nivel global y existe por este mismo motivo una gran cantidad de portales y aplicaciones que llevan a cabo esto mismo. Debido a la gran globalización existente que mencionamos anteriormente, gran parte de las barreras entre los diferentes países ya no existen, llevando a la necesidad del entendimiento de una gran cantidad idiomas tanto en el entorno laboral como en el de turismo o entretenimiento, así como simplemente para navegar por Internet de una manera óptima.

Combinando de forma útil estas dos características obtenemos una herramienta de traducción muy potente, encontrando un mercado con grandes posibilidades y un gran número potencial de clientes que estarán interesados en el producto final en caso de conseguir fusionar ambas partes en un solo bloque que tenga la calidad suficiente para competir con la situación actual de mercado.

1.2 Objetivos

El objetivo principal de este proyecto es el de desarrollar un BOT de TELEGRAM que permita, dado un mensaje de voz a través de un *chat*, detectar lo que dice y traducirlo a otro idioma cualquiera devolviendo dicha traducción por ese mismo chat en forma de audio.

Aprovechando que esta funcionalidad emplea varios pasos que por sí solos dan una utilidad interesante, también se desarrollarán comandos para la traducción texto a texto, texto a audio y audio a texto.

Se busca la mayor simplicidad para el usuario posible, respetando reglas comunitarias para la creación de BOTS de TELEGRAM como los comandos globales[17].

Este BOT guardará la configuración del usuario y permitirá su uso al completo en chats grupales, usando la configuración de cada usuario para que empleen el modo o idiomas que crea oportuno.

Por último, es importante que la instancia del BOT esté activa constantemente para que los usuarios puedan hacer uso de él en cualquier momento, para esto se levantará en un entorno externo.

1.3 Alcance

Eñl grado de consecución del objetivo principal dependerá mucho de las tecnologías que se encuentren para la realización de las tareas más arduas de la aplicación: *speech-to-text*, la traducción y el *text-to-speech*, que se incluirán a partir de las librerías y APIs que se encuentren disponibles, así como en la compatibilidad entre estas funcionalidades.

Además, se implementarán funcionalidades independientes para la traducción texto a texto y texto a audio a disposición del usuario por medio de comandos. También cubriremos los comandos generales definidos por la comunidad de BOTS de TELEGRAM, como son `/start` y `/help`, ciñéndonos en cada paso a los términos de seguridad y privacidad.

Por último, como una de las prioridades, se aportarán los recursos necesarios a que la instancia del BOT esté funcionando en un sistema continuamente, estando listo para escuchar a los usuarios en cualquier momento.

Todos estos puntos deberán de estar implementados con una calidad notable para el fin del proyecto establecido el 24 de octubre, en caso de no poseer dichas funcionalidades, además del desarrollo correspondiente de la memoria, llevarán a la necesidad de la modificación del alcance inicial del proyecto.

Por otro lado, las adiciones que se vayan discutiendo y desarrollando a lo largo del proyecto, así como la necesidad de guardar una configuración para cada usuario tomarán un peso secundario, no repercutiendo al alcance establecido del proyecto a menos que sea estrictamente necesario.

1.4 Estructura de la memoria

A continuación de este capítulo introductorio donde se ha expuesto una primera toma de contacto, la motivación y un contexto, se encuentran el siguiente contenido:

1. **Capítulo 2 - Metodología y Planificación:** Metodología seguida, fases en las que se han dividido las tareas y el seguimiento del avance del proyecto.

2. **Capítulo 3 - Soluciones Existentes:** Proyectos con una funcionalidad similar a nuestro objetivo observados previamente al comienzo del desarrollo.

3. **Capítulo 4 - Marco Conceptual:** Descripción de conceptos teóricos necesarios para la comprensión del proyecto, así como los enfoques que se han tomado para dichos conceptos.

4. **Capítulo 5 - Descripción de la solución:** Se describen los elementos fundamentales de la solución, desde un punto de vista general y de relación entre partes.

5. **Capítulo 6 - Requisitos:** Definición de los requisitos funcionales, no funcionales y de información.

6. **Capítulo 7 - Análisis:** Modelo de dominio, casos de uso y diagramas de actividad.

7. **Capítulo 8 - Diseño:** Patrón de diseño que se ha seguido, el diagrama de clases y la arquitectura del sistema.

8. **Capítulo 9 - Implementación y Pruebas:** Descripción de la herramientas y tecnologías usadas en el proyecto, detalles de la solución implementada y las pruebas de aceptación realizadas.

9. **Capítulo 10 - Conclusión:** Conclusiones obtenidas tras la realización de este proyecto, así como las diferentes posibilidades para un trabajo futuro.

Capítulo 2

Metodología y Planificación

2.1 Metodología

Se ha optado por una metodología ***Agile SCRUM***[13]. Con esta metodología, los proyectos se dividen en pequeñas tareas desarrolladas de forma incremental entregadas en sprints. En este Trabajo de Fin de Grado se han establecido sprints de 2 semanas, que viene a ser la duración estándar (los sprints generalmente comprenden un periodo de tiempo de entre 1 y 4 semanas). Realizado el sprint se lleva a cabo una reunión entre las partes interesadas y el equipo de desarrollo para mostrar los avances del proyecto, donde se evalúan el desarrollo de cualquier cambio y mejora que se considere necesaria.

En este proyecto, las reuniones se realizan en la mitad del sprint para comprobar su estado y que las mejoras se incorporen al entregable pertinente para final de su sprint.

En este modelo de desarrollo existen 3 roles fundamentales:

- ***Product Owner***: Responsable de comunicar la visión del cliente sobre el producto al equipo de desarrollo.
- ***Scrum Master***: Actúa como enlace entre el propietario del producto y el equipo.
- ***Team Member***: Facilita el trabajo de desarrollo.

Estos 3 roles son llevados a cambio por los siguientes participantes:

- ***Product Owner***: Valentín Cardeñoso Payo.
- ***Scrum Master***: Miguel González Santos.
- ***Team Member***: Miguel González Santos.

2.2 Tareas a realizar

1. Definir el trabajo y elaborar una planificación
2. Localizar soluciones similares
3. Investigación de tecnologías
4. Creación del BOT
5. Traducción texto a texto
6. Traducción audio a texto
7. Traducción texto a audio
8. Implementar funcionalidad multiusuario
9. Levantar BOT en entorno externo
10. Pruebas y experimentación
11. Escritura de la memoria

2.3 Fases

El proyecto consta de 300 horas de trabajo, las cuales han sido distribuidas comenzando el **5 de julio** y finalizando el **24 de octubre**. Esto nos lleva a una duración total de **8 sprints de 2 semanas**.

El desarrollo estará dividido en 4 prototipos o entregables que se irán implementando incrementalmente. En este caso, al ser una misma instancia de un mismo BOT, partimos con el requisito de que solo se puede levantar una versión simultáneamente, pero éstas tienen que funcionar al completo independientemente. La funcionalidad que engloba cada prototipo es la siguiente:

1. Personalización del BOT (Nombre, descripción, foto de perfil, etc.) y comando echo simple.
2. Traducción texto a texto.
3. Funcionalidad multiusuario y *logging*.
4. Traducción de audio e instancia corriendo en un entorno externo.

Teniendo esto en cuenta y las tareas mencionadas en la sección anterior:

Nombre de actividad	**Sprints**
Elaborar planificación	1
Localizar soluciones similares	1
Investigación de tecnologías	1
Desarrollo primer prototipo	1
Desarrollo segundo prototipo	2
Desarrollo tercer prototipo	3
Desarrollo cuarto prototipo	4 - 6
Escribir la memoria	6 - 8

Cuadro 2.1: Fases de desarrollo del proyecto previstas.

2.4 Seguimiento por sprints

2.4.1 Sprint 1: Planificación

Duración Del 5 al 19 de julio

Objetivos principales En este sprint inicial se elabora la planificación, dividiendo la funcionalidad en los 4 prototipos que se desarrollarían posteriormente y repartiendo el peso entre los diferentes sprints pertinentemente valorando el peso de cada una de estas funcionalidades, siendo el último prototipo el más delicado y por ello el único que comprende más de 2 semanas de duración.

Actividades Se localizaron varias soluciones similares para nutrirse de ellas y valorar opciones. Estos proyectos se mencionan en mayor detalle en el capítulo Soluciones Existentes 3.

Como última tarea antes del desarrollo del código, hubo una investigación intensiva de las tecnologías que se iban a emplear en el proyecto, comprendiendo desde el lenguaje de programación y herramientas hasta las librerías a emplear para cada funcionalidad.

Para finalizar se desarrolló el primer prototipo, que básicamente consistió en la creación del BOT de manera rápida gracias a botFather[16] y su puesta en marcha en local asociando la instancia a un script de PYTHON. Tras esto simplemente se le añadió un comando echo simple y se dió por finalizado el sprint.

Dificultades No hubo ninguna reseñable

2.4.2 Sprint 2: Segundo prototipo

Duración Del 19 de julio al 2 de agosto

Objetivos principales Este sprint se centra en el desarrollo del segundo prototipo basado en la traducción texto a texto.

Actividades Mediante un comando que traduce y otros dos comandos auxiliares para cambiar los idiomas origen y destino, quedó implementada sin una complicación mayor gracias al uso de una librería en PYTHON.

Dado que fue menos esfuerzo del planeado, investigué mas a fondo tecnologías para la tarea de dejar el BOT corriendo en un entorno externo, en concreto Docker[2] y Heroku[9]. Finalmente, dado que se levantará el código en una máquina virtual de la UVa, decidí dejar esta decisión tras verificar el funcionamiento en dicha máquina.

Tras reunión con tutor: Se propuso una mejora para facilitar el uso de la traducción de texto para el usuario, en la cual no se tuviese que escribir el comando cada vez que se quisiese traducir algo. Por este motivo se implementó el *fast mode*, pudiendo activarlo y desactivarlo a gusto del usuario y que consiste en traducir todos los mensajes que se reciban indiscriminadamente en caso de que este modo se encuentre activado.

Dificultades El nuevo modo implementado chocaba con el comando inicial de traducción dado que a veces se llamaba a ambos simultáneamente, cuando obviamente esto no debería suceder. Esto se solucionó haciendo un uso correcto del patrón de arquitectura y encapsulando las funcionalidades de forma separada en funciones independientes.

2.4.3 Sprint 3: Tercer prototipo

Duración Del 2 al 16 de agosto

Objetivos principales Este sprint se centró en el desarrollo del tercer prototipo, cuya implementación principal era la de proporcionar un registro de *logs*, así como adaptar la funcionalidad para que no existan colisiones entre los diferentes usuarios.

Actividades Para el log de interacciones del usuario con el BOT se empleó la librería de *logging* estándar y como método general se informa del comando elegido, el usuario que está ejecutando dicho comando y la diferente información que se emplee como por ejemplo los idiomas origen y destino y el mensaje enviado en el caso de la traducción texto a texto.

Una vez tratamos esto, pasamos a la funcionalidad principal de este sprint, la cual en términos simplificados era conseguir que cada usuario tuviese su propia instancia personal del BOT para que en un mismo grupo cada persona tenga seleccionados sus propios idiomas y el *fast mode* como se desee. Para esta tarea hubo una tarea densa de investigación, ya que no sabía si existía una necesidad de crear una base de datos, o valía con una estructura interna en el código.

Finalmente la solución fue muy superior a las expectativas, ya que al inspeccionar a fondo la documentación de la librería que empleo para implementar la funcionalidad de TELEGRAM en PYTHON, se encontró que cada usuario tenía su propio contexto (un diccionario donde se pueden almacenar valores), y usando esto para almacenar tanto los idiomas como el interruptor del *fast mode* se pudo llegar a la solución de una manera simple y ese esfuerzo mayor que se planteaba inicialmente.

Dificultades Probando si el multiusuario funcionaba correctamente en grupos de chat, descubrimos que el BOT no podía acceder a los mensajes de los usuarios a menos que estos usaran un comando o lo llamaran al inicio del mensaje debido a la configuración de seguridad inicial de los BOTS de TELEGRAM. Esto cancelaba completamente el *fast mode* en grupos, por lo que finalmente se desactivó esa opción de seguridad, lo cuál se puede visualizar en el propio BOT al acceder desde TELEGRAM.

Otro problema es que inicialmente se detectaba el idioma del usuario desde su perfil para procurar hacer al usuario más sencilla su primera interacción con el BOT. Esto se realizaba al activar un usuario al BOT, pero debido que en los grupos no todos los usuarios han tenido por qué iniciado el BOT de forma separada, finalmente optamos por dejar el idioma origen por defecto en auto-detectar y el destino en inglés.

2.4.4 Sprint 4: Comienzo del cuarto prototipo

Duración Del 16 al 30 de agosto

Objetivos principales Comienzo en el desarrollo del cuarto y último prototipo, teniendo como objetivo el de conseguir recibir un mensaje de voz de un usuario de TELEGRAM y convertirlo a texto.

Actividades Gracias al código de VOICOS mostrado públicamente en su proyecto de GITHUB[8] observamos que la propia librería usada contiene una función para obtener el mensaje de voz como archivo de audio en formato *.ogg* y descargarlo de manera simple, por lo que intentamos implementar dicha aproximación.

Dificultades El formato de archivo *.ogg* no es aceptado por la librería de reconocimiento del habla que queríamos usar, y para transformar de *.ogg* a *.wav*, que es uno de los formatos deseados, existe una dependencia con FFMPEG[5]. Esto no sería un problema de no ser porque es una dependencia algo deprecada y la instalación en WINDOWS de esto por separado no es especialmente simple.

De hecho, tras una investigación a fondo de un gran número de foros de dudas que tenían el mismo problema y probando diferentes soluciones al respecto, se llegó a la decisión de posponer esta instalación para el entorno virtual de la máquina virtual de la UVA, ya que al estar basada en UBUNTU podría ser extremadamente simple de llevar a cabo según alguno de esos foros de dudas investigado.

2.4.5 Sprint 5: Fin del cuarto y último prototipo

Duración Del 30 de agosto al 13 de septiembre

Objetivos principales En principio en este sprint entraba únicamente la transformación de texto a la traducción en forma de audio, pero al arrastrar la finalización de la tarea del sprint anterior y haber recibido la conexión a la máquina virtual de la UVa acabó siendo el sprint con mayor carga de trabajo de desarrollo del proyecto.

Actividades Comenzaré comentando la tarea que requirió el menor tiempo de todas, la cual es de hecho la que se planeaba ejecutar en este sprint. La librería que se había planteado en el primer sprint fue lo suficientemente eficaz y simple para implementar un comando que transformase el texto dado a un audio apoyándose de una librería que emplea la api de GOOGLE para llevar a cabo la funcionalidad.

Tras la reunión de mediados de sprint, el tutor se encargó de preparar la máquina virtual y poblarla con un entorno virtual que contenía todas las dependencias del proyecto. Además, me guió en la conexión con la misma y el traspaso de mi proyecto local.

Llegado al bloqueo sucedido en el anterior sprint, fue un alivio comprobar lo sencillo que era la instalación en el entorno virtual de la máquina, solventándose inmediatamente.

Dificultades Inicialmente estaba planteado emplear SPHINX como motor de reconocimiento del habla, el cual además de dar un problema con las dependencias que requirió de un tiempo inesperado para su solución, finalmente se descartó su uso ya que no detectaba satisfactoriamente la mayoría de mensajes de voz probados. La solución final emplea una token gratuita de la API de GOOGLE para la detección de audio.

Conclusión Tanto el reconocimiento como la síntesis del habla han sido finalmente llevadas a cabo gracias a llamadas indirectas a la API de GOOGLE, lo cual inicialmente no estaba en mente por el coste, pero al esquivar este coste con el uso de librerías externas, finalmente conseguimos una herramienta muy potente para llevar a cabo esta funcionalidad. Tras conectar estas dos funcionalidades a la ya creada de traducción, el objetivo principal quedó cumplido satisfactoriamente.

2.4.6 Sprint 6: Memoria y despliegue de la aplicación

Duración Del 13 al 27 de septiembre

Objetivos principales Este sprint da comienzo a la elaboración de la memoria estando orientado completamente a esta, pero a mediados del sprint se incluye el objetivo de implementar una mayor distribución en la aplicación.

Actividades Una vez decidido el índice a seguir a modo de esquema para una mayor organización en la lectura, comienza la escritura, completando así el capítulo de introducción junto con el resumen y los agradecimientos, además de aprender el uso de la bibliografía y las referencias en LaTeX entre muchas otras técnicas.

Tras la reunión con el tutor se decide que la máquina virtual no es suficiente como entorno externo para dejar la instancia del BOT activa debido a la seguridad y la utilidad. Por este motivo, se propone el uso de DOCKER[2], investigado anteriormente, junto con FASTAPI[4].

En la última mitad del sprint se consiguió esta tarea satisfactoriamente, pudiendo dejar en funcionamiento el BOT continuamente y con toda la funcionalidad correctamente implementada.

Dificultades En cuanto al desarrollo de la memoria, este se vio ralentizado por la inexperiencia en LaTeX por parte del alumno, lo cual supuso la necesidad de una búsqueda de tutoriales y referencias externas a medida que se iba escribiendo en el documento.

En cuanto a la aplicación, la instalación en el contenedor DOCKER del proyecto llevó consigo algunos problemas menores debidos a la inexperiencia del desarrollador con el mismo, además de volver a tener que solucionar de la misma manera la dependencia con FFMPEG, la cual requirió de una instalación especial frente a otras librerías.

2.4.7 Sprint 7: Continuación de la memoria

Duración Del 27 de septiembre al 10 de octubre

Objetivos principales Puesta a punto de los puntos que se han de desarrollar y planteamiento de un *planning* correspondiente, así como el avance correspondiente al plan planteado.

Actividades Se aclara con el tutor el esquema global de la memoria, pasando por los puntos que deben estar desarrollados. Tras la reunión, se elabora un *planning* lo más equilibrado posible con algo de holgura para imprevistos que puedan surgir para los puntos que faltan por desarrollar de la memoria.

Se desarrollan al completo los capítulos de Metodología y Planificación, (salvo los sprints sin llevar a cabo), Requisitos y por último las Soluciones Existentes. Cada vez se va haciendo más ligera la escritura puesto que la escala de aprendizaje en LaTeX se hace presente, así como hacerse al esquema de la memoria.

Dificultades No hubo ninguna reseñable

2.4.8 Sprint 8: Fin del proyecto

Duración Del 10 al 24 de octubre

Objetivos principales Terminar la memoria y comprobar el estado actual la aplicación.

Actividades El desarrollo de la memoria llega a su fin, redactando los capítulos de Marco Conceptual, Descripción de la Solución, Análisis, Diseño e Implementación. Esta redacción ya se produjo de una manera más

fluida debido a la curva de aprendizaje de LaTeX, así como una mayor comodidad y conocimiento de proyecto. Por lo que, pese a que este es el sprint con más carga de trabajo basándose en cantidad de contenido de memoria desarrollado, estuvo relativamente equilibrado en tiempo comparándose con el anterior sprint.

También se realizan unos últimos retoques en la aplicación, ocultando la token del BOT en una variable de entorno para esquivar así un fallo importante en seguridad de cara al repositorio público, así como una pequeña reorganización del código.

Por último, se realizan unas pruebas de toda la funcionalidad para comprobar que todo sigue correctamente, completando el capítulo de Pruebas y finalmente el de Conclusión.

Dificultades Durante las pruebas se detectó que la funcionalidad de autodetección de idioma de la API de MYMEMORY esta deprecada tras una actualización hecha hace un mes, y al ser esta nuestra opción por defecto de traducción para usuarios nuevos, tiene suficiente importancia para intentar comunicarnos con ellos a través de su foro. Finalmente solucionaron el error al poco tiempo de publicarlo, por lo que no hubo que replantear ni cambiar nada de nuestro proyecto y se pudo dar por terminado.

2.5 Costes

En esta sección discutiremos todos los costes que conlleva el proyecto desde su inicio a su fin. Distinguiremos entre costes de personal, de hardware, de software e indirectos.

2.5.1 Personal

El equipo a remunerar para este proyecto está compuesto por un único miembro que ejerce tanto de jefe de proyecto como de programador. Tras una rápida investigación podemos observar que el sueldo de un jefe de proyecto en el sector informático[6] es mayor que el sueldo medio de un programador PYTHON[20] como cabría esperar, por lo que tomaremos el primero para el cálculo del coste.

Dado que este sueldo es de 22,67 por hora, y puesto que el proyecto al completo se compone de 300 horas de trabajo, se multiplicarán por esta cifra para obtener el total.

Rol	Coste estimado
Jefe de proyecto	6.801 €
Total	**6.801 €**

Cuadro 2.2: Tabla de costes de personal

2.5.2 Hardware

Ubicaremos en este apartado el coste estimado del equipo físico que se ha usado para el desarrollo al completo de este proyecto. Dado que el tiempo de vida del proyecto ha sido de 82 días, y tomando un tiempo de vida de hardware medio en un año, calcularemos el coste en el proyecto equivalente a dicho tiempo.

Componente	Coste original	Coste en el proyecto
PcCom Bronze MOBA i5-8600K / 16GB / 1TB / RTX3050	1100 €	247 €
Monitor AOC Gaming 24G2U 24”	200 €	45 €
Teclado Krom Kernel TKL	40 €	9 €
Ratón Logitech G203	25 €	6 €
Total	1.365 €	**307 €**

Cuadro 2.3: Tabla de costes de hardware

2.5.3 Software

El coste de software es de 0€ debido a que todo el software empleado en este proyecto es totalmente gratuito.

2.5.4 Indirectos

Esta sección comprende aquellos costes ajenos al desarrollo directamente, es decir servicios básicos y gastos menores. Estos números hacen referencia al periodo comprendido en los 5 meses de proyecto. Además, en el caso de gastos que dependen de la cantidad de uso se ha reducido en función de las horas semanales.

Concepto	Coste estimado
Teléfono e Internet	240 €
Electricidad, agua y calefacción	160 €.
Total	**400 €**

Cuadro 2.4: Tabla de costes indirectos

2.5.5 Total

Teniendo en cuenta los costes recién mencionados:

Tipo	Coste
Personal	6.801 €
Hardware	307 €
Software	0 €
Indirecto	400 €
Total	**7.508 €**

Cuadro 2.5: Tabla de coste total

2.6 Análisis de riesgos

Un riesgo en proyectos es, citando el PM-BOK[13], un evento o condición incierta que, en caso de que ocurra, tiene un efecto positivo o negativo sobre al menos un objetivo del proyecto, llámese tiempo, costo, alcance o calidad.

El objetivo de este análisis es el de identificar, analizar y proponer una medida de tratamiento antes de que puedan llegar a ocurrir para de esta forma reducir el potencial impacto que pueda tener en la ejecución del proyecto.

Para esto, emplearemos dos unidades de medición principales; la probabilidad y el impacto. Dividiremos sus valores en los intervalos que quedan detallados a continuación:

- **Probabilidad:**

Probabilidad	Rango
Alta	Mayor de 50 % de probabilidad de ocurrencia.
Significante	30-50 % de probabilidad de ocurrencia.
Moderada	10-29 % de probabilidad de ocurrencia.
Baja	Menos de 10 % de probabilidad de ocurrencia.

Cuadro 2.6: Intervalos de probabilidad de los riesgos

- **Impacto:**

Impacto	Rango
Alta	Más de 30 % por encima del presupuesto.
Significante	20-29 % por encima del presupuesto.
Moderada	10-19 % por encima del presupuesto.
Baja	Menos de 10 % por encima del presupuesto.

Cuadro 2.7: Intervalos de impacto de los riesgos

Principales riesgos detectados junto a sus medidas:

R001	Fallo en la estimación
Descripción	Debido a que la estimación es una aproximación, es posible que no se sigan las fechas señaladas, si no se sigue la estimación afectará a la duración final del proyecto
Probabilidad	Moderada.
Impacto	Significante.
Acciones de mitigación	Hacer una estimación algo holgada conociendo los posibles imprevistos que pueden suceder

Cuadro 2.8: R001. Fallo en la estimación

R002	Desconocimiento de tecnologías
Descripción	Debido a que el proyecto se desarrollará en tecnologías nuevas para el desarrollador, es posible que afecte a la duración final del proyecto
Probabilidad	Significante.
Impacto	Baja.
Acciones de mitigación	Investigación previa de tutoriales y guías existentes en la materia

Cuadro 2.9: R002. Desconocimiento de tecnologías

R003	Sinergia de tecnologías
Descripción	Las tecnologías a usar deben funcionar juntas como se espera, factor que no se conoce al completo hasta que se desarrolle.
Probabilidad	Baja.
Impacto	Significante.
Acciones de mitigación	Tener este factor en cuenta en la estimación de los sprints que toquen estas sinergias y en la investigación previa de las tecnologías

Cuadro 2.10: R003. Sinergia de tecnologías

R004	Dependencias con APIs externas
Descripción	Parte de la funcionalidad emplea APIs externas cuyo mantenimiento no depende de este proyecto, por lo que existe la posibilidad de que queden inoperativas o defectuosas en un punto de la vida del mismo
Probabilidad	Baja.
Impacto	Alta.
Acciones de mitigación	Estudiar alternativas para modificar la implementación de la mejor manera posible y así no depender tanto de estas APIs

Cuadro 2.11: R004. Dependencias con APIs externas

R005	Cambio en los requisitos por parte del cliente.
Descripción	Debido a que los requisitos del proyecto están definidos por el cliente, la alteración o confusión en alguno de los requisitos puede suponer un cambio en duración final de proyecto
Probabilidad	Moderada.
Impacto	Significante.
Acciones de mitigación	Planificar reuniones para discutir los posibles requisitos que presenten algún tipo de ambigüedad.

Cuadro 2.12: R005. Cambio en los requisitos por parte del cliente

R006	Disponibilidad del desarrollador.
Descripción	El desarrollador puede ver afectada su disponibilidad por motivos personales o de salud por lo tanto afectaría a la duración final del proyecto.
Probabilidad	Moderada
Impacto	Significante.
Acciones de mitigación	Planificar las tareas según su importancia y dejar tiempo de colchón entre ellas por posibles retrasos.

Cuadro 2.13: R006. Disponibilidad del desarrollador.

R007	Pérdida de ficheros.
Descripción	Pérdida o alteración errónea de algún archivo importante para el proyecto
Probabilidad	Baja
Impacto	Alto.
Acciones de mitigación	Guardar el proyecto en un repositorio remoto a parte de en las diferentes instancias locales que se posean.

Cuadro 2.14: R007. Pérdida de ficheros

R008	Máquina virtual UVa.
Descripción	La caída del servidor de la UVa o un error en su administración podría llegar a dejar inaccesible el proyecto
Probabilidad	Baja
Impacto	Alto.
Acciones de mitigación	Poseer el proyecto en una máquina local listo para funcionar aunque esto suponga el uso de la propia máquina como servidor.

Cuadro 2.15: R008. Máquina virtual UVa

R009	Incumplimiento de las expectativas del cliente.
Descripción	El diseño o el proyecto en sí no cumple con las expectativas del cliente
Probabilidad	Moderada
Impacto	Moderado.
Acciones de mitigación	Establecer una comunicación con el cliente de calidad y asegurarse de que en estas reuniones quede claro el estado actual y futuro del proyecto.

Cuadro 2.16: R009. Incumplimiento de las expectativas del cliente

2.6.1 Riesgos materializados

A lo largo del proyecto se han dado alguno de los riesgos definidos en cierta medida, y aunque no han supuesto un duro golpe en el alcance del proyecto ya que todos los objetivos principales han sido cumplidos, sí que han cerrado las puertas a una mayor profundidad o mejoras adicionales en partes de la aplicación. Estas posibilidades que finalmente no vieron la luz en nuestro proyecto quedan descritas en la sección de Trabajo Futuro 10.1

Riesgos analizados que han sucedido:

- **R001 - Fallo en la estimación:** La funcionalidad planificada para el sprint 4 2.4.4 finalmente no pudo completarse dentro del tiempo establecido, por lo que hubo que pasar parte del esfuerzo estimado al siguiente sprint. Gracias a la mitigación del requisito combinando la holgura de ambos sprints se pudo remontar el curso del proyecto.

- **R002 - Desconocimiento de tecnologías:** Principalmente este riesgo se ha dado en el lanzamiento de la aplicación desde la máquina virtual y DOCKER, que ha conllevado una investigación exhaustiva para llegar a la solución implementada finalmente. También destacamos la ralentización inicial sufrida en la escritura de la memoria, fruto del desconocimiento de LATEX por parte del alumno.

- **R004 - Dependencias con APIs externas:** Tal y como se describe en las dificultades del último sprint 2.4.8, tras una actualización de la API de MYMEMORY, empleada para la traducción en nuestro proyecto, una de las funcionalidades que utilizamos se quedó deprecada, dejando inerte parte de nuestra aplicación. Pese a que existían alternativas tal y como estaba planeado en la mitigación del riesgo, se decidió esperar a si los responsables del mantenimiento de la API lo solucionaban antes de que afectase en gran medida a la estimación, finalmente esta decisión fue la correcta ya que al poco tiempo lo solucionaron y no hubo que recurrir al plan b.

Capítulo 3

Soluciones Existentes

Existen BOTS de TELEGRAM que se aproximan al objetivo principal de nuestro proyecto, proporcionando partes separadas de la funcionalidad que nosotros implementamos como bloque. Por este motivo y dado que han servido como herramienta para aportarnos una visión global y la posterior formación de un concepto inicial, las mencionaremos en esta sección.

Dividiremos estas menciones en BOTS de traducción, *text-to-speech* y *speech-to-text*:

3.1 Traducción

Figura 3.1: Logo de YANDEX.TRANSLATE

YANDEX.TRANSLATE[34] incluye una traducción usando una de las APIs básicas, detectando el idioma del mensaje que recibe, pero pudiendo cambiar manualmente tanto el idioma a detectar como al que se traduce.

Figura 3.2: Logo de LANGUAGE TRANSLATOR

LANGUAGE TRANSLATOR[11] incluye además la posibilidad de cambiar el motor de traducción y visualizar la lista de idiomas que soporta junto al código necesario para elegir cada uno. Bajo mi punto de vista tiene buena funcionalidad pero su uso general es poco práctico para el usuario.

Figura 3.3: Logo de TGTRANSLATOR

TGTRANSLATOR[3] tiene un funcionamiento similar a YANDEX.TRANSLATE, salvo que añade un modo para traducir todos los mensajes que se escriban en el grupo sin uso de comandos.

Figura 3.4: Logo de LINGVANEX TRANSLATOR

LINGVANEX TRANSLATOR[12] es un proyecto ambicioso implementado no solo en TELEGRAM y que agrega lo que comentaremos a continuación, el *text-to-speech*, pero no combina ambas funcionalidades.

3.2 *Text-to-speech*

Figura 3.5: Logo de TEXT TO SPEECH BOT

TEXT TO SPEECH BOT[19], como su propio nombre indica, convierte texto a mensajes de voz en once idiomas diferentes. Como toque adicional se ha cuidado la interfaz agregando botones para la selección de idioma.

3.3 *Speech-to-text*

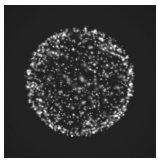

Figura 3.6: Logo de TRANSCRIBER BOT

TRANSCRIBER BOT[21] transforma a texto tanto mensajes de voz como imágenes, pudiendo elegir entre una cantidad considerable de idiomas. Tras probarlo he de destacar que la detección de voz en español funciona extremadamente bien.

Figura 3.7: Logo de SILERO STT

SILERO STT[15] divide tu mensaje de voz en diferentes fragmentos que posteriormente transcribe a texto anotando el intervalo de segundos en el que se dice esa frase. Es un funcionamiento bastante curioso, aunque solo permite audios de corta duración, y por lo que he podido observar, toda la interfaz y detección de idioma está en ruso, por lo que está bastante limitada al uso.

Figura 3.8: VOICOS en GITHUB

Por último quería mencionar el proyecto VOICOS[8], el cual emplea GOOGLE CLOUD SPEECH[7], siendo una herramienta bastante potente y confiable de detección. Destaco este proyecto ya que me he basado en él para la descarga y puesta a punto del mensaje de voz recibido por TELEGRAM. Lamentablemente VOICOS ya no está en marcha debida a su dependencia de pago a GOOGLE.

Capítulo 4

Marco Conceptual

En este capítulo trataremos algunos conceptos necesarios para el entendimiento de la memoria, así como una descripción de como se han enfocado dichos conceptos en el proyecto.

4.1 Conceptos teóricos

4.1.1 BOTS conversacionales

Los BOTS conversacionales[23], más conocidos por su traducción al inglés *chatbots*, son aplicaciones software que surgen en los años 60, y que simulan mantener una conversación con una persona al proveer respuestas automáticas, las cuales son previamente establecidas por un conjunto de expertos a entradas realizadas por el usuario.

Habitualmente, la conversación se establece mediante texto, aunque también hay modelos que disponen de una interfaz de usuario multimedia que permiten la entrada y salida auditiva, dotando de mayor realismo a la interacción con el usuario.

Para establecer una conversación, han de utilizarse frases fácilmente comprensibles y que sean coherentes, aunque la mayoría de los BOTS conversacionales no consiguen comprender del todo. En su lugar, tienen en cuenta las palabras o frases del interlocutor, que les permitirán usar una serie de respuestas preparadas de antemano. Estos son capaces de reconocer la manera en la que una frase está formulada gracias a una serie de patrones comparativos preestablecidos, y de este modo, basándose en las diferentes variables de dicha frase, presentan una respuesta correspondiente.

Antes la creación de un BOT conversacional decente suponía una gran inversión en recursos para su dificultosa programación. Sin embargo, la mejora en el desarrollo y modulación de las librerías de vocabulario y los algoritmos de inteligencia artificial, están simplificando su elaboración. Los BOTS actualmente son una forma sencilla y útil de implementar una funcionalidad y ponerla a mano de un usuario sin necesidad de ningún conocimiento previo por parte de esa persona.

4.1.2 BOTS en TELEGRAM

TELEGRAM no solo es una de las aplicaciones de mensajería más usadas globalmente, si no que tiene un sistema de creación y uso de BOTS bien asentado entre su comunidad que se remonta a 5 años atrás, existiendo una gran variedad de BOTS conversacionales que implementan todo tipo de utilidades para los usuarios. Además, TELEGRAM es multidipositivo, ya que dispone tanto de versiones de escritorio, web y para dispositivos móvil, pudiendo cómodamente acceder desde cualquiera y proporcionando una mayor cobertura a los usuarios.

Estos BOTS no necesitan instalación ni nada extraordinario en tu dispositivo, los usuarios únicamente deberán buscar el nombre del BOT como si de otro usuario cualquiera se tratase y seguir los pasos que aparecerán en la conversación privada o grupal que se haya creado.

Para que hagan lo que uno quiera, simplemente deberá usar los comandos *buit-in* que existen los cuales se pueden ver tanto usando el comando genérico `/help` como en el autocompletado del chat. Hay que tener en cuenta que todos los BOTS siguen una serie de directrices generales marcadas por TELEGRAM para que su uso sea similar entre todos y lo más intuitivo posible, además de implementar ciertas medidas de seguridad y privacidad.

4.1.3 TAC

La Traducción asistida por computadora (TAC)[33], es el proceso de reproducción de un texto en otra lengua que lleva a cabo una persona con la ayuda de software o programas de ordenador desarrollados específicamente a tal fin. Este término abarca diferentes tipos de herramientas o aplicaciones informáticas específicas como, por ejemplo, las que crean y organizan memorias de traducción y los editores de recursos interactivos de software de tipo textual.

Desde la aparición del ordenador en los años 40, siempre hubo distintas corrientes que veían en él una solución para la comunicación entre diferentes culturas, pero inicialmente se desestimó la idea de que algo de tal calibre fuera posible. Finalmente tras el desarrollo de la memoria de traducción sobre los años 70, que consiste en poder disponer de una base datos donde consultar traducciones previas de determinados términos o frases, iguales o similares a aquellos que quieren traducirse, se empezaron a ver las primeras aproximaciones a este proceso.

La estrategia principal y genérica de los sistemas de traducción asistida por computadora estriba en el acceso a las memorias de traducción, para su reutilización. Por una parte, se reutilizan los elementos traducidos y revisados por un humano y, por otra, se recuperan los elementos traducidos con un índice de coincidencia variable mediante la técnica de lógica difusa. Asimismo, estos sistemas cuentan, por regla general, con un gestor de terminología que permite gestionar otro tipo de recursos lingüísticos, los diccionarios. Por este motivo, estas herramientas pueden considerarse como gestores de recursos lingüísticos que pueden ser reutilizados para diferentes proyectos o para otras herramientas diferentes a los de su entorno de creación.

Estas herramientas son sistemas de información que dan soporte al almacenamiento, navegación, extracción y creación de recursos lingüísticos del tipo memorias de traducción y diccionarios terminológicos.

4.1.4 Reconocimiento del habla

El reconocimiento del habla [32] es un campo de la informática y la lingüística computacional que desarrolla metodologías y tecnologías que permiten el reconocimiento y la traducción del lenguaje hablado a texto por parte de los ordenadores con la principal ventaja de la capacidad de búsqueda. También se conoce como *automatic speech recognition* (ASR), *computer speech recognition* o *speech to text* (STT).

Desde el punto de vista tecnológico, el reconocimiento del habla tiene una larga historia de grandes innovaciones. Más recientemente, el campo se ha beneficiado de los avances en el *deep learning* y el *big data*. Los avances se evidencian no solo por la oleada de artículos académicos publicados en este campo, si no, lo que es más importante, por la adopción por parte de la industria mundial de una variedad de métodos de *deep learning* en el diseño e implementación de sistemas de reconocimiento del habla.

4.1.5 Conversión Texto a Voz

La conversión texto a voz (TTS, del inglés *Text To Speech*) [30] se centra en la generación de habla artificial humana a partir de texto. El sistema computacional que es usado con este propósito es llamado computadora de habla o sintetizador de voz y puede ser implementado en productos software o hardware.

El habla sintetizada puede ser creada a través de la concatenación de fragmentos de habla grabados que son almacenados en una base de datos. Los sistemas difieren en el tamaño de las unidades de habla almacenadas; un sistema que almacena *fonos* y *difonos*, es decir fragmentos que dividen una misma palabra como pueden ser el sonido de una única letra o una sílaba, permite un mayor rango de sonidos pero carece de claridad. Para usos específicos, el tamaño del almacenamiento de palabras completas u oraciones permite una mayor calidad de audio. De manera alternativa, un sintetizador puede incorporar un modelo de tracto vocal u otras características de la voz humana para recrear completamente una voz sintética.

Un sistema o motor de *text to speech* (TTS) está compuesto de dos partes: un *front-end* y un *back-end*. El *front-end* tiene dos tareas principales. Primero, convertir el texto con caracteres, números, símbolos y abreviaciones en su equivalente en palabras escritas. Posteriormente el *front-end* asigna una transcripción fonética a cada palabra, marca y divide el texto en unidades prosódicas, como frases, cláusulas y oraciones. Dando como resultado final al concatenarse una cadena de audio, es decir el habla.

Capítulo 5

Descripción de la solución

Debido a la complejidad de los conceptos que hemos descrito en la sección anterior, en este proyecto no se desarrolla una aplicación autónoma donde la implementación corre completamente a cuenta del desarrollador, si no una aplicación en la que existen dependencias con APIs externas que implementan servicios los cuales realizan parte de estas tareas por nosotros, y por tanto tienen un modo de funcionar establecido que ha de ser satisfecho. Por este motivo existe este capítulo, para dar a entender a grandes rasgos las principales cuestiones a afrontar y qué enfoques se han tomado en la solución final.

5.1 Creación del BOT

Explicación del problema: Además del desarrollo de la aplicación que aportará la funcionalidad deseada, tenemos que crear un usuario BOT en TELEGRAM con el cual los demás usuarios puedan comunicarse, meter en grupos, etc. Este BOT debe tener un nombre descriptivo, así como una imagen y descripción que lo acompañen. Por último también debemos generar un token que nos permita comunicarnos con él como su creador y de este modo poder aportarle su funcionalidad.

Solución implementada: Esta tarea es muy sencilla y a mano de cualquier usuario de TELEGRAM, gracias al BOTFATHER[16], un BOT cuya función es precisamente la de aportar al usuario todas las herramientas necesarias para crear un BOT por su cuenta contando además con una interfaz gráfica por medio del uso de botones en el chat que hacen aún más cómoda la interacción gracias al formato de menú que aporta. Simplemente seleccionando las opciones que te muestra el BOTFATHER se consigue cumplir todo lo mencionado sin ningún problema. Para más información seguir esta introducción[17].

Figura 5.1: BOT de TELEGRAM BOTFATHER

5.2 Comunicación con la API de TELEGRAM

Explicación del problema: Para tener una visión rápida de esto solo hay que imaginarse el modo de funcionamiento de un BOT, es decir, un intercambio de mensajes en un chat ya sea privado o grupal de TELEGRAM. Normalmente los mensajes por parte del usuario suelen contener un comando para que el BOT sepa que debe hacer algo y cuál de sus funcionalidades debe llevar a cabo en ese momento, aunque esto tampoco es cierto siempre. Esto nos lleva a dos principales necesidades de la aplicación:

- **Continuidad:** La aplicación debe estar en actividad y comunicación constante con la API preparada para ejecutar una acción que un usuario cualquiera desee llevar a cabo en ese momento.
- **Detección de acciones del usuario:** Una vez un mensaje llega al BOT, éste debe saber si activa alguna de sus funcionalidades; ya sea por medio de un comando, un formato determinado de entrada, etc.

Figura 5.2: Logo de PYTHON-TELEGRAM-BOT

Solución implementada: La librería de PYTHON PYTHON-TELEGRAM-BOT [1] es un *wrapper* que contiene clases y funciones que hacen el desarrollo de BOTS y la comunicación con la API de TELEGRAM BOT mucho más sencillo y directo.

Esta librería es la base de este proyecto, dando solución al problema con la única contra de reducir en cierto grado la legibilidad y comprensión del código, ya que al implementar más funcionalidad de la que necesitamos en nuestro caso, existen más clases y contenedores de los que en primera instancia harían falta. Esto no es un problema mayor ya que tienen nombres explicativos y un uso bastante intuitivo dentro de lo que se podría esperar.

5.3 TAC

Explicación del problema: Debemos implementar un traductor de texto confiable y a poder ser gratuito que a su vez tenga una compatibilidad lo suficientemente simple con PYTHON, así como una manera de que el usuario elija los idiomas que se emplearan en la traducción. De no existir nada similar, habría que crear una funcionalidad relativamente compleja que extendería la carga de trabajo considerablemente.

Además, en caso de encontrar varias soluciones, el traductor debe ser soportar la mayor cantidad de idiomas posible y dar la traducción más fiel posible.

Solución implementada: Tras probar diferentes librerías y traductores, finalmente llegamos a la conclusión de que la opción que mejor cumplía nuestros requisitos era la librería TRANSLATE[18], que tras importarla nos permite simplemente crear un traductor que tiene la capacidad de seleccionar idiomas en formato *IETF language tag*[26] y emplear diversas APIs de traducción conocidas como MICROSOFT TRANSLATION, MYMEMORY, LIBRETRANSLATE y DEEPL.

Figura 5.3: Logo de MYMEMORY

Después de emplear una misma batería de *tests* en las APIs gratuitas la que mejor resultados dio fue MYMEMORY, el cual se puede usar libremente en su página web[22]. Esta API permite la traducción de una cantidad considerable de idiomas dando unos resultados increíblemente certeros en todos los *tests* que se realizaron.

5.4 Voz a texto y texto a voz

Explicación del problema: Al igual que en el problema anterior, no encontrar una herramienta externa confiable supondría un desarrollo de una gran carga y en este caso un producto de calidad mediocre, por lo que la importancia de emplear una librería confiable es de prioridad absoluta.

Además, habrá que adaptarse a los formatos de audio y funcionamiento que tengan las herramientas y, a mayores, se hace presente el problema de la compatibilidad entre ellas y la solución implementada para la traducción, ya que deberán de trabajar juntas para llevar a cabo la funcionalidad principal de esta aplicación.

Solución implementada: Para el reconocimiento de audio se usa la librería SPEECHRECOGNITION[32], la cual tiene compatibilidad con un gran número de APIs como son CMU SPHINX, GOOGLE SPEECH RECOGNITION, GOOGLE CLOUD SPEECH, WIT.AI, MICROSOFT BING VOICE RECOGNITION, HOUNDIFY, IBP SPEECH TO TEXT y SNOWBOY HOTWORD DETECTION.

Entre estas APIs, solo unas pocas son gratuitas y sin límite de uso, además de que al ser una tarea tan exigente, la mayoría dejan mucho que desear o tienen una gran limitación en idiomas. Por suerte para nosotros,

gracias a esta librería podemos usar la API de GOOGLE SPEECH RECOGNITION de forma gratuita, la cual funciona extremadamente bien y además es compatible con el formato de idiomas que usamos para la traducción.

Como último detalle de esta librería, destacamos el uso de otra librería auxiliar llamada PYDUB[10] para solventar el problema de adaptarse a los formatos de audio empleados por SPEECHRECOGNITION, pudiendo transformar el formato obtenido por los mensajes de voz de TELEGRAM (*.ogg*) en el formato deseado (*.wav*) gracias a FFMPEG[5]. Para resumirlo, primero se descarga el mensaje de voz de TELEGRAM mediante el *Updater* en formato *.ogg*, tras esto, se crea un *AudioSegment* de la librería PYDUB introduciendo dicho archivo. Esta clase contiene múltiples funciones para el cambio de formatos, por lo que empleamos la función para cambiar de *.ogg* a *.wav* y tras esto introducimos el *.wav* a una clase *AudioFile* procedente de la misma librería y está listo para ser usado por el *Recognizer* de SPEECHRECOGNITION.

En cuanto a la creación de audios lo perfecto era usar también la API de GOOGLE, y afortunadamente de nuevo, existe una librería, en este caso exactamente para emplear GOOGLE TEXT-TO-SPEECH API, llamada GTTS[14]. Nuevamente, nos viene como anillo al dedo ya que acepta como parámetros los idiomas en el mismo formato con el que estábamos trabajando, dando los resultados esperados de manera idéntica a las funciones que implementa GOOGLE en su traductor por ejemplo.

Como conclusión, hemos usado dos APIs de GOOGLE para solventar con gran calidad estos problemas únicamente costándonos un pequeño rodeo para adaptar el formato del audio recibido por TELEGRAM ya que se trabaja con el mismo formato de idioma durante toda la aplicación.

Capítulo 6

Requisitos

En este capítulo indicaremos los diferentes requisitos de la aplicación, es decir, una descripción lo más completa que sea posible del sistema en forma de objetivos, capacidades y características que posee.

Diferenciamos:

- **Requisitos funcionales:** Funciones que debe implementar la aplicación.
- **Requisitos no funcionales:** Cualidades, restricciones y características del sistema.
- **Requisitos de información:** Datos conocidos y almacenados con los que se trabaja.

6.1 Requisitos funcionales

Dado que cada función que implementa un BOT está encapsulada en un comando, usaremos el propio comando como nombre de requisito para una mayor claridad.

Código	Requisito	Descripción
RF-01	*start*	La aplicación devolverá un mensaje de bienvenida al usuario refiriéndose por su nombre donde se describe de forma simple las posibilidades del BOT.
RF-02	*help*	Se muestra la lista de comandos al completo con su formato de uso y una breve descripción de su utilidad.
RF-03	*echo*	Recibe una cadena de texto por parte del usuario y la devuelve sin ninguna modificación.
RF-04	*fromlanguage*	Actualiza el idioma desde el que se traducirán las entradas del usuario a partir de su ejecución. Esta función aceptará entradas en formato *IETF language tag*[26].

RF-05	*tolanguage*	Actualiza el idioma al que se traducirán las entradas del usuario a partir de su ejecución. Esta función aceptará entradas en formato *IETF language tag*.
RF-06	*translate*	Traduce el mensaje de texto recibido por el usuario y lo devuelve en formato texto. Por defecto, es decir sin usar los comandos de los dos últimos requisitos, se detectará automáticamente el idioma de entrada y se traducirá al inglés.
RF-07	*fastmode*	Activa o desactiva el modo rápido para el usuario dependiendo de su estado actual. El modo rápido traduce todo lo que escriba el usuario por el chat sin necesidad de emplear el comando *translate*.
RF-08	*toaudio*	Traduce el mensaje de texto recibido a un audio de voz. Por defecto, es decir sin usar los comandos de los dos últimos requisitos, se detectará automáticamente el idioma de entrada y se traducirá al inglés.
RF-09	Traducción mensaje de voz	Funcionalidad principal del BOT. No posee un comando ya que se activará al recibir un mensaje de voz por el chat. Detectará lo que se ha dicho según el idioma escogido mediante *fromlanguage*, el cual en este caso es de uso previo obligatorio, y posteriormente devolverá tanto lo que se ha detectado como un audio de voz traducido por defecto al inglés.

Cuadro 6.1: Requisitos funcionales

6.2 Requisitos no funcionales

Código	Requisito	Descripción
RNF-01	Lenguaje de programación	Durante el desarrollo se empleará como único lenguaje de programación PYTHON.
RNF-02	Disponibilidad	El sistema estará disponible para su uso las 24 horas del día durante todo el año.
RNF-03	Entorno del usuario	El usuario únicamente interactuará con el BOT mediante TELEGRAM.
RNF-04	Comunicación con API	Se empleará la librería PYTHON-TELEGRAM-BOT[1] como intermediario entre las comunicaciones del código y la API de TELEGRAM.
RNF-05	Multiplataforma	Se podrá usar tanto desde la versión web, como de escritorio y por último de móvil de TELEGRAM
RNF-06	Multiusuario	El sistema debe soportar el uso simultáneo de varios usuarios ya sea desde un grupo o por separado no viéndose afectados entre sí
RNF-07	Intuitivo	El uso del BOT será dado al usuario de la manera más sencilla posible sin requerir ningún conocimiento inicial mayor que el uso de comandos en TELEGRAM
RNF-08	Máquina virtual UVa	Todas las herramientas así como la creación del entorno externo se llevarán a cabo en una máquina virtual con sistema operativo Ubuntu aportado por la UVA
RNF-09	FastAPI	Se construirá un *framework* web mediante FASTAPI[4]
RNF-10	Docker	Se *dockerizará* la aplicación, es decir, se creará un contenedor DOCKER[2] desde el que se lanzará la API

Cuadro 6.2: Requisitos no funcionales

6.3 Requisitos de información

Código	Requisito	Descripción
RI-01	Instancias	Se almacenarán tanto la instancia del BOT como la conexión con cada chat. Ambas se usan en todo momento por la librería.
RI-02	Idioma origen	Idioma del que se parte en la traducción de los comandos. Cada usuario tiene su propia entrada en un diccionario interno de la librería.
RI-03	Idioma destino	Idioma al que se traduce en los comandos. Cada usuario tiene su propia entrada en un diccionario interno de la librería.
RI-04	Modo rápido	Indica si el usuario tiene el modo rápido activado. Cada usuario tiene su propia entrada en un diccionario interno de la librería.

Cuadro 6.3: Requisitos de información

Capítulo 7

Análisis

7.1 Modelo de dominio

La función de este modelo es la de ilustrar una abstracción de las entidades del problema, así como sus atributos y como éstas se relacionan entre sí. Desde el punto de vista del desarrollador, es un esquema orientativo para saber qué debe existir y con que clase de conceptos se va a trabajar.

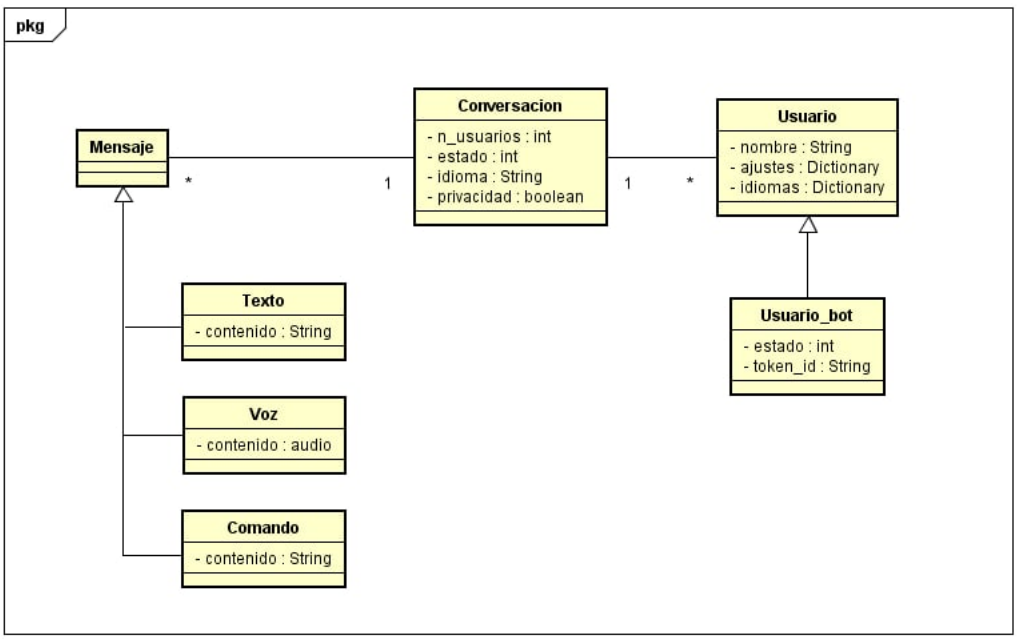

Figura 7.1: Modelo de Dominio

Al trabajar sobre una aplicación de mensajería, nuestro dominio estará compuesto por mensajes, conversaciones y usuarios. Una misma conversación podrá contener varios usuarios dependiendo de si es grupal o privada, pudiendo uno de estos ser el BOT conversacional desarrollado. La interacción entre usuarios dentro

de una conversación se producirá por medio de mensajes, los cuales podrán ser de texto, de voz o comandos reservados para la comunicación de órdenes hacia el BOT.

Un usuario tendrá ajustes personales almacenados, así como los idiomas seleccionados para las diferentes funcionalidades que se ofrecen. Además, el usuario BOT, el cual es único, poseerá un token para identificarse y así poder llevar a cabo sus acciones.

7.2 Casos de uso

En esta sección mostraremos el diagrama de casos de uso, así como una descripción detallada de cada uno de ellos.

7.2.1 Diagrama de casos de uso

Un diagrama de casos de uso es una representación esquemática de la interacción entre el los actores y el sistema para llevar a cabo las diferentes funcionalidades que están implementadas en él.

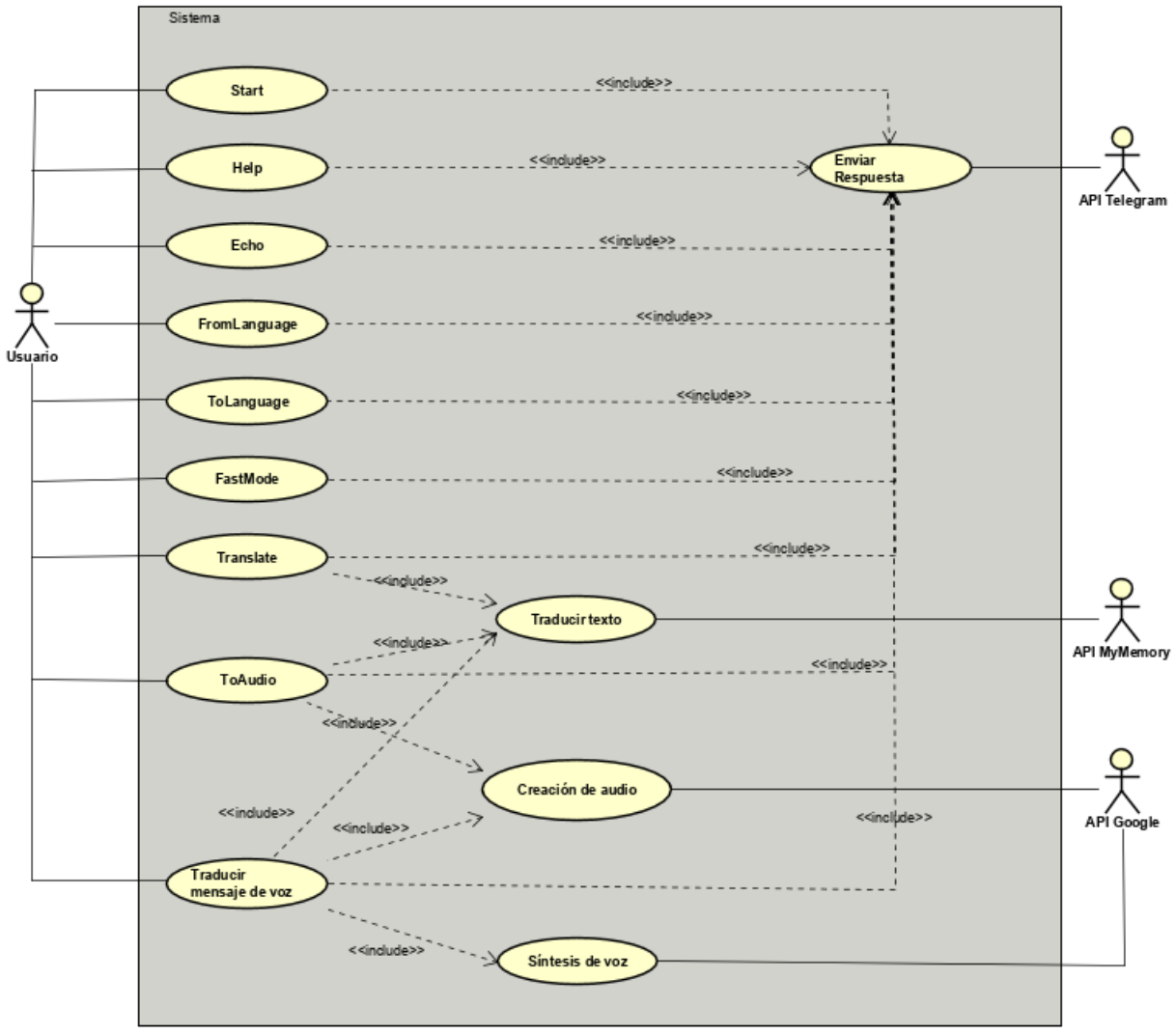

Figura 7.2: Diagrama de casos de uso

7.2.2 Descripción casos de uso

Identificador	CU-01 *Start*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Recibir mensaje de bienvenida.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/start`. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se envía un mensaje de bienvenida por el chat que encamina al usuario en cómo usar el BOT.
Secuencia Alternativa	
Postcondiciones	1. Se ha enviado el mensaje al chat origen.

Cuadro 7.1: CU-01. *Start*.

Identificador	CU-02 *Help*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Recibir lista de comandos.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/help`. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se envía un mensaje que contiene la lista de comandos con una breve descripción de cada uno de ellos.
Secuencia Alternativa	
Postcondiciones	1. Se ha enviado el mensaje al chat origen.

Cuadro 7.2: CU-02. *Help*.

Identificador	CU-03 *Echo*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Replicar un mensaje.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/echo` seguido del mensaje a replicar. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se envía el mismo mensaje que se ha recibido (sin el comando).
Secuencia Alternativa	
Postcondiciones	1. Se ha enviado el mensaje al chat origen.

Cuadro 7.3: CU-03. *Echo*.

Identificador	CU-04 *FromLanguage*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Modificar el idioma origen de las traducciones.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/fromlanguage` seguido de un idioma en formato *IETF language tag*[26]. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se actualiza la entrada correspondiente al idioma origen del usuario en el diccionario y se envía un mensaje informando que ha sido un éxito.
Secuencia Alternativa	3.1 Si no se recibe un idioma se envía un mensaje informando del valor del diccionario actual.
Postcondiciones	1. Se ha enviado el mensaje al chat origen. 2. Se ha modificado el valor del diccionario.

Cuadro 7.4: CU-04. *FromLanguage*.

Identificador	CU-05 *ToLanguage*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Modificar el idioma destino de las traducciones.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/tolanguage` seguido de un idioma en formato *IETF language tag*. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se actualiza la entrada correspondiente al idioma origen del usuario en el diccionario y se envía un mensaje informando que ha sido un éxito.
Secuencia Alternativa	3.1 Si no se recibe un idioma se envía un mensaje informando del valor del diccionario actual.
Postcondiciones	1. Se ha enviado el mensaje al chat origen. 2. Se ha modificado el valor del diccionario.

Cuadro 7.5: CU-05. *ToLanguage*.

Identificador	CU-06 *FastMode*
Actor principal	Usuario.
Actor implicado	API TELEGRAM.
Descripción	Activar o desactivar el modo rápido de traducción.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/fastmode`. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se activa o desactiva el modo rápido haciendo que la entrada del diccionario del usuario correspondiente tome el valor contrario al que tenía y se envía un mensaje de la modificación.
Secuencia Alternativa	
Postcondiciones	1. Se ha enviado el mensaje al chat origen. 2. Se ha modificado el valor del diccionario.

Cuadro 7.6: CU-06. *FastMode*.

Identificador	CU-07 *Translate*
Actor principal	Usuario.
Actor implicado	API TELEGRAM. API MYMEMORY.
Descripción	Traducir mensaje de texto a texto.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/translate` seguido del texto que desee traducir. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se recuperan los valores idioma origen e idioma destino del diccionario del usuario. 4. Se traduce el texto a través de la API de MYMEMORY. 5. Se envía la traducción al chat origen.
Secuencia Alternativa	3.1 Si el usuario tiene activado el modo rápido se detecta como texto a traducir cualquier mensaje entero. 4.1 Si los idiomas no son válidos se envía un mensaje de error y finaliza el caso de uso.
Postcondiciones	1. Se ha enviado el mensaje al chat origen.

Cuadro 7.7: CU-07. *Translate*.

Identificador	CU-08 *ToAudio*
Actor principal	Usuario.
Actor implicado	API TELEGRAM. API MYMEMORY. API GOOGLE.
Descripción	Traducir mensaje de texto a audio.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía `/toaudio` seguido del texto que desee traducir. 2. El sistema detecta el comando y llama a la función correspondiente. 3. Se recuperan los valores idioma origen e idioma destino del diccionario del usuario. 4. Se traduce el texto a través de la API de MyMemory. 5. Se crea un archivo mp3 a partir de la traducción gracias a la API de Google. 6. Se envía el archivo mp3 al chat origen.
Secuencia Alternativa	4.1 Si los idiomas no son válidos se envía un mensaje de error y finaliza el caso de uso.
Postcondiciones	1. Se ha enviado el archivo *mp3* al chat origen.

Cuadro 7.8: CU-08. *ToAudio*.

Identificador	CU-09 Traducir mensaje de voz.
Actor principal	Usuario.
Actor implicado	API TELEGRAM. API MYMEMORY. API GOOGLE.
Descripción	Traducir mensaje voz a audio.
Precondiciones	1. El usuario está en chat privado o grupal con el BOT.
Secuencia Normal	1. El usuario envía un mensaje de voz. 2. El sistema detecta la entrada y llama a la función correspondiente. 3. Se envía el texto detectado tras el reconocimiento. 4. Se recuperan los valores idioma origen e idioma destino del diccionario del usuario. 5. Mediante la API de GOOGLE se realiza la transcripción a texto del mensaje de voz recibido. 6. Se traduce el texto a través de la API de MYMEMORY. 7. Se crea un archivo mp3 a partir de la traducción gracias a la API de GOOGLE. 8. Se envía el archivo *mp3* al chat origen.
Secuencia Alternativa	4.1 Si no hay idioma origen se envía un mensaje de error informando al usuario de que seleccione uno. 5.1 Si el idioma origen no es válido se envía un mensaje de error y finaliza el caso de uso. 6.1 Si el idioma destino no es válido se envía un mensaje de error y finaliza el caso de uso.
Postcondiciones	1. Se ha enviado el texto sintetizado y el archivo *mp3* al chat origen.

Cuadro 7.9: CU-09. Traducir mensaje de voz.

7.3 Diagramas de actividad

Los diagramas de actividad ilustran el flujo de tareas que se realizan en el orden determinado poniendo énfasis en una lectura estructurada y limpia.

En este caso nos centraremos en la interacción entre la aplicación y las APIs con las que se comunica, omitiendo la acción del usuario, dado que siempre será el envío de un comando o mensaje de voz por el chat y reduciría la legibilidad de los diagramas.

7.3.1 *Start*, *Help* y *Echo*

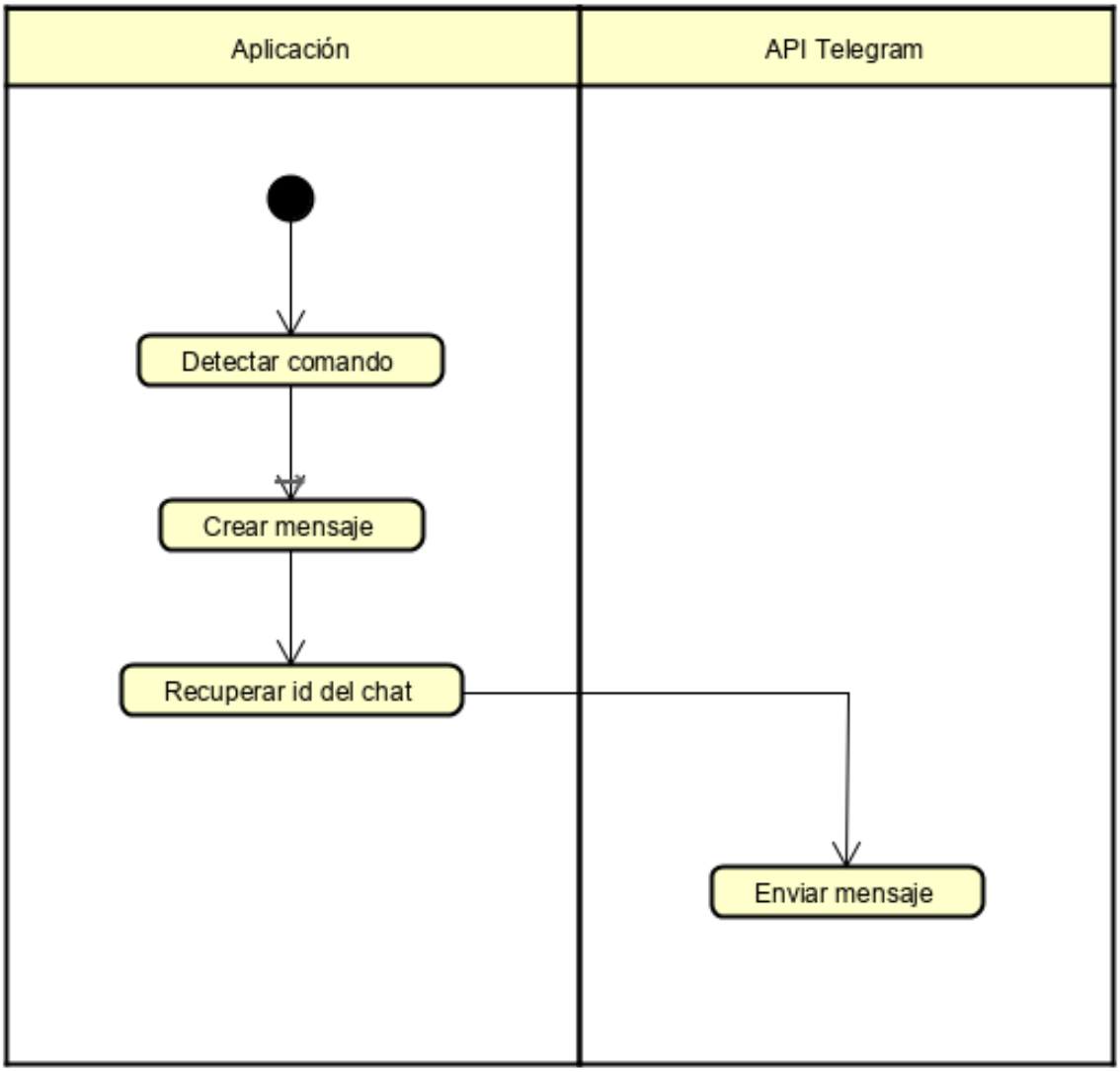

Figura 7.3: Diagrama de actividad: *Start*, *Help* y *Echo*

Estos tres casos de uso comparten flujo, diferenciándose entre sí en la construcción del mensaje de respuesta. En *Start* y *Help* serán mensajes de respuesta predefinidos siendo estos de introducción y una lista de comandos respectivamente, mientras que en el caso de *Echo* el mensaje de respuesta será idéntico al recibido por parte del usuario.

7.3.2 *FromLanguage* y *ToLanguage*

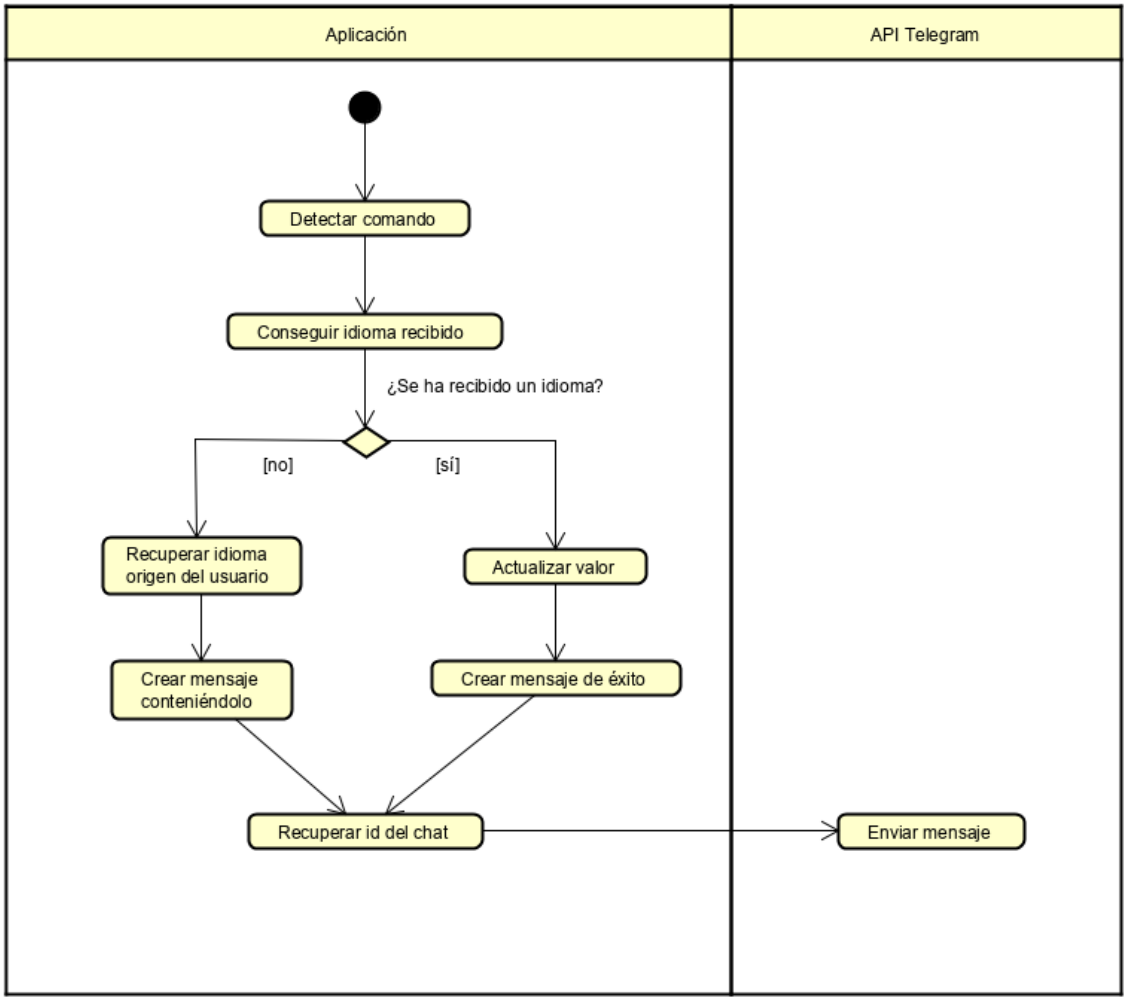

Figura 7.4: Diagrama de actividad: *FromLanguage*

Ambos casos de uso comparten el mismo procedimiento, ya que que son la misma funcionalidad trabajando sobre el idioma origen o destino del usuario.

7.3.3 *FastMode*

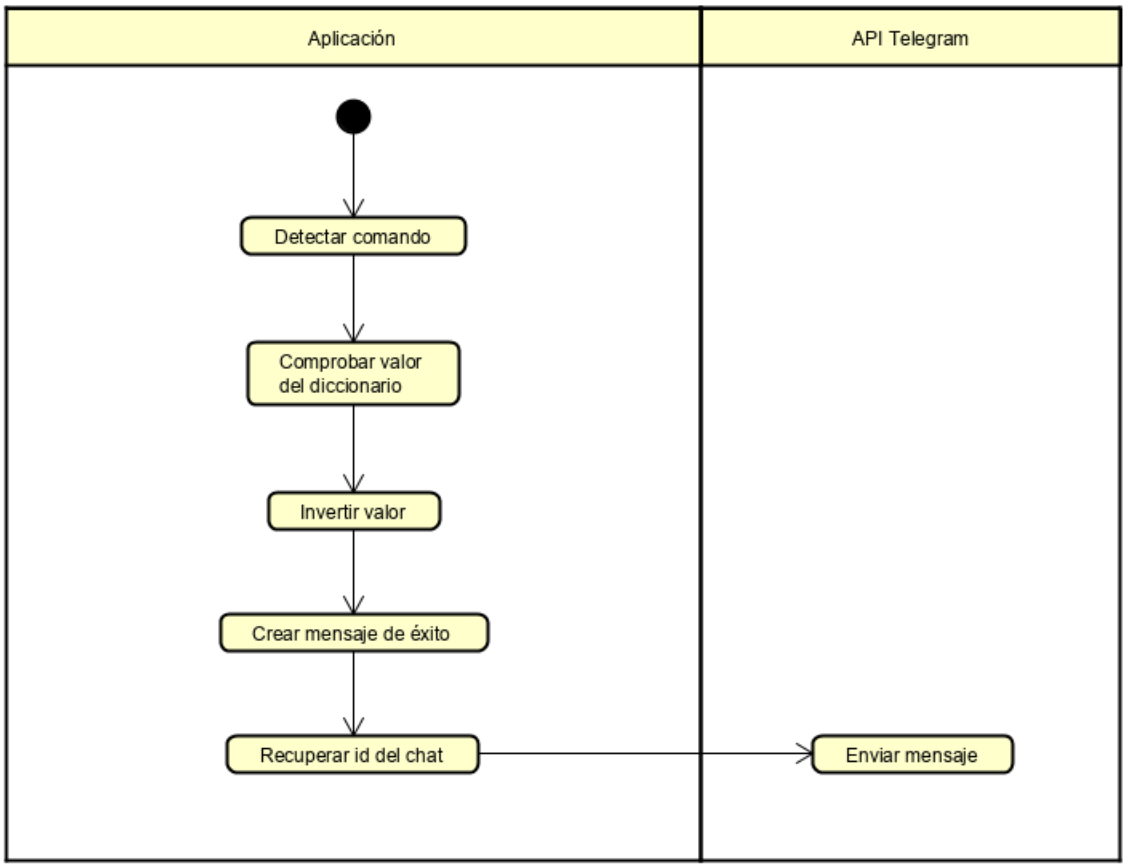

Figura 7.5: Diagrama de actividad: *FastMode*

7.3.4 *Translate*

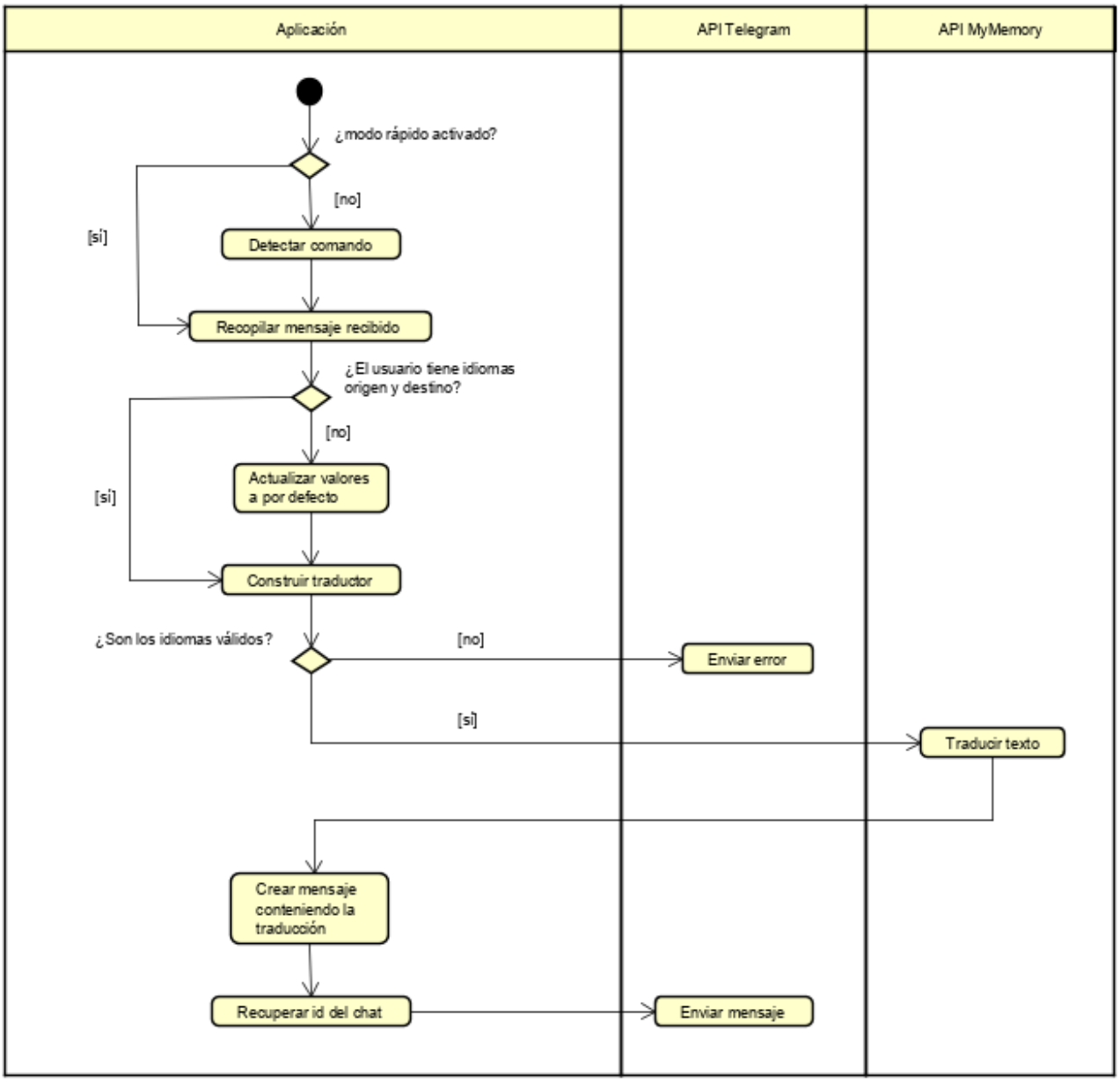

Figura 7.6: Diagrama de actividad: *Translate*

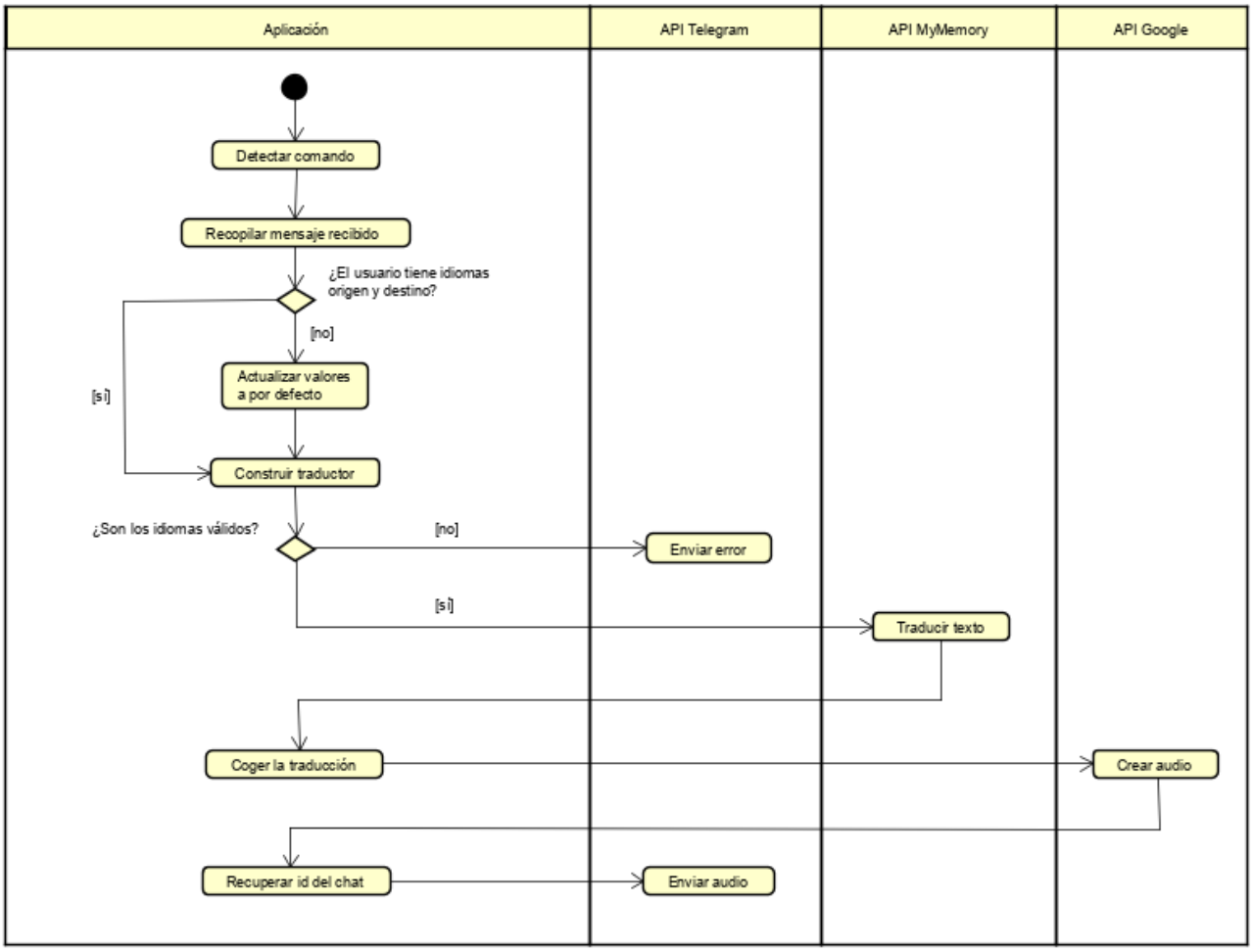

Figura 7.7: Diagrama de actividad: *ToAudio*

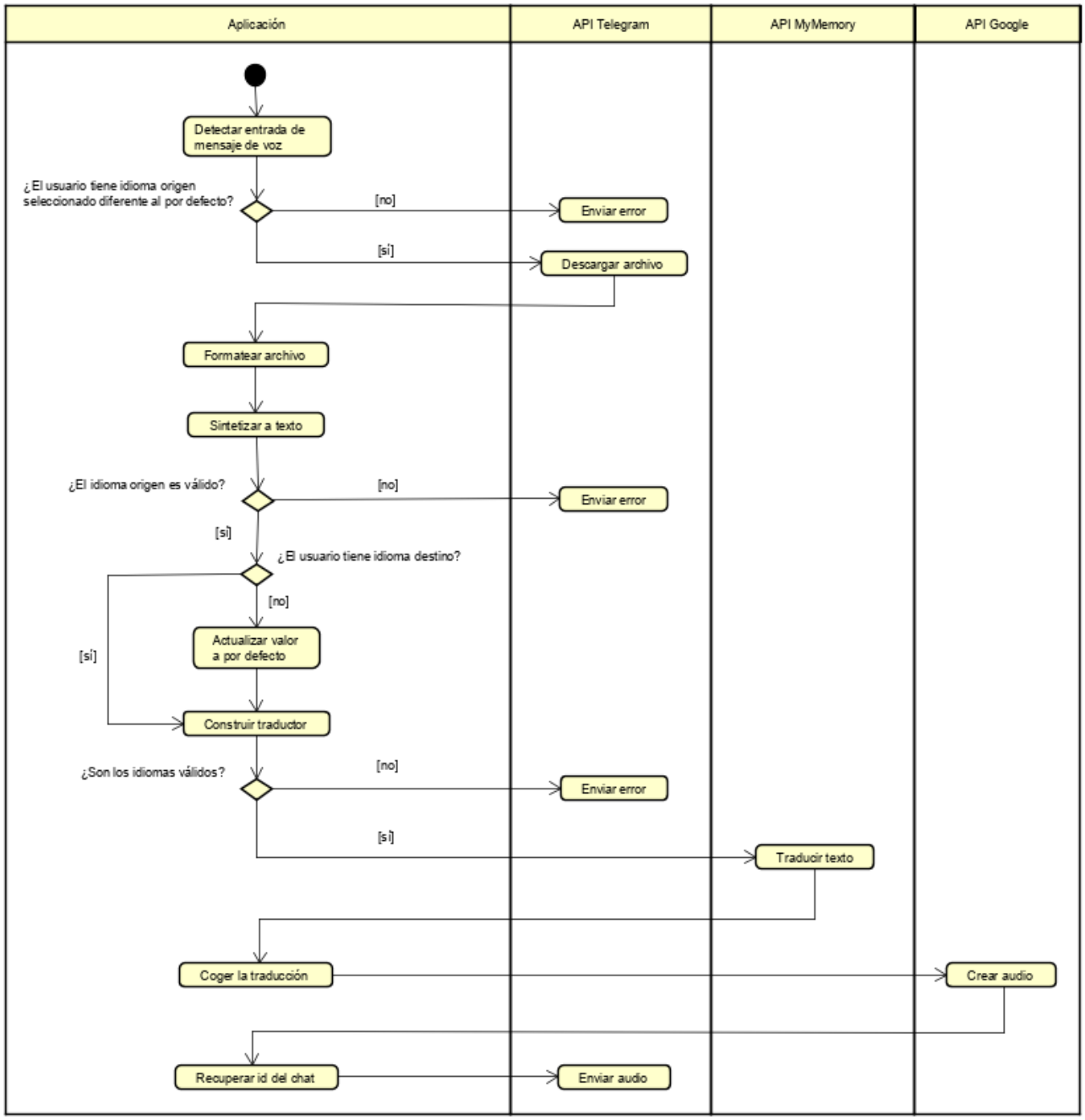

Figura 7.8: Diagrama de actividad: Traducir mensaje de voz

Capítulo 8

Diseño

8.1 Patrón de diseño

El patrón de diseño elegido para el desarrollo de la aplicación es el ***chain of responsibility pattern***[24] con un pequeño añadido.

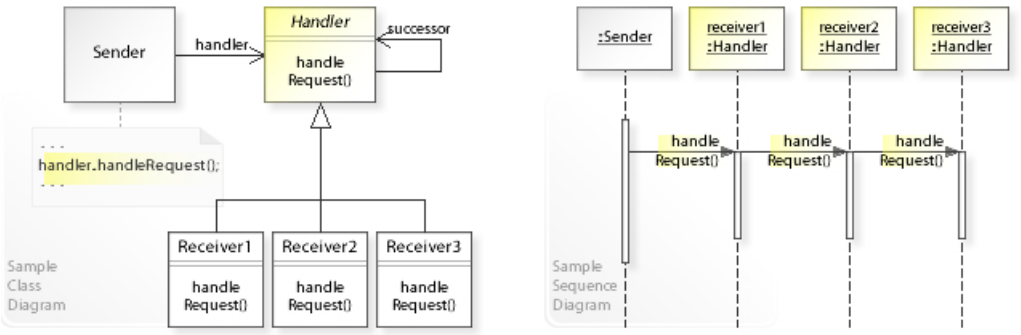

Figura 8.1: Cadena de responsabilidad simple

Tal y como se puede apreciar en la figura anterior, este patrón originalmente se basa en un conjunto de *handlers* que reciben la orden del cliente. Esta orden es recibida por el primero de los *handlers*, en caso de que no aplique a él o requiera de una acción a mayores, se envía la orden al siguiente *handlers*. Esto continúa hasta que se da con un *handler* que cumple la orden y da por finalizado el proceso.

El añadido implementado en nuestra solución es el uso de un ***Dispatcher***, cuya misión es ahorrar este esfuerzo y llamadas innecesarias que se realizan en el patrón original. Mediante la detección de comandos o formatos de entrada, somos capaces de dividir las diferentes funciones en *handlers* específicos para cada una de ellas. Esto nos aporta una simplicidad de procesamiento y lectura mayor. Destacamos la importancia de la librería PYTHON-TELEGRAM-BOT [1] una vez más para la implementación de este patrón, ya que aporta todas estas clases necesarias para llevarlo a cabo de manera altamente eficiente.

8.2 Diagrama de clases

En este diagrama de clases especificamos en detalle las entidades de la solución. Cabe mencionar que en esta implementación las entidades principales con las que trabajamos son parte de una librería externa muy completa, y por ello describiré únicamente aquellos atributos que empleemos para evitar explayarse en exceso. Para información relacionada mucho más exhaustiva mirar la documentación de la librería [1].

Las entidades con las que trabajamos explicadas especialmente para su uso en este proyecto son:

- ***Updater*:** Se encarga de la comunicación con la API de TELEGRAM manteniendo la funcionalidad y el código conectados y haciendo que el BOT se mantenga escuchando los chats.
- ***Dispatcher*:** Se encarga de la detección de acciones por parte del usuario así como de la distribución al comando adecuado.
- ***CommandHandler*:** Instancias específicas para cada comando que emplea el *Dispatcher*.
- ***MessageHandler*:** Instancias específicas para llamadas a funciones por mensajes que no contienen comandos.
- ***Filters*:** Filtros que asisten al *MessageHandler* para la detección de mensajes que tienen que activar funciones.
- ***Update*:** Contiene un cambio o actualización en el estado del BOT.
- ***Message*:** Contiene el mensaje enviado por el usuario. así como información relativa al punto desde el que se ha enviado.
- ***Context*:** Contiene información útil para llevar a cabo el comando.
- ***bot*:** Instancia del BOT usada para ejecutar una acción.
- ***Translator*:** Entidad encargada de traducir texto a texto.
- ***AudioSegment*:** Como su nombre indica, contiene segmentos de audio. También es el encargado del cambio de formato de audio.
- ***AudioFile*:** Contiene los segmentos de audio y es lo que reconoce Recognizer.
- ***Recognizer*:** Entidad encargada de la tanscripción a texto del mensaje de voz.
- ***gTTS*:** Entidad encargada de la síntesis de texto a audio.

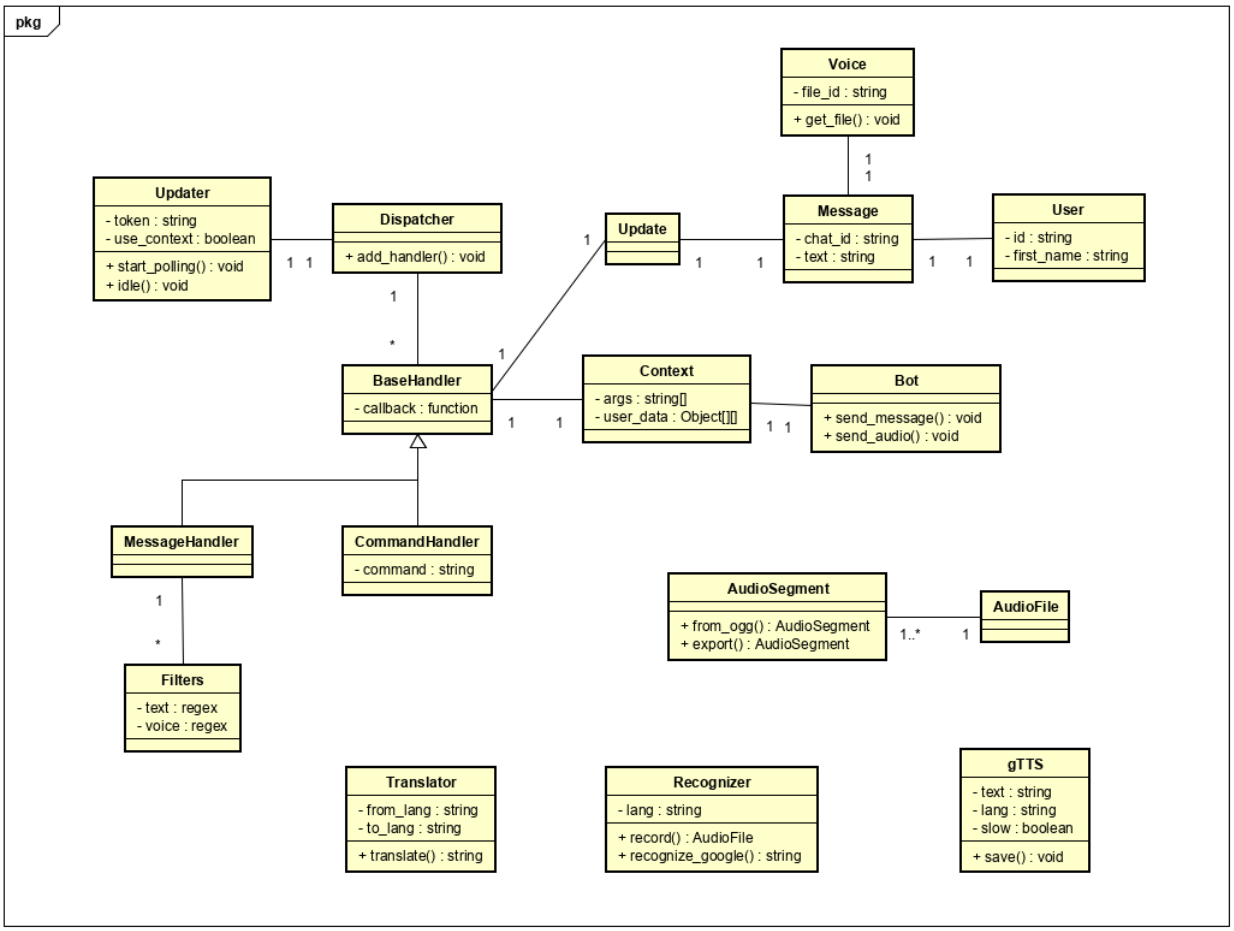

Figura 8.2: Diagrama de Clases

8.3 Arquitectura lógica

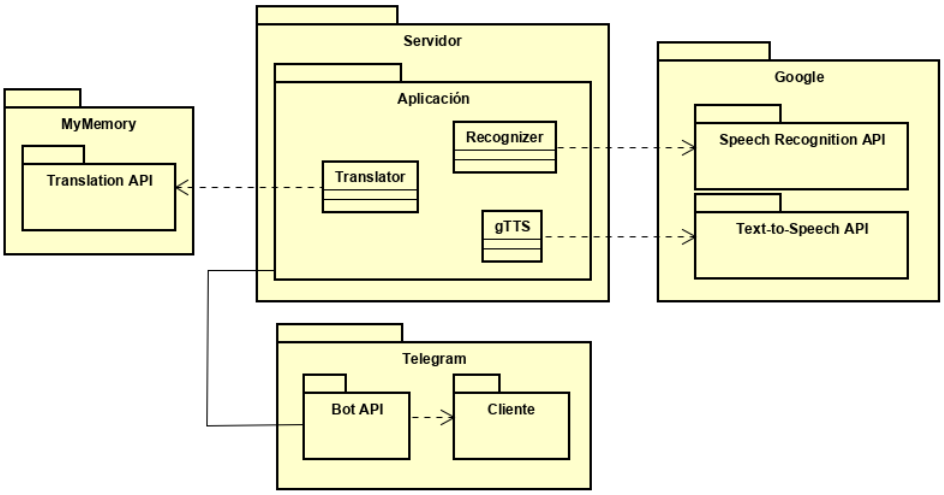

Figura 8.3: Diagrama de arquitectura lógica

En este diagrama queda reflejado el despliegue del proyecto, incluyendo las pertenencias, así como las comunicaciones y dependencias que existen interna y externamente.

Como podemos observar en la figura, el cliente es el propio servicio TELEGRAM, el cual es observado por la API para BOTS de TELEGRAM y esta es la que se comunica con nuestra aplicación gracias al *Updater* que describimos en el diagrama de clases 8.2. Por otro lado, el servidor es lo que mantendrá viva nuestra aplicación, escuchando 24/7 al cliente.

A parte de la dependencia con la API para BOTS de TELEGRAM, encontramos también las llamadas del traductor a MYMEMORY y del reconocedor y convertidor a GOOGLE, cada uno a su API correspondiente.

8.4 Arquitectura física

Del lado del **cliente**, el sistema electrónico principalmente puede ser tanto un teléfono móvil como un ordenador personal, pudiendo este último variar entre el sistema web o la aplicación de escritorio. En general, cualquier dispositivo que pueda acceder a cualquiera de las variantes que ofrece TELEGRAM es válido.

En cuanto a la aplicación, está dentro de varias capas que forman lo que en el apartado anterior denominamos **servidor**. Las capas en orden creciente son:

- **Máquina virtual:** Sistema operativo basado en UBUNTU que contiene de forma local el archivo PYTHON de la aplicación, así como todo lo necesario para la creación y mantenimiento de la siguiente capa. Esta máquina virtual fue creada y aportada al completo por la Universidad de Ingeniería Informática de Valladolid, y su mantenimiento corre completamente a su cargo.

- **Contenedor** DOCKER**:** Construido y levantado desde la capa anterior, tiene como único propósito levantar la aplicación y por ello contiene todo lo necesario para llevarlo a cabo, desde el sistema de carpetas hasta el lenguaje de programación y las librerías instaladas de la manera correcta. Esto es una forma sencilla de aportar seguridad y facilidad de mantenimiento sin perder velocidad de despliegue.

- FASTAPI**:** Como último paso para levantar la aplicación, contenemos el lanzamiento del BOT dentro de un *endpoint* http el cual, a través de un puerto que dejamos preparado en el contenedor DOCKER, da la señal de salida a la aplicación al acceder a él.

Capítulo 9

Implementación y Pruebas

9.1 Herramientas y tecnologías

9.1.1 Lenguaje de programación

Figura 9.1: Logo de Python

Python [29] es un lenguaje de alto nivel de programación interpretado desarrollado por Python Software Foundation cuya filosofía hace hincapié en la legibilidad de su código y el cual se puede utilizar para desarrollar aplicaciones de todo tipo.

Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Está clasificado como uno de los lenguajes de programación más populares y también uno de los más solicitados en el entorno laboral debido al crecimiento en plazas relacionadas con el análisis y tratamiento de datos entre otras cosas.

Se ha elegido para este proyecto debido principalmente al gran abanico de librerías y sinergia con APIs con la que cuenta, factor crucial en este proyecto. Como motivos secundarios, es extremadamente cómodo y legible para proyectos de tamaño moderado, además del interés inicial de trabajar con él ya que es un lenguaje que he visto poco en la carrera para el desarrollo de aplicaciones ajenas a estudio de datos.

9.1.2 Entornos de desarrollo

Figura 9.2: Logo de PyCharm

PyCharm[28] es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains específicamente para el desarrollo en Python. Algunas de sus **características principales** son:

- **Asistencia y análisis de la codificación**, con completado de código, resaltado de sintaxis y errores, integración de *linters* y correcciones rápidas.
- **Navegación por el proyecto y el código**: vistas especializadas del proyecto, vistas de la estructura de archivos y salto rápido entre archivos, clases, métodos y usos.
- **Refactorización de** Python: incluye renombrar, extraer método, introducir variable, introducir constante, tirar hacia arriba, empujar hacia abajo y otros.
- **Depurador de** Python **integrado**.
- **Pruebas unitarias integradas**, con cobertura de código línea por línea.
- **Integración del control de versiones**: interfaz de usuario unificada para Mercurial, Git, Subversion, Perforce y CVS con listas de cambios y fusión.

PyCharm cuenta con tres versiones: La ***Community Edition*** se publica bajo la licencia apache, la *Professional Edition*, que contiene características adicionales, publicada bajo una licencia propietaria financiada por suscripción, por último también existe una versión educativa. En este proyecto nos decantamos por la edición comunitaria ya que no se iba a dar uso a las características adicionales que proporciona la edición profesional.

Este IDE fue el usado hasta la tercera versión de la aplicación para su desarrollo local previo al uso de la máquina virtual, principalmente debido a que el desarrollador estaba familiarizado con la herramienta y simplifica en gran medida la puesta a punto inicial de un proyecto, así como la instalación de librerías y su documentación.

Figura 9.3: Logo de Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

Tal y como hemos mencionado, empleamos PyCharm para el desarrollo hasta la tercera versión de la aplicación, por lo que todo el resto del desarrollo cayó en manos de Visual Studio Code. Este cambio fue debido a que este IDE nos permite conectarnos remotamente a la máquina virtual donde reside la aplicación y editar directamente sus archivos, así como el correspondiente control de versiones y la creación de consolas para cualquier motivo mayor.

9.1.3 Control de versiones

Figura 9.4: Logo de GitHub

GitHub[25] es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones de Git. El código de los proyectos alojados en GitHub, al igual que este, se almacena generalmente de forma pública.

El 4 de junio de 2018, Microsoft compró GitHub por la cantidad de 7500 millones de dólares. Al inicio, el cambio de propietario generó preocupaciones y la salida de algunos proyectos de este sitio; sin embargo, no fueron representativos. GitHub continúa siendo la plataforma más importante de colaboración para proyectos de código abierto.

Figura 9.5: Logo de Sourcetree

Sourcetree es una interfaz gráfica de usuario (GUI) colaborativa orientada a simplificar el manejo del control de versiones, haciendo la visualización del código alterado su misión principal. Se puede conectar con el repositorio GitHub y ser usada como única herramienta para el control de versiones ya que permite el manejo de ramas, así como su actualización y mantenimiento.

Dentro del proyecto, esta herramienta fue empleada de forma conjunta a PyCharm y por tanto su uso comprende el mismo intervalo de tiempo. Una vez se empezó a desarrollar en Visual Studio Code para el desarrollo interno de la máquina virtual, se pasó a usar el propio IDE para el control de versiones entre otros motivos porque Sourcetree no tiene soporte para Linux.

9.1.4 Diagramas de análisis y diseño

Figura 9.6: Logo de ASTAH PROFESSIONAL

ASTAH[31], conocido formalmente como JUDE, es una herramienta de modelado *UML* creada por la compañía japonesa CHANGE VISION. Se ha empleado para desarrollar todos los diagramas que se encuentran en la memoria.

9.1.5 Desarrollo de Memoria

Figura 9.7: Logo de Overleaf

Overleaf[27] es un editor LaTeX colaborativo en línea gratuito creado por John Hammersley y John Lees-Miller. Se ha usado como única herramienta para la escritura al completo de esta memoria.

9.2 Detalles de Implementación

Gran parte de la implementación ya ha sido mencionada en el capítulo 4: Marco Conceptual, pero en esta sección explicaremos algo más en detalle la implementación desde el punto de vista práctico, mostrando fragmentos de código u otras imágenes orientativas. Dado que se han empleado librerías externas para toda la funcionalidad importante, se han ahorrado una gran cantidad de código propio tanto en tamaño como en complejidad, por lo que el seguimiento de este apartado debería ser llevadero para cualquier lector.

Para cualquier duda que pueda surgir durante la lectura de esta sección, existe una descripción de las librerías y las clases empleadas en 9.2.2 y 8.2 respectivamente.

9.2.1 Creación del BOT

Lo primero es lo primero, y no se puede implementar funcionalidad a un BOT sin antes tener un BOT, por lo que el paso inicial será su creación. Resumiendo, existe un BOT oficial de TELEGRAM llamado BOTFATHER, con el cual iniciaremos una conversación como si de cualquier otro BOT se tratase y nos proporcionará las opciones necesarias para la creación y customización completa de nuestro nuevo BOT. Una vez creado nos enviará la token de nuestro BOT, y esto es muy importante ya que será la llave a la conexión entre el código implementado y la API de TELEGRAM BOT, es decir, quien tenga este token puede modificar o incluso eliminar el BOT al completo.

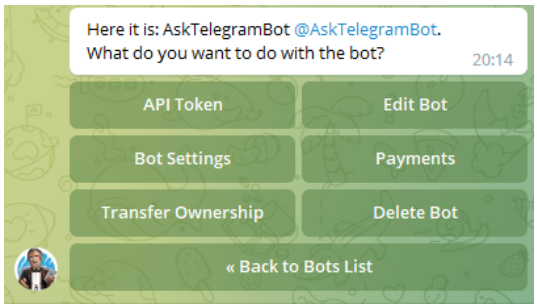

Figura 9.8: Menú principal de customización de un BOT con BOTFATHER

Tras editar las opciones que aparecen en *Edit bot*, el perfil quedó tal que así:

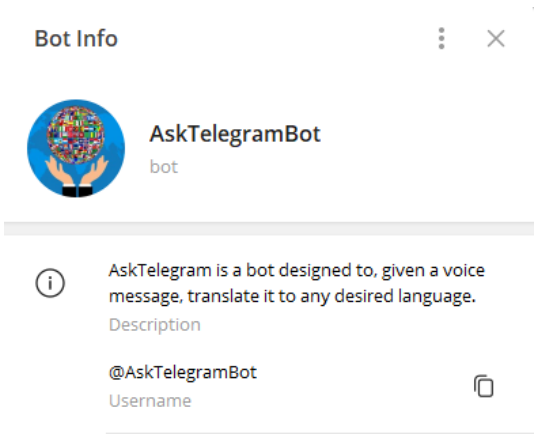

Figura 9.9: Información de @ASKTELEGRAMBOT

Destacamos la importancia de que al crear un BOT, por defecto se iniciará con la seguridad de chats grupales activada. Esto evita que el BOT pueda leer y por tanto tomar medidas con los mensajes de un grupo que no vayan explícitamente dirigidos a él, ya sea mediante un comando personal o llamándolo directamente con su nombre de usuario previo al resto del mensaje. En nuestro caso, hubo que desactivar esta opción de seguridad debido a la implementación del modo rápido de traducción.

9.2.2 Librerías empleadas

```
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
from translate import Translator
import logging as log
import speech_recognition as sr
import os
from gtts import gTTS
from pydub import AudioSegment
import uvicorn
from fastapi import FastAPI
```

Figura 9.10: Importaciones de las librerías

- TELEGRAM.EXT: Paquete principal de la librería PYTHON-TELEGRAM-BOT, que nos aporta las herramientas para el manejo de la API para BOTS de TELEGRAM.
- TRANSLATE: Para la creación del traductor que lleva a cabo todas las traducciones de las diferentes funcionalidades del BOT. Incluye soporte a diferentes APIs de traducción, aunque en este proyecto solo usaremos MYMEMORY API.
- LOGGING: Herramienta de *logs* internos básica.
- SPEECH_RECOGNITION: Sintetizador de voz a texto. Incluye soporte a diferentes APIs, aunque en este proyecto solo usaremos la de GOOGLE SPEECH RECOGNITION.
- OS: Ejecuta acciones del sistema indiferentemente del sistema operativo en el que esté contenido.
- GTTS: Creación de audios a través de voz mediante la API de GOOGLE TEXT-TO-SPEECH.
- PYDUB: Manejo y procesamiento de archivos de audio.
- UVICORN: Constructor de la conexión con nuestra API.
- FASTAPI: Comprende las diferentes herramientas para la creación y manejo de nuestra API.

9.2.3 Lanzamiento de API

```
app = FastAPI()


@app.get("/")
def index():
    log.basicConfig(level=log.INFO)
    launch_bot()


if __name__ == '__main__':
    uvicorn.run("v4:app", port=8000, reload=True)
```

Figura 9.11: Código para el lanzamiento de la API

Este fragmento de código compone la fase inicial de la aplicación previa al lanzamiento del BOT. Se puede observar cómo se ejecuta el constructor que inicializa la instancia API llamada *app* incluyendo el *script* PYTHON v4, es decir la última versión de nuestra aplicación. Esto permite que al acceder en el navegador a la

dirección determinada (revisar el manual de instalación A), el *endpoint* que se ve en la mitad de la figura se active, inicializando la herramienta de *logs* y llamando a la función que lanzará la implementación del BOT.

9.2.4 Lanzamiento del BOT

```python
def launch_bot():
    token = os.getenv('TOKEN')
    updater = Updater(token, use_context=True)
    dispatcher = updater.dispatcher

    dispatcher.add_handler(CommandHandler('start', start))
    dispatcher.add_handler(CommandHandler('help', _help))
    dispatcher.add_handler(CommandHandler('echo', echo))
    dispatcher.add_handler(CommandHandler('translate', translate))
    dispatcher.add_handler(CommandHandler('fromlanguage', fromlanguage))
    dispatcher.add_handler(CommandHandler('tolanguage', tolanguage))
    dispatcher.add_handler(CommandHandler('fastmode', fastmode))
    dispatcher.add_handler(CommandHandler('toaudio', toaudio))
    dispatcher.add_handler(MessageHandler(Filters.voice, speech_translate))
    dispatcher.add_handler(MessageHandler(Filters.text, fast_translate))
    # Starts the bot
    updater.start_polling()
    log.info('Bot launched')
    # Bot nonstop hearing the channel
    updater.idle()
```

Figura 9.12: Función *launch_bot*

Esta función contiene la inicialización de la conexión con la API para BOTS de TELEGRAM, creando el ***Updater*** gracias a la token de autoría del BOT para posteriormente, gracias a las funciones *start_polling* y *idle*, iniciar nuestro BOT y dejarle escuchando los diferentes canales que existan con él.

Una vez iniciado, evaluará los mensajes que reciba gracias al ***Dispatcher*** y los *handlers* que se definen en la sección del medio de la imagen. Los *CommandHandlers* detectarán el uso de los comandos que se describen en el primer parámetro, ejecutando la función que se envía como segundo parámetro en caso de que se cumpla. Por otro lado, los *MessageHandlers* recibirán todos los mensajes por parte del usuario, aplicándoles el filtro de si son mensajes de texto o de voz sin que intervenga ningún comando.

9.2.5 *Logs*, recepción de parámetros y envío de mensajes

```python
def echo(update, context):
    log.info(f' User "{update.message.from_user.id}" used command /echo sending "{" ".join(context.args)}"')
    context.bot.send_message(update.message.chat_id, ' '.join(context.args))
```

Figura 9.13: Función echo

Aprovechando la simplicidad de la función echo, la usaremos como ejemplo para ilustrar el funcionamiento de estos tres factores:

- ***Logs*:** El objetivo principal de los *logs* en esta aplicación es la de informar internamente del flujo de llamadas a funciones que existe por parte de los usuarios, incluyendo la información variable que se dé dentro de la misma. Esto nos permite detectar, en caso de que el BOT fallase en cierto momento, ver cuál ha sido la última interacción realizada y por tanto la causante de la excepción, junto con los datos necesarios para detectar el error y replicarlo para su respectivo test.

En este caso, se informa de que el usuario ha llamado a esta función usando el comando `/echo` junto con la frase que haya enviado junto con el comando.

- **Recepción de parámetros:** Dentro de *Context*, que recibimos como parámetro, encontraremos los argumentos pasados por el usuario tras el comando. En este caso, debido a que recibimos una frase de duración indefinida, los argumentos serán un *array* de *strings* donde cada uno contiene una palabra ya que se ha dividido la frase por espacios. Por lo que, uniendo todos sus elementos junto con un espacio, conseguiremos recuperar la frase al completo.

- **Envío de mensajes:** Para enviar un mensaje de texto usaremos la función *send_message* del objeto BOT que está también contenido en *Context*, donde, además del propio mensaje, requeriremos de la id del chat al que deseemos enviárselo. A menos que desees realizar prácticas poco ortodoxas o informar al usuario por chat privado de algo, este chat destino siempre coincidirá con el origen desde el que alguien ha llamado a la función, por lo que simplemente usaremos la variable *chat_id* que está contenida en el objeto *Message*, el cual recibimos por medio del parámetro *Update*.

9.2.6 Persistencia

```python
def fastmode(update, context):
    log.info(f' User "{update.message.from_user.id}" used command /fastmode')
    if context.user_data.get('fastmode') is None:
        context.user_data['fastmode'] = False
    fast_mode = context.user_data.get('fastmode')
    context.user_data['fastmode'] = not fast_mode
    context.bot.send_message(update.message.chat_id, 'Fast mode deactivated.' if fast_mode else 'Fast mode activated.')
```

Figura 9.14: Función *fast_mode*

Como herramienta para almacenar datos necesarios de cada usuario, empleamos el diccionario *user_data* que está contenido en *Context*, en él almacenamos el idioma origen y destino de las traducciones y el booleano que indica si el modo rápido se encuentra activado o desactivado.

Esta última variable que hemos mencionado es la empleada en la función de la imagen superior, donde se puede apreciar como podemos acceder al valor mediante la función *get* y actualizar el valor con una simple igualdad. Para inicializar un valor por primera vez en el diccionario, se haría de forma idéntica a una actualización. Como puesta en contexto adicional en caso de querer seguir el código, recordamos que esta función activa o desactiva el modo rápido dependiendo de su valor actual.

Este diccionario no persiste más allá de la instancia de nuestra aplicación, lo que quiere decir que en caso de cerrarse ya sea por un fallo en el sistema o por elección propia, este diccionario perdería todos los datos que contenga. Dada la naturaleza de la funcionalidad que deseamos dar, este problema no tiene una importancia mayor, debido a que son datos que están guardados principalmente para diferenciar usuarios que lo usen simultáneamente y normalmente se encontrarán en constante cambio por parte del usuario. Además, en caso de que el usuario desee volver al estado en el que estaba previo a la caída del servicio, no le costará más de medio minuto de su tiempo.

9.2.7 Función principal

Por último describiremos el funcionamiento de la traducción de mensajes de voz, ya que es el objetivo principal de este BOT y por tanto el código más importante, que a su vez contiene funcionalidad implementada en *translate* y *toaudio*.

```
def speech_translate(update, context):
    file_name = f'voice_messages/{update.message.voice.file_id}'
    fromlanguage = context.user_data.get('from')
    log.info(f' User "{update.message.from_user.id}" translating speech on file "{file_name}" from language "{fromlanguage}"')

    if not fromlanguage or fromlanguage == 'autodetect':
        context.bot.send_message(update.message.chat_id, 'Select a specific language using /fromlanguage')
        return
```

Figura 9.15: Función *speech_translate*: Inicio

Comenzamos comprobando si existe un idioma origen, ya que será el usado para el reconocimiento del habla del mensaje de voz recibido. En caso de no haber sido seleccionado uno, o que se encuentre en modo de detección del idioma, el cual no es válido en el reconocimiento de voz, se enviará un mensaje de error y finalizará la función.

```
ogg = file_name + '.ogg'
wav = file_name + '.wav'
update.message.voice.get_file().download(ogg)
sound = AudioSegment.from_ogg(ogg)
sound.export(wav, format='wav')
r = sr.Recognizer()
with sr.AudioFile(wav) as source:
    audio = r.record(source)
os.remove(ogg)
os.remove(wav)

recognized_text = r.recognize_google(audio, language=context.user_data.get('from'))
context.bot.send_message(update.message.chat_id, 'You said: ' + recognized_text)
```

Figura 9.16: Función *speech_translate*: Audio a texto

Tras esto, descargaremos el mensaje de voz recibido por parte del usuario contenido en *Message*. Este archivo viene en formato *.ogg*, pero lo necesitamos en *.wav*, por lo que mediante la función *export* de *AudioSegment* lo convertiremos a dicha extensión. Una vez hecho esto, introducimos el segmento de audio en un *AudioFile* y lo grabamos mediante la función *record* del *Recognizer*. Por último, borramos los archivos para que no se nos amontonen archivos innecesarios en el sistema y procedemos al reconocimiento del archivo mediante la API de GOOGLE con la función *recognize_google*, pasándole a mayores el idioma origen del usuario.

```
translator = get_translator(context)
translated_text = translator.translate(recognized_text)

gtts = gTTS(text=translated_text, lang=context.user_data.get("to"), slow=False)
gtts.save('voice_messages/translation.mp3')
context.bot.send_audio(chat_id=update.message.chat_id, audio=open('voice_messages/translation.mp3', 'rb'))
os.remove('voice_messages/translation.mp3')
```

Figura 9.17: Función *speech_translate*: Traducción y texto a audio

Tenemos el texto a traducir, por lo que llamamos a *get_translator*, función creada personalmente que construye un traductor con los idiomas origen y destino del usuario o valores por defecto (autodetectar como origen y inglés como destino) en caso de no tener ninguno seleccionado. Con el traductor creado, traducimos gracias a la API de MYMEMORY, que al ser la opción por defecto de la librería *translate*, evita tener que pasar un parámetro adicional a la función. Como paso final, construimos el creador de audios junto con el texto y el idioma destino del usuario, guardamos el audio como un mp3 y se lo enviamos al usuario llamando a *send_audio* en vez del *send_message* que veníamos usando.

9.3 Pruebas de aceptación

En este capítulo se describirán las pruebas realizadas en la aplicación para comprobar su correcto funcionamiento, probando cada posible caso de uso. Estas pruebas se han realizado al final del desarrollo con el objetivo de dar el visto bueno final al proyecto.

Identificador	PA-01.
Nombre	Caso de uso: *Start*.
Entrada	Mensaje de texto `/start`.
Salida esperada	Mensaje de texto conteniendo una introducción al BOT.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.1: PA-01. Caso de uso: *Start*.

Identificador	PA-02.
Nombre	Caso de uso: *Help*.
Entrada	Mensaje de texto `/help`.
Salida esperada	Mensaje de texto conteniendo ayuda y una lista de comandos.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.2: PA-02. Caso de uso: *Help*.

Identificador	PA-03.
Nombre	Caso de uso: *Echo*.
Entrada	Mensaje de texto `/echo hola mundo`.
Salida esperada	Mensaje de texto *hola mundo*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.3: PA-03. Caso de uso: *Echo*.

Identificador	PA-04.
Nombre	Caso de uso: *FromLanguage*.
Entrada	Dos mensajes de texto, el primero conteniendo `/fromlanguage es` y el segundo `/fromlanguage`.
Salida esperada	Primera respuesta: *Language updated*. Segunda respuesta: *es*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.4: PA-04. Caso de uso: *FromLanguage*.

Identificador	PA-05.
Nombre	Caso de uso: *ToLanguage*.
Entrada	Dos mensajes de texto, el primero conteniendo `/tolanguage es` y el segundo `/tolanguage`.
Salida esperada	Primera respuesta: *Language updated*. Segunda respuesta: *es*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.5: PA-05. Caso de uso: *ToLanguage*.

Identificador	PA-06.
Nombre	Caso de uso: *FastMode*.
Entrada	Dos mensajes de texto conteniendo `/fastmode` ambos.
Salida esperada	Primera respuesta: *fast mode activated*. Segunda respuesta: *fast mode deactivated*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.6: PA-06. Caso de uso: *FastMode*.

Identificador	PA-07.
Nombre	Caso de uso: *Translate*.
Entrada	Teniendo el usuario guardados como idioma origen español y como idioma destino inglés, envía el mensaje de texto `/translate hola mundo`.
Salida esperada	Mensaje de texto *hello world*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.7: PA-07. Caso de uso: *Translate*.

Identificador	PA-08.
Nombre	Caso de uso: *Translate* fallo en autodetección.
Entrada	Teniendo el usuario guardados como idioma origen *autodetect* y como idioma destino inglés, envía el mensaje de texto `/translate hola mundo`.
Salida esperada	Mensaje de texto *hello world*.
Salida obtenida	Error recibido de la API de MyMemory diciendo que no se ha elegido un par de idiomas correcto.

Cuadro 9.8: PA-08. Caso de uso: *Translate* fallo en autodetección.

Este caso se probó cuando se desarrolló este comando en el segundo sprint del proyecto y funcionaba correctamente. Tras una revisión de la la documentación de la API de MyMemory comprobamos que ha sido actualizada un mes atrás, por lo que informamos del problema en su foro de discusiones.

Finalmente el equipo de mantenimiento de la API se encargó de solucionar este error al poco tiempo de publicarlo en el foro, por lo que no hubo que modificar nada de nuestro proyecto.

Identificador	PA-09.
Nombre	Caso de uso: *ToAudio*.
Entrada	Teniendo el usuario guardados como idioma origen español y como idioma destino inglés, envía el mensaje de texto `/toaudio hola mundo`.
Salida esperada	Mensaje de voz con acento en inglés diciendo *hello world*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.9: PA-09. Caso de uso: *ToAudio*.

Identificador	PA-10.
Nombre	Caso de uso: Traducir mensaje de voz.
Entrada	Teniendo el usuario guardados como idioma origen español y como idioma destino inglés, envía el mensaje de voz diciendo alto y claro ”hola mundo”.
Salida esperada	Mensaje de voz con acento en inglés diciendo *hello world*.
Salida obtenida	Obtenemos la salida esperada.

Cuadro 9.10: PA-10. Caso de uso: Traducir mensaje de voz.

Capítulo 10

Conclusiones

En este proyecto hemos podido comprobar el gran abanico de posibilidades que proporcionan los BOTS conversacionales hoy en día, así como la preparación de diversas plataformas para el desarrollo de los mismos, dando una aproximación sencilla en TELEGRAM por medio de PYTHON. También hemos observado la potencia actual para la traducción, el reconocimiento del habla y la síntesis de voz que se puede lograr con la tecnología y algoritmia actuales, proporcionando un grado de calidad que no se tomaba por fijo al inicio del proyecto.

Tras su finalización, podemos afirmar que se han cumplido todos los objetivos planteados inicialmente, así como características y mejoras que se han ido añadiendo durante la realización del proyecto para pulir el producto final y aportar la mayor profundidad posible dentro del alcance definido. Teniendo esto en cuenta, concluimos que el proyecto ha sido un éxito.

Inicialmente, el desarrollo de un *chatbot* puede parecer una tarea muy ardua con un gran trabajo de desarrollo por detrás, pero gracias a las facilidades aportadas por TELEGRAM en su API para BOTS, así como la documentación y guías que esta contiene, finalmente ha sido una de las menores preocupaciones en la implementación de la aplicación.

Por otro lado, gracias al trabajo de investigación de herramientas y librerías inicial y todos los aportes de la comunidad PYTHON, un proyecto que conllevaría de otra forma de una extensa implementación de código con un grado considerable de dificultad como es implementar *TAC*, *text-to-speech* y *speech-to-text*, finalmente ha sido realizado en una solución de tamaño relativamente pequeño, consiguiendo además aproximaciones de una calidad sobresaliente debido al uso de *wrappers* y APIs externas de la mano de GOOGLE y MYMEMORY.

Por último, quiero destacar el seguimiento del tutor, el cual ha sido excelente en todo momento, ejerciendo un buen papel de cliente exigiendo mejoras pero al mismo tiempo el de orientador incluso realizando ayudas de apoyo en algún determinado momento como la puesta a punto de la máquina virtual. Por lo que solo me queda agradecer su labor y dedicación compaginando esto con su trabajo y asuntos personales.

10.1 Trabajo futuro

Como funcionalidades adicionales que se podrían implementar en un futuro tenemos:

- **Elección de APIs externas:** Una de las opciones que se barajaron fue aprovechar la polivalencia de las librerías empleadas para la traducción y el reconocimiento de voz, ya que tienen soporte para varias APIs diferentes, siendo algunas de ellas más potentes que las opciones actuales con la desventaja de ser de pago. La idea sería aportar un comando o un parámetro adicional para que el usuario pueda establecer otra API como su determinada para dichas funcionalidades, informando de su token de pago en caso de requerirse en dicha API.

- **Uso avanzado de** FastAPI [4] Actualmente está implementado en el código el lanzamiento del BOT por medio de esta herramienta, pero podríamos hacer que el BOT al completo pudiese ser controlado vía web, desde sus comandos hasta opciones de propietario como alterar la ejecución o finalizarla. Esta herramienta proporciona una interfaz gráfica a partir de los *endpoints* que se encuentren instanciados, por lo que esta opción es completamente viable.

- **Creación de librerías propias:** Gran parte de la funcionalidad de esta aplicación esta delegada a servicios externos para los que sería interesante dar nuestra propia solución, obviamente su desarrollo llevaría a un tiempo bastante elevado debido a la naturaleza de los problemas, pudiendo incluso contar el desarrollo de alguna de ellas como un trabajo de fin de grado por sí mismo.

- **Logs avanzados:** Se podría realizar un *logging* más detallado para un mayor seguimiento de las acciones del usuario o su repercusión en el sistema, ya que lo que existe ahora mismo es una herramienta principalmente implementada para la detección y prueba de errores.

- **Mayor persistencia:** Tal y como comentamos al final de la sección 9.2.6, en caso de una caída o reinicio de la aplicación, se perderían los datos de los usuarios. Aunque esto no suponga un problema sustancial debido a la poca repercusión que esto conlleva, la implementación de una base de datos u opción similar que evite esto podría mejorar la experiencia y profesionalidad de la aplicación.

- **Comando `/settings`:** Comando que al ser introducido informe al usuario del estado actual de todos sus ajustes, es decir, idiomas y modo rápido.

Apéndices

Apéndice A

Manual de Instalación

En este apartado se definen las herramientas y pasos a realizar para instalar y utilizar la aplicación en una máquina Linux.

En caso estándar, el BOT estará ya levantado y funcionando en el servidor, por lo que este manual se aporta en caso de que esto no sea así o de que se quiera tomar como ejemplo para un propósito personal.

Nos saltaremos la creación del BOT en TELEGRAM[17] ya que la instancia ya está creada y solamente hace falta levantar el contexto de la aplicación para que haga referencia al mismo mediante su token personal.

A.1 Requisitos de instalación

Dado que el contexto se levanta mediante Docker[2], el único requerimiento inicial será tener instalado en la máquina PIP para la posterior descarga de DOCKER. En este caso la versión empleada es la *20.10.12*.

Todas las demás dependencias serán resueltas siguiendo los pasos de creación del Docker en la siguiente sección.

A.2 Inicio de la aplicación

Seguimos los siguientes pasos:

1. Clonamos en una carpeta todo el contenido del proyecto que está subido a GITHUB usando el comando:

```
$ git clone https://github.com/mgosanto/AskTelegramBot.git
```

2. Creamos el *Dockerfile* mediante el comando:

```
$ touch Dockerfile
```

Dentro escribimos lo que se puede ver en la imagen A.1, modificando las rutas del primer parámetro de los dos comandos `COPY` por las equivalentes a nuestra carpeta creada en el primer paso.

```
FROM python:3.8

RUN mkdir app
RUN mkdir app/voice_messages

COPY AskTelegramBot/v4.py /app
COPY AskTelegramBot/requirements.txt /app

WORKDIR /app

RUN pip install --no-cache-dir --upgrade pip
RUN pip install -r /app/requirements.txt
RUN apt-get update
RUN echo Y | apt install ffmpeg

EXPOSE 8000

CMD ["uvicorn", "v4:app", "--host=0.0.0.0", "--reload"]
```

Figura A.1: *Dockerfile*

Todas las dependencias con librerías PYTHON se verán resueltas gracias al *requirements.txt* y la instalación independiente de FFMPEG.

```
httpx ~= 0.23.0
tornado~=6.1
cachetools~=4.2.2
APScheduler~=3.6.3
pytz>=2018.6

# Telegram BOT API wrapper
python-telegram-bot~=13.14

# translation
translate~=3.6.1

# text-to-speech
gTTS ~= 2.2.4

# API creation
fastapi ~= 0.85.0

# audio formatter
pydub ~= 0.25.1

# speech-to-text
SpeechRecognition ~= 3.8.1

# fastapi dependency
uvicorn ~= 0.18.3

# instalation dependency
wheel ~= 0.37.1
```

Figura A.2: Contenido de *requirements.txt*

3. Construimos la imagen docker mediante el comando:

```
$ docker build -t botcontainer .
```

4. Ejecutamos el docker con el comando:

```
$ docker run -p 8000:8000 --name my-api botcontainer
```

   Nota: En caso de querer cambiar el puerto, se deberá modificar tanto en el *Dockerfile* visto en el paso 2, como en el código de la aplicación.

5. Por último, accedemos a la url *localhost* o *127.0.0.1* junto con el puerto pertinente para desplegar la API, en nuestro caso al completo sería *127.0.0.1:8000*.

   Con esto tendríamos al BOT levantado y listo para dar respuestas en TELEGRAM.

Apéndice B

Manual de Usuario

Para utilizar la aplicación necesitaremos primero tener iniciada ya sea la aplicación de escritorio o la versión web de TELEGRAM. Tras esto, buscaremos en la barra de búsqueda ASKTELEGRAMBOT e iniciaremos una conversación con él.

De primeras, nos aparecerá el mensaje de bienvenida y un botón grande abajo para comenzar. A partir de este momento valdrá con seguir las instrucciones intuitivas del BOT para usarlo correctamente. En caso de querer información sobre sus posibilidades con más detalle se puede usar el comando `/help` tal y como dice el mensaje inicial.

Para probar la característica principal, es decir, la traducción de audio a audio, tendremos que seleccionar un idioma origen mediante `/fromlanguage` seguido de una *IETF language tag*[26]. Tras esto enviamos un mensaje de voz al BOT y listo.

Bibliografía

[1] Creative Commons. *Librería python-telegram-bot*. https://python-telegram-bot.org.

[2] Docker. *Home*. https://www.docker.com.

[3] Dubzer. *TgTranslator*. https://github.com/Dubzer/TgTranslator.

[4] FastAPI. *FastAPI framework, high performance, easy to learn, fast to code, ready for production*. https://fastapi.tiangolo.com.

[5] FFmpeg. *A complete, cross-platform solution to record, convert and stream audio and video*. https://ffmpeg.org.

[6] Glassdoor. *Sueldos para el puesto Jefe De Proyecto en España*. https://www.glassdoor.es/Sueldos/jefe-de-proyecto-sueldo-SRCH_KO0,16.htm. 04/10/2022.

[7] Google. *Google Cloud: Speech-to-Text*. https://cloud.google.com/speech-to-text. (2022).

[8] granyk. *Voicos*. https://github.com/graynk/voicos.

[9] Heroku. *Cloud Application Platform*. https://www.heroku.com.

[10] James Robert. *Librería pydub*. https://pypi.org/project/pydub/.

[11] *Language Translator*. https://t.me/lang_translate_bot.

[12] Lingvanex. *Lingvanex Translator*. https://t.me/lingvanexbot.

[13] Mike Cotterell y Bob Hughes. *Software project management. 5.a ed. Tata McGraw Hill Education Private Limited.* (2011, ISBN: 978-0-07-107274-8.)

[14] pndurette. *Librería gTTS*. https://pypi.org/project/gTTS/.

[15] *Silero STT*. https://t.me/silero_audio_bot.

[16] Telegram. *BotFather*. https://telegram.me/BotFather.

[17] Telegram. *Bots: An introduction for developers*. https://core.telegram.org/bots. (2022).

[18] Terry Yin. *Librería translate*. https://pypi.org/project/translate/.

[19] *Text to Speech Bot*. https://t.me/texttsbot.

[20] Tokio School. *Sueldo programador Python: descubre su salario*. https://www.tokioschool.com/noticias/sueldo-programador-python-descubre-su-salario/. 11/02/2022.

[21] *TranscriberBot*. https://t.me/transcriber_bot.

[22] Translated LABS. *MyMemory*. https://mymemory.translated.net.

[23] Wikipedia. *Bots conversacionales*. https://es.wikipedia.org/wiki/Bot_conversacional. (2022).

[24] Wikipedia. *Chain of responsibility pattern*. https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern. (2022).

[25] Wikipedia. *GitHub*. https://es.wikipedia.org/wiki/GitHub. (2022).

[26] Wikipedia. *IETF language tag*. https://en.wikipedia.org/wiki/IETF_language_tag. (2022).

[27] Wikipedia. *Overleaf*. https://en.wikipedia.org/wiki/Overleaf. (2022).

[28] Wikipedia. *PyCharm*. https://es.wikipedia.org/wiki/PyCharm. (2022).

[29] Wikipedia. *Python*. https://es.wikipedia.org/wiki/Python. (2022).

[30] Wikipedia. *Síntesis del habla*. https://es.wikipedia.org/wiki/Síntesis_de_habla. (2022).

[31] Wikipedia. *Sourcetree*. https://en.wikipedia.org/wiki/Astah*. (2022).

[32] Wikipedia. *Speech Recognition*. https://en.wikipedia.org/wiki/Speech_recognition. (2022).

[33] Wikipedia. *Traducción asistida por computadora*. https://es.wikipedia.org/wiki/Traducción_asistida_por_computadora. (2022).

[34] Yandex. *Yandex.Translate*. https://telegram.me/ytranslatebot.