Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Transcripción de audio a texto para mensajería instantánea

Estudante: Alejandro Pereira Carballo

Dirección: Laura M. Castro Souto

A Coruña, noviembre de 2022.

A mi familia y amigos

Agradecimientos

Gracias a mis seres queridos y a la directora de este trabajo, Laura Milagros Castro Souto. Sin su apoyo y orientación este proyecto habría sido imposible.

Resumen

Este proyecto consiste en el estudio, diseño e implementación de un servicio de transcripción de audio en una aplicación de mensajería, Telegram, de código abierto. La plataforma sobre la que se ha desarrollado la funcionalidad es Android, usando el lenguaje Java.

Con este trabajo se pretende mejorar la calidad de vida de los usuarios de la aplicación, pensando en personas con problemas de audición o a las que les es más conveniente leer que escuchar un audio.

Este trabajo me ha permitido estudiar un ejemplo de aplicación de nivel empresarial, así como aprender sobre el desarrollo en Android y la participación en la comunidad del software de código abierto.

Abstract

This project consists of the assessment, design and implementation of a transcription service in an open source messaging app, Telegram. The platform on which this service was developed was Android, using the Java programming language.

This program is intended to improve the quality of life of the app's users, especially people with hearing problems or that would rather read than listen to an audio.

This project has allowed me to study the insides of an enterprise level application, and also learn about Android development and how it is to participate in the open source community.

Palabras clave:

- Telegram
- Android
- Java
- Mensajería instantánea
- Transcripción
- Audio
- Computación en la nube

Keywords:

- Telegram
- Android
- Java
- Instant messaging
- Transcription
- Audio
- Cloud computing

Índice general

Índice de figuras

Capítulo 1
Introducción

La finalidad de este trabajo es la incorporación de la funcionalidad de transcripción de audio en una aplicación de mensajería. La aplicación en cuestión es Telegram, dado que ofrece su código fuente con una licencia libre. El código de este proyecto se puede encontrar en su repositorio de GitHub [1].

Las aplicaciones de esta funcionalidad abarcan desde una comodidad añadida para situaciones en las que no sea oportuno o posible escuchar un audio, como pueden ser una sala de espera o el transporte público; hasta mejoras en la calidad de vida para personas con discapacidad auditiva, por ejemplo.

1.1 Mensajería instantánea

La mensajería instantánea existe desde la llegada de los sistemas operativos multi-usuario, antes de la existencia de Internet, a mediados de los años 60; y se ha extendido a la mayoría de plataformas que han ido apareciendo con el tiempo, como los ordenadores de escritorio, las PDAs, o, en el actual eslabón evolutivo, los *smartphones*.

Con la popularización de los ordenadores personales, los portátiles y los teléfonos con conectividad a internet, así como con la simplificación en el uso de estos, el público objetivo de las aplicaciones de mensajería se ha expandido del personal científico y los *geeks* a prácticamente todo el mundo. Otros motivos que explican este *boom* en uso son la rapidez con la que dos o más personas pueden comunicarse, la conveniencia de enviar un simple mensaje en vez de tener que llamar, o la capacidad de mantenerse en contacto con gente con la que no sería posible con otros medios.

Se consideran mensajería instantánea los *chats online* (salas de conversación en red) que permiten el envío y la recepción de texto a través de Internet. Esta tecnología vivió una revitalización con la aparición de las aplicaciones móviles, siendo cruciales *BlackBerry Messenger* [2] en 2005 y *WhatsApp* [3] en 2009. Fue en esta segunda primavera cuando nacieron las

demás aplicaciones de mensajería que se usan a día de hoy, como *LINE* [4], *WeChat* [5], o *Telegram* [6]. Todas estas aplicaciones son desarrolladas por grandes empresas, como *Meta* o *Tencent*, y si bien *Telegram* no es realmente una excepción (está desarrollada por la organización con su mismo nombre), es la única que ofrece el código fuente de manera libre.

La funcionalidad más básica de la mensajería instantánea es, como decimos, la transmisión de texto, pero a día de hoy todas las aplicaciones de uso extendido incorporan, al menos: texto, emojis, mensajes de voz, llamadas, *stickers*, y archivos multimedia de varios tipos. Sin embargo, todas carecen de la funcionalidad presentada en este trabajo: la transcripción de mensajes de audio a texto. Es verdad que existen métodos para llevarlo a cabo, como pueden ser *bots* o aplicaciones de terceros a los que reenviar el audio, pero ninguna permite hacerlo rápidamente y de manera directa, sin pausar la conversación.

De cara a poder implementar una solución para esta carencia tan generalizada, es preciso escoger una aplicación a la que contribuir. De las mencionadas anteriormente, las que mayor número de usuarios acumulan son *WhatsApp* y *Telegram*. Y, dado que la única que permite acceder y modificar el código fuente es *Telegram*, será esta el objeto de este trabajo.

1.2 Identificación de necesidades

Como se mencionó previamente, la transcripción de mensajes de audio proporciona comodidad y una vía de comunicación para personas en circunstancias particulares. Según la Organización Mundial de la Salud, se estima que alrededor de 1500 millones de personas viven con algún grado de pérdida de audición [7]; un número importante de personas que se podrían beneficiar de esta tecnología.

Con una mirada no enfocada en la salud sino en la diversidad, la transcripción también puede ser de ayuda en el día a día de cualquier persona usuaria. Si bien las estadísticas se han visto alteradas por la pandemia de los últimos años, cerca de 4000 millones de personas en el mundo han hecho uso del transporte público al menos una vez [8]. En estos casos, no solo es de buena *etiqueta* tratar de no molestar al resto de pasajeros, sino que es posible estar discutiendo algún tema privado y no querer que un desconocido pueda escuchar detalles al respecto. Este mismo caso se da en otros lugares, como consultas médicas o bibliotecas, haciendo todavía más evidente la utilidad y conveniencia de la transcripción de mensajes de audio.

Como último ejemplo, están los bebés y niños pequeños. Todos reconocemos la familiaridad de esa escena en la que una madre, padre o cuidador/a está tratando de dormir al bebé a su cargo, para que a continuación el más mínimo ruido lo despierte. En estas situaciones, se puede reducir el número de amenazas al sueño de los más jóvenes mediante la tecnología de transcripción de voz; una funcionalidad que agradecerían muchos progenitores en todo el mundo.

1.2.1 Transcripción de audio

La transcripción de audio consiste en la transformación de secuencias de sonido en texto. Esta capacidad se usa desde hace tiempo en diversos ámbitos; ejemplos notables son la transcripción de entrevistas para periódicos y medios escritos, el registro de conversaciones, reuniones en empresas y juicios; o como una de las formas que emplean algunos autores para desarrollar sus historias de manera ágil.

Tradicionalmente, la transcripción se ha llevado a cabo manualmente, con una persona escuchando y escribiendo. Sin embargo, es posible automatizar este proceso mediante inteligencia artificial, obteniendo resultados mucho más rápidos y con un menor coste, aunque con la desventaja de que pueden ser más imprecisos en comparación con los producidos por un humano.

En el siguiente capítulo se tratan en mayor profundidad las tecnologías y herramientas existentes para la transcripción de audio.

1.3 Objetivos y metodología

El objetivo central de este proyecto es, pues, la inclusión de un servicio de transcripción de audio en la aplicación de mensajería instantánea Telegram, teniendo en cuenta la estética y el rendimiento establecidos por el resto de componentes de la misma para que la funcionalidad quede integrada lo mejor posible.

Para conseguir esto, el proyecto se divide en hitos o pequeñas metas con las que medir su avance global. Estas metas son las siguientes:

O1 Creación de una opción *mock* dentro del menú de acciones de un texto.
Los sub-objetivos de este punto son:

1. Familiarizarse con el código fuente de Telegram.
2. Ubicar la creación del menú y la inserción de opciones.
3. Introducir una opción personalizada sin funcionalidad.

O2 Creación de una alerta que muestre el contenido del mensaje seleccionado.
Los sub-objetivos de este punto son:

1. Ubicar la llamada a la alerta de traducción.
2. Crear la alerta de transcripción a partir de la de traducción.
3. Vincular la opción añadida previamente con la llamada a la alerta.

O3 Transcripción del audio de entrada a texto.
Los sub-objetivos de este punto son:

1. Obtener los parámetros generales de los audios enviados con Telegram.
2. Conseguir el fichero que contiene el audio seleccionado.
3. Implementar la llamada al servicio de transcripción escogido.
4. Procesar el resultado y mostrarlo en la alerta.

O4 Refactorización, optimización y pruebas.
Los sub-objetivos de este punto son:

1. Limitar la aparición de la opción personalizada a los mensajes de audio.
2. Comprobar la usabilidad y rendimiento de la funcionalidad desarrollada.
3. Escoger un nombre e icono apropiados para la opción.

Con esta hoja de ruta en mente, se decidió que una metodología ágil era lo más conveniente. En la práctica, se utilizó una versión modificada de Scrum [9] y Kanban [10], ya que se consideró contraproducente aplicar estas metodologías de manera estricta en un desarrollo unipersonal. El proceso de desarrollo consistió en dividir las tareas en fragmentos del menor tamaño posible, establecer el orden de prioridades de dichos fragmentos, y proceder a cumplirlos uno a uno. Cada paso de este método se discutió en reuniones que se celebraron cada una o más semanas, en las que también se decidía cuál era el progreso apropiado que se debería alcanzar para la siguiente reunión. En el capítulo de seguimiento se tratan en mayor profundidad los detalles de los *sprints* realizados. Los términos técnicos y herramientas se explicarán en capítulos posteriores, así como el razonamiento tras las decisiones tomadas.

Capítulo 2

Tecnologías y técnicas de transcripción

En este capítulo, ofreceremos información del contexto técnico del proyecto, útil para la comprensión posterior de los retos enfrentados y las decisiones tomadas. Por ejemplo, para el desarrollo de este proyecto, ha habido una serie de limitaciones y procesos diferentes a la compilación de aplicaciones Android más habituales. Una de las restricciones más relevantes es la prohibición del uso de librerías de terceros no incluidas previamente, como ocurre con las herramientas de transcripción mencionadas más adelante.

2.1 Lenguaje y contexto de la aplicación

Telegram dispone de aplicaciones nativas para Android, iOS, Windows, MacOS, y un gran número de distribuciones Linux, además de un cliente web para navegador. Cada una de estas versiones es un producto separado, con su propio código fuente escrito en un lenguage distinto al resto.

Este proyecto se basa en la versión de Telegram para Android [11], escrita en Java, como es habitual con aplicaciones de este sistema operativo. El motivo de esto es simple: las versiones más utilizadas son las de móvil, Android e iOS, de las cuales la primera es la que menos complicaciones presenta a la hora de compilar y probar el código. Para la compilación de la versión de Android, solo es necesario un entorno capaz de ejecutar Docker [12], mientras que para iOS es necesario disponer de un dispositivo con el sistema operativo MacOS. En el primer caso, para realizar pruebas en dispositivos diferentes solo hay que seleccionar e instalar el paquete correspondiente, y los dispositivos Android virtuales pueden ejecutarse sobre cualquier sistema de escritorio; por el contrario, en el caso de iOS, tanto la instalación en equipos físicos como la creación de virtuales requiere MacOS, o de un *setup* de mayor complejidad.

2.1.1 Restricciones

Durante el desarrollo de cualquier aplicación, sea o no para Android, suele ser habitual el uso de librerías no incluidas en el Software Development Kit (SDK) nativo del entorno para agilizar o introducir mejoras, bien porque ofrecen funcionalidades superiores a las ya presentes en dicho SDK, o bien porque poseen ciertas capacidades que no es necesario replicar si ya están disponibles desde otras fuentes.

Por tanto, es evidente que una de las restricciones más relevantes a la hora de crear una aplicación, sobre todo una tan compleja como Telegram, es la prohibición sobre el uso de estas librerías. En primer lugar, la aplicación se compila mediante un contenedor de Docker personalizado, con una configuración concreta que tendría que modificarse para incluir cualquier librería nueva. Este proceso no es trivial, ya que simplemente añadir la dependencia resulta en un error de compilación. Con suficiente investigación, habría sido posible introducir todas las modificaciones deseadas sin causar ninguna disrupción en el proceso de compilación. Sin embargo, es un esfuerzo que se consideró innecesario, teniendo en cuenta el resto de motivos: Telegram es un proyecto grande y de considerable complejidad, e introducir nuevas dependencias puede crear problemas de compatibilidad, y aumentar su tamaño de manera importante; por otro lado, al no usar dependencias se facilita considerablemente el trabajo de los desarrolladores encargados de mantener y distribuir el proyecto Telegram, además de que no hay que preocuparse por las licencias de uso de dichas librerías, que pueden ser más o menos permisivas que la de Telegram. Por último, al evaluar la dificultad de desarrollar todas las funcionalidades, se consideró que no era un coste suficiente como para ignorar todas estas premisas.

Por estos motivos no fue posible emplear librerías como, por ejemplo, las ofrecidas por Google Cloud para simplificar la interacción con sus Application Programming Interface (API), o alguna de las existentes para la creación de JWT, por lo que hubo que desarrollar las funcionalidades a mano, utilizando solo las funciones y métodos incluidos en las APIs de Android y Java, y en la propia aplicación de Telegram.

2.2 Transcripción y tecnologías

Las técnicas de transcripción de audio a texto pueden dividirse en dos clases: manuales y automáticas. La transcripción manual la realizan personas que escuchan el audio y escriben a mano, o utlizando un ordenador, las palabras que escuchan. Este método tiene la ventaja de que un ser humano es capaz de interpretar sonidos en función del contexto, lo que permite que grabaciones de calidad inferior o con interferencias den resultados casi tan precisos como las de calidad superior, que de por sí poseen un índice de fidelidad muy alto.

Por otro lado, la transcripción automática emplea inteligencia artificial para medir los

parámetros de las ondas de sonido, compararlos con modelos ya existentes en el idioma indicado, y devolver uno o varios resultados que el sistema considera cercanos al contenido real del audio. Por regla general, este método es notablemente menos preciso que la transcripción manual, ya que no se realizan correcciones y consideraciones en función del contexto, o las que se realizan no son tan completas aún como las realizadas por las personas. Sin embargo, el tiempo que el método manual necesita para devolver resultados es, como mínimo, igual a la duración del audio de entrada; mientras que el automático puede generarlos en apenas segundos.

Ambas formas presentan diferentes fortalezas, y aunque hoy en día está mucho más extendida la automática, sigue habiendo servicios que ofrecen transcripciones manuales bajo demanda; y clientes que los consumen, como periodistas o escritores. Entre estos servicios, los más conocidos son *Alphatrad* [13], *Transcripciones.net* [14], o *GoTranscript* [15].

Por otro lado, la transcripción automática tiene servicios de pago, como los manuales, pero también ofrece métodos gratuitos y maneras de integrarla con otras aplicaciones y servicios. Algunos de los proveedores en la nube más conocidos son *Amberscript* [16], *VEED* [17] o *Happyscribe* [18], o en caso de buscar una integración mediante API, *Google*, *IBM* [19] o *Microsoft* [20]. Sin embargo, también hay aplicaciones que permiten transcribir mensajes pequeños, como la aplicación de teclado de Google, *Gboard* [21], o *Speechnotes* [22].

En el caso de Android, existe incluso un API nativa que permite usar los modelos que se instalan por defecto con el sistema operativo de Google, con el fin de que funcionen tanto el Asistente de Google [23] como Gboard, además de otras funcionalidades propias del sistema. Al inicio del proyecto, este fue el método con el que se intentó implementar la funcionalidad, pero debido a una serie de problemas, que se explicarán en el capítulo sobre implementación, fue necesario recurrir a un servicio en la nube de los mencionados anteriormente, *Google Cloud*. El motivo principal de no emplear esta solución desde el principio fue el deseo de garantizar la privacidad y seguridad de los mensajes de cada usuario, primando siempre su custodia local. Por otro lado, existe una ventaja al usar esta implementación, y es que no es dependiente del sistema operativo utilizado, por lo que introducirla en las demás versiones de Telegram sería relativamente sencillo.

2.2.1 Google Cloud Speech-to-Text

Google Cloud es una plataforma de computación y almacenamiento en la nube ofrecida por Google. Ofrece diversas APIs web y librerías cliente para los lenguajes más utilizados. Este proyecto en concreto se vale del *endpoint Speech-to-Text* en su versión REST.

Este servicio ofrece distintos parámetros y modelos de reconocimiento para transcribir audio de manera síncrona, asíncrona, o en tiempo real. En este trabajo se utiliza el reconocimiento síncrono, que limita la duración del audio a transcribir a un minuto, pero permite

realizar la operación sin subir el fichero a la nube.

Aparte de Google Cloud, otros servicios de transcripción en la nube que se tuvieron en cuenta fueron los de IBM y Microsoft. El servicio de IBM es, según opiniones de varios usuarios, complicado de configurar y poner a punto, y requiere mantenimiento. El de Microsoft no tiene, a priori, ningún problema o complicación en comparación con Google Cloud. Sin embargo, este último ofrece 60 minutos de transcripción gratuita cada mes, además de la ya mencionada opción de no almacenar los datos subidos al servicio para su uso por parte de Google. Además, Telegram ya usa la API de traducción de Google, por lo que se evita la posible distribución de los datos de los usuarios a más empresas.

2.2.2 JSON Web Token

JSON Web Token (JWT), es un estándar web para la creación de datos con o sin encriptación o firma. Un JWT suele firmarse con un secreto o clave privada.

Con la limitación sobre el uso de librerías, fue preciso diseñar una función que recogiera los datos relevantes, y los usara para generar un token con el que autorizar y autenticar las llamadas a la API de Google Cloud.

Habitualmente, el JWT se usa contra la API de autenticación de Google, que se encarga de responder con un token de acceso con el que realizar la llamada a la API deseada. Sin embargo, para ciertos servicios, como el de transcripción, no es necesario el paso intermedio, sino que se puede emplear el JWT como token de acceso directamente.

2.2.3 Base64

Base64 es un sistema de numeración posicional que usa 64 como base, ya que es la mayor potencia que puede ser representada usando sólo los caracteres imprimibles de la tabla ASCII. Existen distintas variantes: coinciden en los primeros 62 dígitos, que son caracteres alfanuméricos, y se diferencian en los dos últimos. Además, existen distintas formas de convertir datos a Base64.

Dentro de la API de Android, existen varios parámetros configurables para convertir datos a Base64, como el final de línea, si añadir *padding* al final o no, si omitir los saltos de línea, o si usar la variante segura para URLs y nombres de fichero. En este proyecto se ha usado cada una de estas variantes en función de los requisitos de la API de transcripción.

2.2.4 Clave privada

Una clave privada es una de las claves generadas en un sistema criptográfico asimétrico [24]. Una de sus funciones es la autenticación, es decir, verificar la identidad del usuario o servicio que intenta acceder a un recurso.

En este trabajo, se utiliza una clave privada generada por Google Cloud para autenticar la peticiones a la API de transcripción, concretamente en la firma del JWT.

Capítulo 3

Telegram

En este capítulo, ofreceremos información del producto concreto Telegram como aplicación Android desarrollada en Java, objetivo específico de este trabajo. Esta información es imprescindible para la profundización que seguirá en los capítulos de implementación y pruebas.

3.1 Licencia de código abierto: GNU v2

El repositorio de Telegram tiene una licencia de código abierto *GNU General Public License v2.0* [25]. Esta licencia permite la distribución, modificación, y usos comercial y privado del código, al igual que indica que no ofrece ninguna garantía ni responsabilidad sobre su uso. También obliga a todo el que modifica o usa el código a incluir en su proyecto una licencia igual o similar, así como de incluir las licencias y derechos de autor a la hora de distribuirlo, que no puede ocurrir sin antes liberar el código al público. También obliga a indicar todos los cambios efectuados sobre el código original.

3.2 Compilación

Habitualmente, las aplicaciones de Android se compilan mediante Gradle [26], directamente o, más comúnmente, a través Android Studio. El código de Telegram no es diferente en ese aspecto; sin embargo, se desvía en la ejecución: utiliza un contenedor de Docker personalizado, creado a partir del fichero *Dockerfile*, incluido en el propio repositorio. La aplicación de Docker y el Dockerfile se explicarán en mayor detalle en el siguiente capítulo.

Las instrucciones para llevar a cabo la compilación se pueden encontrar en el *README* del repositorio oficial. En él, se explica que es necesario solicitar un ID para usar la API de Telegram, así como indicar el nombre y distintos datos de la aplicación que va a usar dicho ID. Este paso es imprescindible, ya que la API solo responde a llamadas que proveen un identificador

registrado, y el que viene incluido en el código fuente no es más que un *placeholder*.

A continuación se provee un enlace a la página web de Telegram, al apartado de *Reproducible Builds* [27]. En esta página se explica, paso a paso, como descargar y compilar el código de una versión concreta de la aplicación Android de Telegram, y como extraer la aplicación instalada en un dispositivo para compararla con la versión compilada. Esto se hace para poder demostrar que la aplicación ofrecida en los canales oficiales es la misma que se obtiene de compilar el repositorio. En este proyecto, solo hace falta seguir los pasos hasta la generación del APK, ya el objetivo no es crear una aplicación nueva, sino extender la oficial.

Volviendo al repositorio, también aparecen una serie de pasos que terminan de explicar los requisitos de compilación. No todos los pasos son necesarios, ya que algunos están pensados para publicar la aplicación. Los imprescindibles son, en resumen, descargar el código fuente, abrir el proyecto en Android Studio teniendo cuidado de no importarlo, y rellenar los valores de las variables que aparecen en la clase *BuildVars*.

3.3 Diagramas

Tanto el aspecto visual como la implementación de la transcripción se basaron desde el principio en la funcionalidad de traducción, introducida poco antes del comienzo del proyecto; comparten características clave, como el uso de un mensaje a modo de entrada, la llamada a una API que procesa dicha entrada, y la presentación no permanente del resultado final en forma de texto.

Dicho esto, fue necesario ubicar en el código diferentes puntos: la inserción de la opción *"Traducir"* en el menú flotante, la creación de la alerta que contiene la traducción, y el método para obtener dicha traducción. Para encontrar todo esto fue preciso investigar la estructura del código, ya que no se encontró ningún tipo de diagrama con la jerarquía o interacción entre clases. A partir de esta investigación se crearon los diagramas de arquitectura que se presentan a continuación, utilizando el modelo de representación C4 [28], centrados exclusivamente en la función de traducción.

3.3.1 Funcionalidad usada como modelo: traducción de texto

Las figuras 3.1 y 3.2 describen la interacción entre el usuario, el sistema Telegram y el proveedor de traducciones. Ambos diagramas definen también, con exactitud, los integrantes y dependencias de la funcionalidad de transcripción diseñada e implementada en este trabajo, con la excepción de que, en este caso, Google Cloud provee transcripciones en lugar de traducciones.

Figura 3.1: Diagrama C4 de nivel de contexto del sistema Telegram

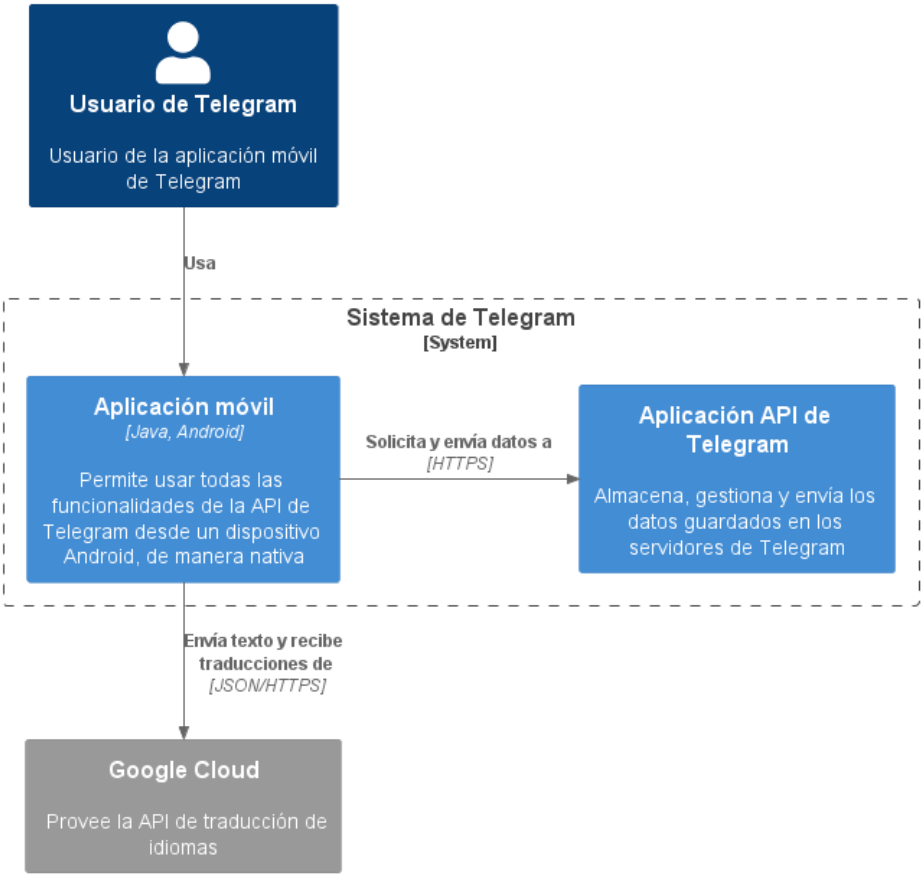


Figura 3.2: Diagrama C4 de nivel de contenedor del sistema Telegram

La figura 3.3 también es muy similar en ambas funcionalidades, excepto que para la transcripción no es necesario el detector de idioma, ya que se asume que es el del sistema; y la alerta pasa a ser de transcripción, no de traducción, que se diferencia en los datos a presentar. Esta diferencia se muestra más tarde, en la figura 3.7.

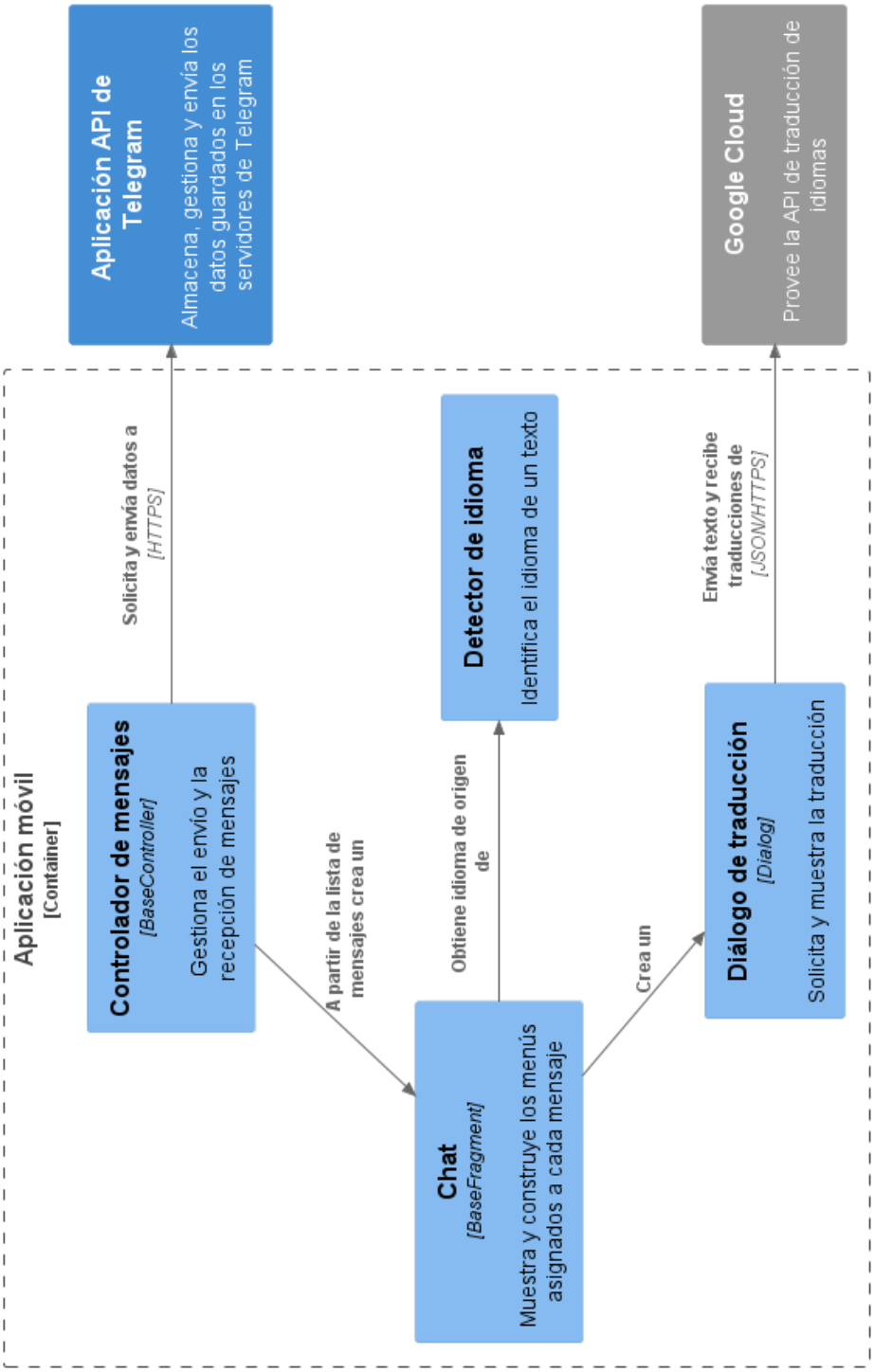


Figura 3.3: Diagrama C4 de nivel de componente de la aplicación Android de Telegram

3.3.2 Proceso de construcción de vista y diferencias funcionales

A la hora de introducir la nueva alerta, se tuvieron que crear/modificar dos clases: *ChatActivity* y *TranscribeAlert*. La primera se encarga de gestionar distintos aspectos de la pantalla de chat, mientras que la otra sirve para mostrar el resultado de la transcripción al usuario. La interacción entre las clases y el usuario se muestra en el diagrama de la figura 3.4.

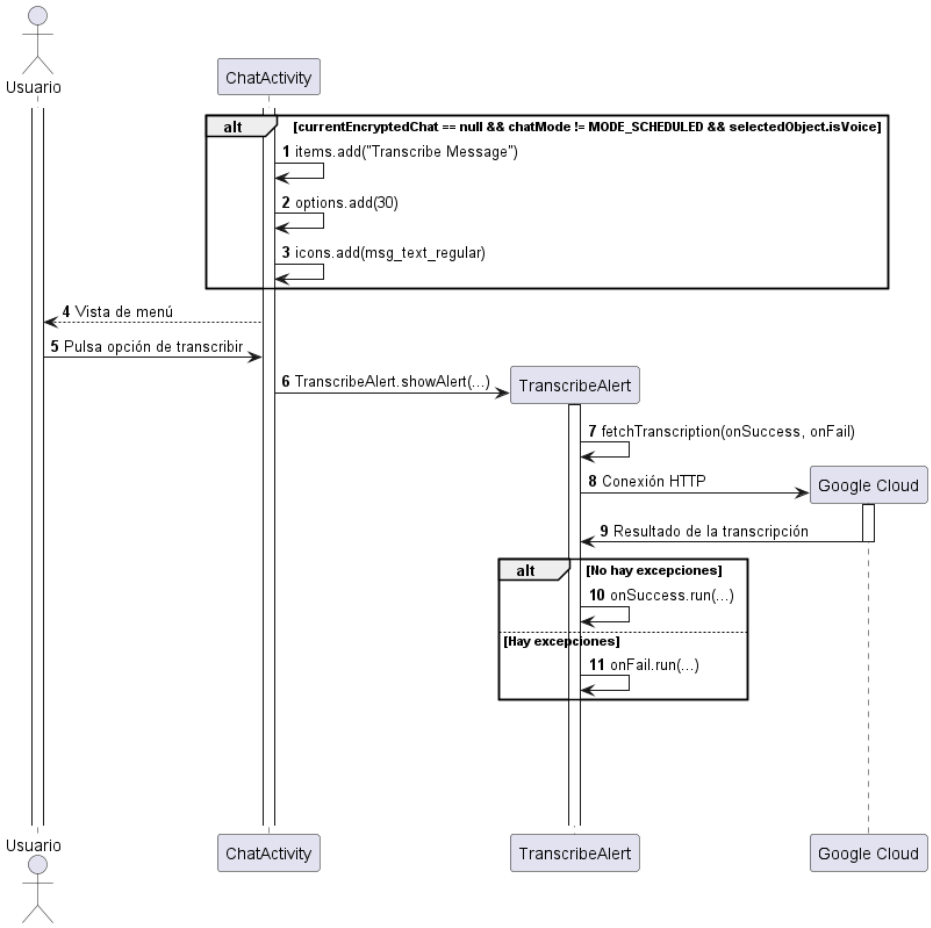


Figura 3.4: Diagrama de secuencia de la transcripción

Este diagrama se aplica para todas las opciones disponibles en un chat hasta el paso 4. A partir de ahí entran en juego las modificaciones hechas para incorporar la funcionalidad objetivo de este proyecto. Cuando el usuario pulsa el botón de transcribir, se dispara el método *showAlert* de la clase *TranscribeAlert*, que ejecuta el constructor de dicha clase. Durante la construcción del objeto, se realiza la conexión a Google Cloud, y se muestra el resultado o un mensaje de error, según corresponda. Como indica el diagrama, el control de la aplicación pasa a manos de la alerta en el momento en que el usuario selecciona la opción.

3.4 Interfaz de usuario

La vista principal de Telegram es la lista de *chats* o conversaciones. Es una vista casi exactamente igual en la mayoría de aplicaciones de mensajería. La vista de mensajes de un chat también es muy similar, como se observa en la figura 3.5, que contiene un chat de ejemplo con un bot inactivo; sin embargo, las diferencias aparecen al presionar un mensaje. Como se aprecia en la figura 3.6, aparece un menú con acciones a aplicar sobre el mensaje seleccionado, entre las que destacan la de traducir y la de transcribir, introducida en este proyecto. También se puede ver cómo las opciones disponibles varían en función del tipo, autor o idioma del mensaje.

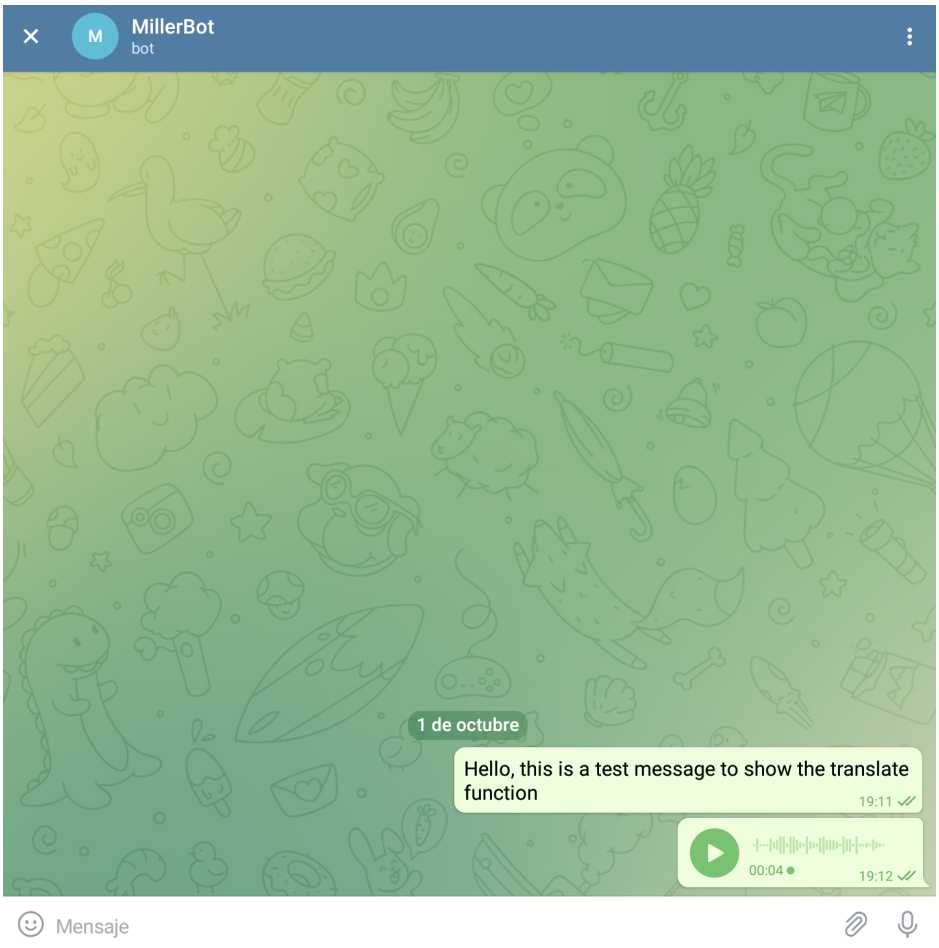
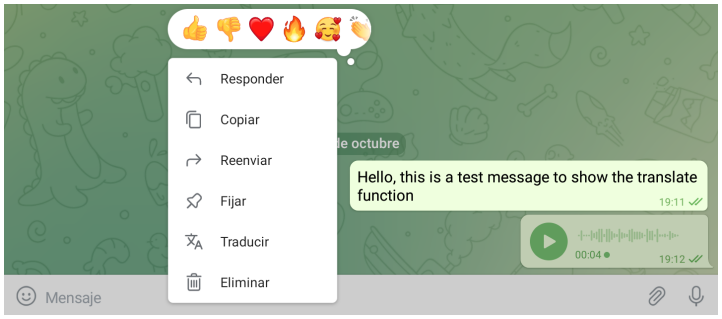
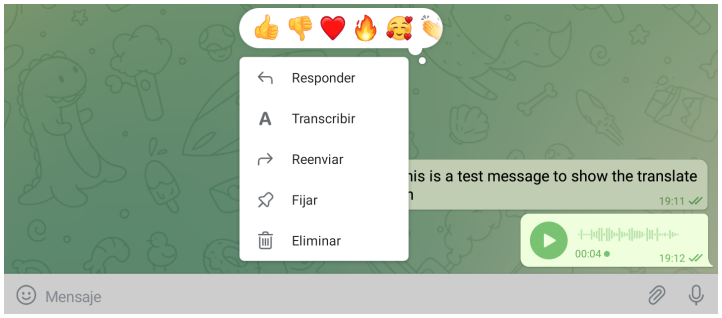


Figura 3.5: Vista de mensajes dentro de un chat



(a) Acciones aplicables a texto

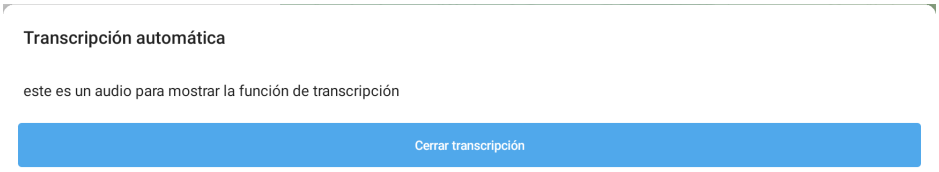


(b) Acciones aplicables a audio

Figura 3.6: Menús de acciones sobre mensajes

Como se mencionó anteriormente, la transcripción se apoyó en la recientemente implementada traducción, y el resultado es una interfaz casi igual en ambos casos. Esto se hizo a propósito, para mantener la estética distintiva de Telegram. La principal diferencia es la eliminación del subtexto indicando los idiomas de origen y destino. A mayores, aunque no es común que aparezca salvo en condiciones de mala conexión, el texto parpadeante que indica el proceso de traducción se ha cambiado por un mensaje de carga. En la figura 3.7 se muestran, lado a lado, las alertas propias de estas funcionalidades.

(a) Alerta de traducción



(b) Alerta de transcripción

Figura 3.7: Comparativa de las alertas de traducción y transcripción

Capítulo 4

Implementación

En este capítulo se explicará brevemente la implementación inicial mediante la API nativa de Android, que resultó inviable. También se tratará con mayor detalle la implementación final de la llamada al servicio de transcripción en la nube, así como la creación y el uso de las herramientas necesarias para alcanzar el objetivo final.

4.1 Extensión de la interfaz gráfica

Como ya hemos comentado, la primera modificación del código original se efectuó en el menú que aparece al presionar un mensaje. La clase encargada de gestionar dicho menú, entre otras cosas, es *ChatActivity*, concretamente el método *createMenu*. Aquí, existe una serie de sentencias condicionales anidadas que dictaminan, en función del tipo de mensaje y chat, las acciones que pueden aparecer. Como se muestra en la figura 4.1, cada opción consiste en un icono, un texto y un número entero, que se añaden a sus respectivas colecciones cuando la opción en cuestión debe aparecer. Este código es necesario y suficiente para que aparezca una opción nueva en el menú, aunque no tenga funcionalidad incluida.

```
} else if (selectedObject.isVoice() && !getMessagesController().isChatNoForwards(currentChat)) {
    items.add(LocaleController.getString( key: "TranscribeMessage", R.string.TranscribeMessage));
    options.add(30);
    icons.add(R.drawable.msg_text_regular);
}
```

Figura 4.1: Incorporación de componentes entre las opciones gráficas

La siguiente y última modificación dentro de esta clase ocurre en este mismo método. Al poco de terminar la composición del menú, hay un bucle que itera la colección de enteros, y establece *callbacks* que se disparan al pulsar una de las acciones. Para la opción se transcripción, se escogió un valor que no estaba en uso, y se escribió el código que aparece en la figura 4.2, basado en la llamada a la alerta de traducción. Lo que se hace diferente en este es utilizar

un método estático de la clase *FileLoader*, ya incluida en el código original, para obtener la ruta del archivo de audio del mensaje. Esta ruta se utiliza como parámetro para la creación de la alerta.

```
if (option == 30) {
    // "Transcribe" button
    File document = FileLoader.getPathToAttach(selectedObject.getDocument());
    TranscribeAlert.OnLinkPress onLinkPress = (link) -> {
        didPressMessageUrl(link, longPress: false, selectedObject, v instanceof ChatMessageCell ? (ChatMessageCell) v : null);
    };
    cell.setOnClickListener(e -> {
        if (selectedObject == null || i >= options.size() || getParentActivity() == null) {
            return;
        }
        TranscribeAlert.showAlert(getParentActivity(), fragment: this, document, selectedObject.getDuration(),
                noforwards: currentChat != null && currentChat.noforwards, onLinkPress);
        scrimView = null;
        scrimViewReaction = null;
        contentView.invalidate();
        chatListView.invalidate();
        if (scrimPopupWindow != null) {
            scrimPopupWindow.dismiss();
        }
    });
}
```

Figura 4.2: Vinculación de la acción y la llamada a la alerta

Por último, está la ya mencionada alerta, *TranscribeAlert*. Esta se creó a partir de la de traducción, *TranslateAlert*, y se eliminaron las referencias a texto de entrada e idiomas de origen y destino. Se modificaron el constructor de la clase, que aparece en la figura 4.3, y el método dedicado a la conexión HTTP.

```
public TranscribeAlert(BaseFragment fragment, Context context, File file, int duration, boolean noforwards, OnLinkPress onLinkPress) {
    super(context, R.style.TransparentDialog);

    this.onLinkPress = onLinkPress;
    this.noforwards = noforwards;
    this.fragment = fragment;
    this.duration = duration;
    this.file = file;
```

Figura 4.3: Constructor de la alerta

4.2 Transcripción nativa en Android

Si bien los cambios para extender la interfaz, una vez analizado y estudiado el código del sistema, fueron triviales, el método que provee las transcripciones fue más complicado. La primera línea de trabajo contemplaba la transcripción nativa.

Como se ha explicado previamente en varias ocasiones, el SDK de Android ofrece un servicio de transcripción de serie. Este servicio es el utilizado por aplicaciones como el Asistente de Google, y está orientado a dictados de muy corta duración, de unos pocos segundos. Una

vía que podría permitir el uso de este servicio local sería dividir los audios de entrada en fragmentos de corta duración, y transcribirlos uno a uno, pero dividir un fichero de audio desde el código de la aplicación es relativamente complejo, además que se perdería el contexto y los resultados serían menos precisos; o directamente erróneos si se cortase alguna palabra, cambiando la frase por completo.

No obstante, se determinó otra opción posible. Para llevar a cabo la llamada al servicio es necesario proveer un objeto *handler* que gestione las acciones de inicio y fin, y que se encargue de las excepciones y errores. Usualmente, se provee un objeto creado a partir de una de las clases ya implementadas en el SDK; sin embargo, se acepta cualquier objeto que implemente una interfaz pública relativamente sencilla. Por tanto, la alternativa pasaba por implementar dicha interfaz, haciendo que al final de cada periodo de transcripción se compruebe si se ha alcanzado el final del audio, y si no es el caso, se almacene el resultado provisional y se empiece otra transcripción con un *offset* igual a la duración del audio ya procesado. Cuando se alcanza el final, se unen todos los resultados parciales. Esto ofrece dos inconvenientes: primero, el consumo de recursos es relativamente alto, ya que la API no está pensada para uso prolongado, y segundo, debe estar instalado en modelo de voz del idioma de entrada; aunque también hay que tener en cuenta que el uso de esta función va a ser puntual, y que el modelo suele instalarse por defecto con el resto de aplicaciones de Google.

En última instancia, hubo un obstáculo insalvable, y es que el uso de ficheros como entrada (necesario para poder dividir el audio en fragmentos y procesarlo de ese modo) no está soportado para versiones del sistema inferiores a Android 12. En versiones anteriores solo es posible utilizar como entrada al servicio de transcripción de serie el sonido registrado por el micrófono del dispositivo. Como parche, se intentó redirigir el contenido del fichero de audio a la entrada del micrófono, como es posible hacer en la terminal de Linux, pero sin éxito. También se probó a crear un micrófono virtual y usarlo en vez del físico, pero ninguna de estas tácticas dio resultado.

4.3 Transcripción en la nube

Descartada la transcripción local, reorientamos la aproximación para cumplir el objetivo usando transcripción en la nube.

Para utilizar el servicio de Google Cloud ya mencionado en capítulos anteriores, fue preciso escribir código para construir un objeto JSON con los parámetros requeridos por la API de Google, y crear un flujo de salida por el que enviar dicho objeto. También se modificó ligeramente el código para la recepción de la respuesta, para acomodarlo al esquema proporcionado en la documentación. En la figura 4.4 se puede observar la configuración de la conexión HTTP, mientras que en las figuras 4.5 y 4.6 se aprecian la creación del JSON y el

parseo de la respuesta, respectivamente.

```
private void fetchTranscription(TranscribeAlert.OnTranscriptionSuccess onSuccess, TranscribeAlert.OnTranscriptionFail onFail) {
    new Thread() {
        @Override
        public void run() {
            String uri = "";
            HttpURLConnection connection = null;
            long start = SystemClock.elapsedRealtime();
            try {
                uri = "https://speech.googleapis.com/v1/speech:recognize?key=";
                uri += Uri.encode(API_KEY);
                uri += "&access_token=";
                uri += Uri.encode(getOAuthToken());
                connection = (HttpURLConnection) new URI(uri).toURL().openConnection();
                connection.setRequestMethod("POST");
                connection.setDoInput(true);
                connection.setDoOutput(true);
                connection.setRequestProperty("Accept", "application/json");
                connection.setRequestProperty("Content-Type", "application/json");
```

Figura 4.4: Conexión HTTP a la API

```
FileInputStream fileInputStreamReader = new FileInputStream(file);
byte[] bytes = new byte[(int) file.length()];
fileInputStreamReader.read(bytes);

JSONObject body = new JSONObject();
JSONObject config = new JSONObject();
JSONObject audio = new JSONObject();
config.put( name: "languageCode", LocaleController.getLocaleStringIso639());
config.put( name: "encoding",  value: "OGG_OPUS");
config.put( name: "sampleRateHertz",  value: 16000);
config.put( name: "model",  value: "default");
audio.put( name: "content", Base64.encodeToString(bytes, Base64.NO_WRAP));
body.put( name: "config", config);
body.put( name: "audio", audio);

String jsonInputString = body.toString();
try (OutputStream os = connection.getOutputStream()) {
    byte[] input = jsonInputString.getBytes(Charset.forName("UTF-8"));
    os.write(input,  off: 0, input.length);
}
```

Figura 4.5: Creación del JSON de la petición

```
StringBuilder textBuilder = new StringBuilder();
try (Reader reader = new BufferedReader(new InputStreamReader(connection.getInputStream(), Charset.forName("UTF-8")))) {
    int c = 0;
    while ((c = reader.read()) != -1) {
        textBuilder.append((char) c);
    }
}
String jsonString = textBuilder.toString();

JSONTokener tokener = new JSONTokener(jsonString);
JSONArray results = (JSONArray) new JSONObject(tokener).get("results");
JSONArray alternatives = (JSONArray) results.getJSONObject( index: 0).get("alternatives");
final String result = (String) alternatives.getJSONObject( index: 0).get("transcript");
```

Figura 4.6: Procesamiento de la respuesta

A mayores, también se añadió un método para generar el JWT necesario para poder realizar la llamada a la API. Este método, que se desarrolla en las figuras 4.7, 4.8, y 4.9, recoge los datos y la clave privada creados en la consola de administración de Google, y genera el token de acuerdo a las especificaciones de la documentación del servicio [29]. De manera concisa, este método crea un objeto de clave privada y un objeto JSON a partir del fichero descargado de la consola, codifica las distintas partes del token en Base64, y utiliza la clave para firmar el contenido y terminar de generar el token. Al final, lo convierte en una cadena de caracteres para poder incluirlo en la petición.

```
private String getOAuthToken() throws JSONException, IOException, NoSuchAlgorithmException, InvalidKeySpecException,
        InvalidKeyException, SignatureException {
    InputStream resourceStream = getContext().getResources().openRawResource(R.raw.gcloud_speech_to_text_credential);
    StringBuilder builder = new StringBuilder();

    try (BufferedReader reader = new BufferedReader(new InputStreamReader(resourceStream))) {
        String line = reader.readLine();
        while (line != null) {
            builder.append(line);
            line = reader.readLine();
        }
    }

    JSONTokener tokener = new JSONTokener(builder.toString());
    JSONObject jsonObject = new JSONObject(tokener);
    String kid = jsonObject.getString( name: "private_key_id");
    String email = jsonObject.getString( name: "client_email");
    String privateKeyEncoded = jsonObject.getString( name: "private_key");
    privateKeyEncoded = privateKeyEncoded.replace( target: "-----BEGIN PRIVATE KEY-----", replacement: "");
    privateKeyEncoded = privateKeyEncoded.replace( target: "-----END PRIVATE KEY-----", replacement: "");
    privateKeyEncoded = privateKeyEncoded.replaceAll( regex: "\\s+", replacement: "");
    long iat = Instant.now().getEpochSecond();
    long exp = iat + 3600;
```

Figura 4.7: Lectura y parseo de los datos

```
JSONObject header = new JSONObject();
header.put( name: "alg", value: "RS256");
header.put( name: "typ", value: "JWT");
header.put( name: "kid", kid);

JSONObject payload = new JSONObject();
payload.put( name: "iss", email);
payload.put( name: "sub", email);
payload.put( name: "aud", value: "https://speech.googleapis.com/");
payload.put( name: "iat", iat);
payload.put( name: "exp", exp);

byte[] headerEncoded = Base64.encode(header.toString().getBytes( charsetName: "UTF-8"), Base64.NO_WRAP);
byte[] payloadEncoded = Base64.encode(payload.toString().getBytes( charsetName: "UTF-8"), Base64.NO_WRAP);
byte[] jwt = new byte[headerEncoded.length + payloadEncoded.length + ".".getBytes( charsetName: "UTF-8").length];
ByteBuffer buffer = ByteBuffer.wrap(jwt);
buffer.put(headerEncoded);
buffer.put(".".getBytes( charsetName: "UTF-8"));
buffer.put(payloadEncoded);

byte[] privateKeyDecoded = Base64.decode(privateKeyEncoded, Base64.NO_WRAP);
PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKeyDecoded);
KeyFactory kf = KeyFactory.getInstance("RSA");
PrivateKey privateKey = kf.generatePrivate(keySpec);

Signature signature = Signature.getInstance("SHA256withRSA");
signature.initSign(privateKey);
signature.update(buffer.array());
byte[] signatureBytes = signature.sign();
```

Figura 4.8: Generación del payload y cabecera del JWT

```
StringBuilder builder1 = new StringBuilder();
builder1.append(Base64.encodeToString(headerEncoded, Base64.NO_WRAP));
builder1.append(".");
builder1.append(Base64.encodeToString(payloadEncoded, Base64.NO_WRAP));
builder1.append(".");
builder1.append(Base64.encodeToString(signatureBytes, Base64.NO_WRAP));

return builder1.toString();
```

Figura 4.9: Generación y firma del JWT

4.4 Herramientas

Las siguientes herramientas, aunque no forman parte de la solución diseñada, resultaron determinantes durante el proceso de desarrollo.

4.4.1 Comando "file"

El comando *file*, incluido en la mayoría de distribuciones Linux, se usa para determinar el tipo de un fichero. En este proyecto, se usó para obtener la información de los audios de Telegram. En la figura 4.10 se muestra el resultado de este comando, aplicado sobre uno de los audios de prueba.

```
~ ❯ file file_20.oga
file_20.oga: Ogg data, Opus audio, version 0.1, mono, 16000 Hz (Input Sample Rate)
```

Figura 4.10: Resultado del comando file

Esta información se usó para averiguar los datos a cubrir en las peticiones HTTP a Google Cloud, así como para comprobar que todos los audios tienen los mismos atributos, independientemente del dispositivo de origen.

4.4.2 Python

Python es un lenguaje de programación interpretado de alto nivel, desarrollado en 1991, y que es usado para aplicaciones y scripts de todo tipo. En el caso de este proyecto, su función fue realizar llamadas exploratorias a las APIs de Telegram y Google. Del lado de Telegram, se construyó un *bot* que descarga los mensajes de audio que recibe, y los almacena en el servidor en el que se ejecuta. Con esto se consiguieron los ficheros sobre los que ejecutar el comando *file* mencionado anteriormente.

Estos archivos también sirvieron como entrada para un script dedicado a probar la transcripción en la nube. Dicho script carga los datos de autenticación de la API y genera un JWT mediante una de las librerías recomendadas por Google, con el que realiza la conexión HTTP y obtiene el resultado. Con esta herramienta fue posible determinar la causa de varios errores en la propia aplicación de Telegram.

Por último, también se creó un script que compara los JWT generados tanto manualmente como mediante la librería, con el fin de refinar el proceso manual y hacer que ambos sean perfectamente equivalentes.

Capítulo 5

Validación y pruebas

El código fuente de Telegram no incluye ningún tipo de prueba unitaria o de integración. El motivo puede ser que la compilación se lleva a cabo fuera de Android Studio, que es el entorno que proporciona las herramientas de *debug* compatibles con aplicaciones de Android. Esto quiere decir que no es posible hacer uso de *breakpoints* u otras funciones habituales en esta fase del desarrollo.

A continuación se describen los escenarios de prueba que se tuvieron en cuenta.

5.1 *Happy path*: no existe ningún problema

Esta es la situación ideal, en la que todo funciona sin impedimentos y se puede llevar a cabo la operación sin problemas. Aquí, tras seleccionar la funcionalidad, se abre la alerta de transcripción con un texto de carga parpadeante. Cuando se recibe la respuesta del servicio en la nube, se cambia el contenido de la alerta y el texto deja de parpadear. Hay un ejemplo de ambos pasos en las figuras 5.1 y 3.7b.

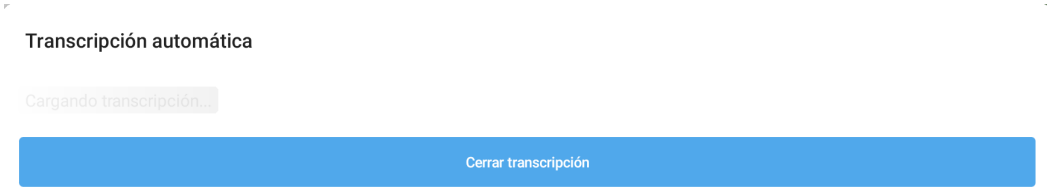


Figura 5.1: Alerta cargando el resultado

5.2 Audios demasiado largos

Como se mencionó previamente, actualmente existe una limitación de 60 segundos en la duración del audio a procesar por parte del servicio de transcripción de Google Cloud. En caso

de que se sobrepase este límite, la propia alerta muestra un mensaje de error, el de la figura 5.2, indicando que no se pueden procesar mensajes de audio tan largos.

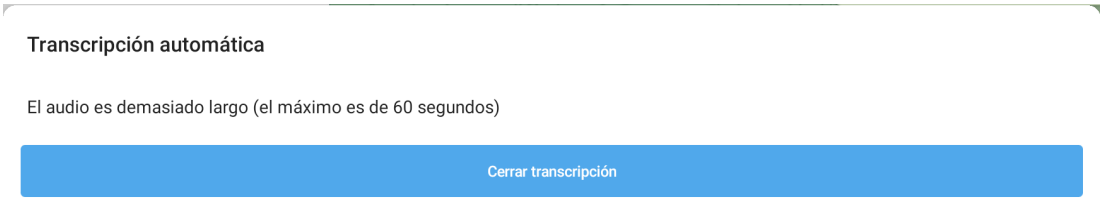


Figura 5.2: Advertencia de audio muy largo

Una alternativa a esta implementación sería que la opción apareciese sombreada en el menú, pero podría ser confuso para el usuario (al no explicitar el motivo de la no disponibilidad de la funcionalidad), por lo que consideramos que es preferible indicarlo expresamente en la alerta.

5.3 Errores de conexión y otros problemas

Este caso es similar al comportamiento de la funcionalidad tomada como modelo, la alerta de traducción. La implementación de ambas alertas se basa en un bloque *try catch* de Java, donde se capturan todas las excepciones indiscriminadamente, y se lanza una notificación *toast* que indica que ha ocurrido un error. El único error evidente que puede ocurrir y que no se ha tenido en cuenta aún es un fallo en la conexión a Google Cloud, bien por ausencia de acceso a Internet, bien porque el servicio está caído. Tanto en estos casos como en otros no previstos, no es un problema que a priori se pueda solucionar del lado de Telegram, por lo que el modus operandi es mostrar dicha notificación y permitir que el usuario lo intente de nuevo si así lo desea. En la figura 5.3 se muestra la *toast* [30], y en las figuras 5.4 y 5.5 aparece el código encargado de lanzarla.

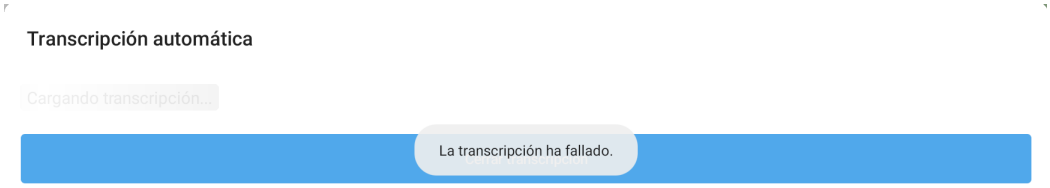


Figura 5.3: Notificación *toast* de error

```
} catch (Exception e) {
    try {
        Log.e( tag: "transcribe",  msg: "failed to transcribe an audio " +
                (connection != null ? connection.getResponseCode() : null) + " " +
                (connection != null ? connection.getResponseMessage() : null));
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
    e.printStackTrace();

    if (onFail != null && !dismissed) {
        try {
            final boolean rateLimit = connection != null && connection.getResponseCode() == 429;
            AndroidUtilities.runOnUIThread(() -> {
                onFail.run(rateLimit);
            });
        } catch (Exception e2) {
            AndroidUtilities.runOnUIThread(() -> {
                onFail.run( rateLimit: false);
            });
        }
    }
}
```

Figura 5.4: Bloque catch para todas las excepciones

```
fetchTranscription(
        (String transcribedText) -> {...},
        (boolean rateLimit) -> {
            if (rateLimit)
                Toast.makeText(getContext(), "Transcription failed. Try again later.", Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(getContext(), "Transcription failed.", Toast.LENGTH_SHORT).show();
        }
);
```

Figura 5.5: Creación de toast

Capítulo 6

Planificación y seguimiento

En este capítulo se describe la planificación del trabajo, esto es, las etapas (*sprints*) que fueron necesarias para su desarrollo, así como la evaluación del esfuerzo y coste, junto con el análisis de las desviaciones registradas.

6.1 Desarrollo temporal del trabajo

A continuación detallamos la temporalidad registrada en el avance del proyecto, desde su arranque hasta su cierre:

1. **Sprint 1** (10/12/2021 - 10/01/2022): 57 horas.
   En este primer sprint, el objetivo era asentar las bases del proyecto: encontrar y familiarizarse con el código fuente, instalar y configurar todas las herramientas necesarias, aprender a compilar la aplicación, e investigar y seleccionar formas de llevar a cabo la transcripción. Se decide usar el servicio que ofrece Android de forma nativa.

2. **Sprint 2** (10/01/2022 - 26/01/2022): 25 horas.
   Se revisó el código de creación del menú de opciones, así como el flujo de comprobación de tipo de mensaje e inserción de acciones. Se introduce una acción que simplemente copiaba el contenido del mensaje, pero mostrando un texto e icono distintos a la opción de copiar original, a modo de prueba de concepto.

3. **Sprint 3** (26/01/2022 - 14/02/2022): 42 horas.
   Se creó la alerta para mostrar un texto de ejemplo a partir del código de la alerta de traducción ya presente. Se vinculó el código de la opción añadida previamente a la creación y llamada de la nueva alerta, pero como resultado desapareció del menú dicha opción.

4. **Sprint 4** (14/02/2022 - 07/03/2022): 49 horas.
   Se volvió a escribir tanto el código de la alerta como el de la opción, pero el problema

seguía presente. Al final se pudo solucionar: se había introducido accidentalmente una línea que escondía la opción. Ahora que aparecía el texto de ejemplo, se cambió la llamada para usar el contenido del mensaje como entrada. Sin embargo, surgió un nuevo *bug*, y es que el texto aparecía cargando permanentemente.

5. **Sprint 5** (07/03/2022 - 21/03/2022): 31 horas.
   En este sprint se consiguió solucionar el *bug* del texto que aparecía cargando continuamente. También se investigaron las limitaciones de la Application Programming Interface (API) nativa de transcripción de Android, así como formas de paliarlas y poder hacer uso ilimitado de esta.

6. **Sprint 6** (21/03/2022 - 04/04/2022): 36 horas.
   Se implementó la solución para la llamada a la API de transcripción con un archivo de audio como entrada, pero no funcionó. Se llevaron a cabo varias ediciones y ajustes; en vano, porque el resultado fue igual, y es que existe otra limitación comentada muy ligeramente y en letra pequeña: solo se pueden usar ficheros como entrada a partir de la versión 12 de Android. Se buscó algún tipo de clase alternativa, pero no dio fruto.

7. **Sprint 7** (04/04/2022 - 19/04/2022): 38 horas.
   Más de la mitad de este sprint se empleó en tratar de crear un *pipe*, como los de los intérpretes de comandos, para redireccionar la entrada del micrófono al fichero de audio. También si intentó emular un micrófono virtual, pero ninguna de estas tácticas dio resultado. El resto del sprint se usó en revisar los servicios de transcripción en la nube y escoger uno, Google Cloud [31].

8. **Sprint 8** (19/04/2022 - 02/05/2022): 22 horas.
   En este sprint se aprendieron las limitaciones de Google Cloud, y se buscó en el código fuente una forma de obtener la duración y parámetros de los audios, pero solo se encontró la primera.

9. **Sprint 9** (02/05/2022 - 18/05/2022): 41 horas.
   Al ser inviable obtener los datos necesarios para las peticiones de transcripción, se creó un *bot* de Telegram que descarga los audios que recibe y los guarda con el nombre original de los servidores de Telegram. Con esto, se comprobó que el dispositivo de origen es indiferente al formato del audio, ya que todos los parámetros son iguales a la hora de almacenarlo. De esta manera, se obtuvo la codificación, el número de canales, y la frecuencia de muestreo, imprescindibles en la llamada a la API, de la cual se solicitó la clave. También se implementó la petición HTTP y el procesado de la respuesta, pero no funcionó.

10. **Sprint 10** (18/05/2022 - 01/06/2022): 19 horas.
    Se encontró el origen del problema en el uso de la API: no era suficiente con la clave de servicio, si no que también era necesario usar un JSON Web Token (JWT) para autenticar las peticiones.

11. **Sprint 11** (01/06/2022 - 08/06/2022): 26 horas.
    Tras revisar varias guías y descripciones del estándar, se desarrolló un programa en *Python* que crea un JWT de forma manual, y otro mediante una librería recomendada por Google. Con esto, se pudo afinar el proceso de creación del token, y llevarlo al código de Telegram. Se escribió el código para leer el fichero con las claves y generar el token, e insertarlo en las cabeceras HTTP de la petición. En circunstancias normales, este token se usa para soliticar otro proporcionado por el servidor de autenticación de Google, pero para ciertos servicios se puede ignorar este segundo paso y usar el original. Aún así, no se obtuvieron resultados.

12. **Sprint 12** (08/06/2022 - 21/06/2022): 24 horas.
    Con el fin de pulir las peticiones de manera más ágil, se creó otro programa en *Python* que crea un token válido, lee el contenido de un audio de prueba, y realiza la llamada HTTP. Funcionó, pero la respuesta del servidor era un mensaje en blanco, por lo que se introdujo código de *debug* en la aplicación, aunque sin éxito en la localización del problema.

13. **Sprint 13** (21/06/2022 - 28/06/2022): 10 horas.
    Tras revisar tanto el código como las peticiones en *Javascript* de la página que ofrece Google para probar sus servicios, se descubrió que el valor introducido como frecuencia de muestreo era incorrecto. Tanto la ventana de propiedades de Windows como las pruebas de Google apuntaban a un valor de 48 kHz, pero en realidad eran 16 kHz. Corregido esto, el problema pasó a ser otro, pero se pudo descartar la generación del JWT y la creación de la petición como origen del error.

14. **Sprint 14** (28/06/2022 - 12/07/2022): 19 horas.
    En este sprint se empezó a redactar la memoria del trabajo de acuerdo a las indicaciones de la tutora.

15. **Sprint 15** (12/07/2022 - 19/07/2022): 9 horas.
    Tras un breve periodo dedicado exclusivamente a la memoria del proyecto, se prosiguió con el desarrollo de la transcripción. Desafortunadamente, la aplicación de Docker dejó de funcionar, por lo que fue imposible compilar la aplicación hasta que se reparó.

16. **Sprint 16** (19/07/2022 - 27/07/2022): 22 horas.
    Tras varias compilaciones para ubicar el origen del problema, se corrigió una línea en

la creación del JWT que usaba un conjunto de caracteres incorrecto. Solucionado esto, el error pasó a ser una excepción relacionada con un fichero no encontrado y la URL.

17. **Sprint 17** (27/07/2022 - 03/08/2022): 8 horas.
    En este sprint, se estudiaron los flujos de entrada y salida de la conexión HTTP, pero no se encontró una solución al problema previo.

18. **Sprint 18** (03/08/2022 - 10/08/2022): 17 horas.
    En el último sprint del desarrollo, se movió el token a la URL como parámetro en vez de como parte de las cabeceras HTTP. A mayores, se incrementaron los permisos de las claves usadas, y con esto el código pasó a funcionar sin problemas. Finalmente, se eliminó todo el código de *debug*, se estructuró el código de manera más ordenada, y se establecieron textos e iconos más apropiados a la funcionalidad.

19. **Sprint 19** (10/08/2022 - 11/11/2022): 106 horas.
    Con la parte de programación terminada, este sprint se dedicó a continuar y terminar la memoria. También se repasó por última vez el código de la aplicación para confirmar que funciona correctamente.

El esfuerzo total en horas-persona resultado de la agregación de todos los sprints ha ascendido a 601. La figura 6.1 es un diagrama de Gantt que representa el proyecto. Este diagrama no es una traducción exacta de la lista de sprints, ya que ha habido algunos que se han mezclado o partido en dos para mantener la homogeneidad en el contenido de cada uno. La equivalencia entre cada tarea del diagrama y los sprints se indica en la siguiente lista.

1. *Investigación y configuración del entorno de trabajo* equivale al primer sprint.
2. *Inserción de opción de prueba* es el segundo sprint.
3. *Creación de alerta con texto de mensaje* está compuesto por los sprints 3 y 4.
4. *Implementación de la solución nativa* comprende de los sprints 5, 6, y parte del 7.
5. La segunda parte del sprint 7 se convierte en la tarea *Revisión de servicios web*.
6. Los sprints 8 y 9 pasan a ser la tarea *Obtención de parámetros de audio e implementación de petición HTTP*.
7. Los sprints 10 y 11 equivalen a *Investigación e implementación del JWT*.
8. Los sprints 12 y 13 son la tarea *Corrección de los parámetros de audio establecidos*.
9. El sprint 14 y el inicio del 15 pasan a ser *Comienzo de la redacción de la memoria*.

10. El resto del sprint 15 equivale a *Reparación de entorno de trabajo*.
11. *Corrección de la generación del JWT* corresponde al sprint 16.
12. Los sprints 17 y 18 comparten la tarea *Corrección del uso del JWT y limpieza del código*.
13. El último sprint, el 19, está representado por la tarea *Continuación y finalización de la memoria*.

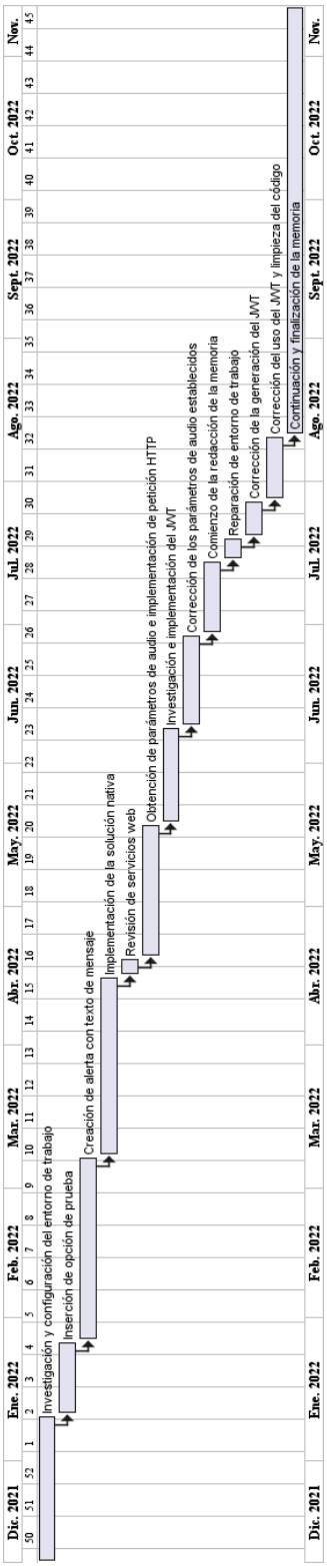


Figura 6.1: Diagrama de Gantt del proyecto

6.2 Desviaciones

Como se indicó anteriormente, hubo que cambiar el método de transcripción durante el desarrollo del proyecto. A continuación se hace una simulación de qué podría haber ocurrido si la transcripción local hubiese funcionado, o si se hubiese escogido la nube desde el principio.

6.2.1 Transcripción local exitosa

En este caso consideraremos que la API de transcripción nativa acepta ficheros sin depender de la versión del sistema. Todos los *sprints* a partir del sexto se verían afectados, este incluido. El proyecto pasaría directamente al estado del decimoctavo *sprint*, en el que se limpia y organiza el código. El número de horas empleadas se reduciría a menos de 385, casi la mitad de la duración real.

También podría plantearse el caso hipotético de que no haya límite de duración en el servicio local; de ser así, no habría sido necesario buscar soluciones en el quinto sprint, por lo que se habría terminado la implementación al poco de comenzar el sexto, reduciendo de nuevo la duración en cerca de 20 horas, por un total de 365.

6.2.2 Transcripción en la nube desde el principio

En este planteamiento, los *sprints* afectados son la segunda parte del quinto, el sexto, y la primera parte del séptimo. Por simplicidad, consideraremos que las partes no afectadas del quinto y séptimo se producen en un solo *sprint*, de una duración estimada en 30 horas. El resto sería exactamente igual, por lo que el número final de horas sería 526, 75 menos que la cifra real.

Este resultado es comprensible y totalmente plausible, teniendo en cuenta que la solución local falló relativamente pronto en el desarrollo del proyecto. La diferencia en el número final de horas entre estos supuestos muestra cómo de relevante fue el cambio de plataforma sobre la que desarrollar este trabajo.

6.3 Costes

Como comentamos al final de la sección 6.1, el total de las horas empleadas en el desarrollo de este proyecto fue de 601, que corresponden exclusivamente al trabajo del alumno, y no tienen en cuenta la dedicación de la tutora, que sería de aproximadamente una hora por sprint; resultando en un total de 620 horas entre ambos.

El coste monetario del proyecto se puede desglosar en salario, precio del equipo, precio del software, y coste de uso de los servicios mencionados. El software usado es completamente gratuito, y el servicio de transcripción en la nube, si bien es de pago, ofrece una hora

gratuita cada mes, que es suficiente para las pruebas de funcionamiento y el *debug*. El equipo hardware utilizado es relevante para aspectos como la velocidad de compilación, por lo que debe incluirse en el coste total del proyecto. El coste del equipo es de 1.000 euros. Por otro lado, el salario medio de un ingeniero de software en España es de 17'44 euros [32], por lo que el precio del total de horas resulta en 10.481'44 euros. La suma final es, por tanto, 11.481'44 euros; u 11.812'8 euros si se tienen en cuenta las horas invertidas por la tutora.

También podrían calcularse los costes de las desviaciones presentadas en la sección anterior. En el caso de la transcripción local exitosa, las horas totales estimadas son 385 y 365, siguiendo el orden en el que aparecen las variantes. A ambos totales se les pueden sumar 8 horas invertidas por la tutora, equivalentes al nuevo número de sprints. Esto resulta en un coste de 7.853'92 euros para la primera alternativa, y 7.505'12 euros para la segunda.

En a la segunda desviación, donde se escoge la transcripción en la nube desde el principio, el número de horas total es 526. El número de sprints se reduce en 2, por lo que las horas empleadas por parte de la tutora pasa a ser 17; esto se traduce en un coste total de 10.469'92 euros.

Capítulo 7
Conclusiones

En este último capítulo, se comparará el estado final del proyecto con los objetivos iniciales, y se presentarán las líneas de trabajo que se podrían acometer como pasos futuros, así como las lecciones aprendidas.

7.1 Objetivos alcanzados

El objetivo principal de este proyecto era la implementación de la funcionalidad de transcribir mensajes de audio, teniendo en cuenta los estándares de rendimiento y estética originales de Telegram. El proyecto, en su estado actual, cumple este objetivo central de manera aceptable, si bien las transcripciones están limitadas a audios de un minuto como máximo. El código de este proyecto se puede encontrar en su repositorio de GitHub [1].

7.2 Trabajos futuros

La mejora más evidente del proyecto sería incrementar la duración máxima de manera indefinida, o al menos al límite establecido por el proveedor del servicio. Esta mejora ya se ha estudiado para comprobar que es posible. La solución pasa por almacenar el fichero de audio en un *bucket* [33] de Google Cloud, y realizar la llamada a la API con el identificador en el campo de entrada de la petición. Como esta implementación requiere guardar el archivo en los servidores de Google, también es imprescindible crear un mensaje que alerte al usuario de esta circunstancia. A mayores, sería ideal incluir en los ajustes de la aplicación una opción para habilitar y deshabilitar la funcionalidad, tal y como ocurre con la traducción, como se puede ver en la figura 7.1.

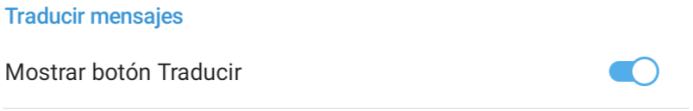


Figura 7.1: *Slider* para elegir la disponibilidad de la acción de traducir

Una última mejora sería almacenar el resultado de la transcripción junto al audio, para que repetidas llamadas a la función resulten en una única solicitud a la API, la de la primera vez. De esta forma, se podrían también consultar las transcripciones sin conexión, y se ahorraría en el coste del servicio en la nube. Esta implementación conllevaría un mayor consumo de espacio en los servidores de Telegram, pero al ser solo texto no debería ser un cambio notable.

Por otro lado, sería necesario introducir esta funcionalidad en el resto de clientes de Telegram, ya que por norma general intentan que todas las plataformas a las que dan soporte disfruten de los mismos servicios. Este proyecto resulta útil para este fin, ya que las secciones críticas (la conexión HTTP y la generación del JWT) se pueden extrapolar fácilmente a otros lenguajes. También se pueden aprovechar otros aspectos, como los valores necesarios para cada campo de las solicitudes REST, o la dirección URL a la que realizarlas; por no hablar de la configuración de cada conversión a Base64, que de no ser precisa haría que el servicio dejase de funcionar completamente.

También existe la posibilidad de implementar de nuevo, en el futuro, la solución nativa mencionada en otros capítulos. A medida que avance el tiempo, se extenderá el uso de Android 12 a más dispositivos; aunque aquí habría que considerar si los desarrolladores de Telegram estarían dispuestos a dejar de dar soporte a versiones antiguas de Android. También hay que tener en cuenta el problema de la fragmentación en Android [34], que existe desde hace más de una década [35] y no parece que se vaya a solucionar pronto. Por último, hay que tener en cuenta lo mencionado en el anterior párrafo, y es que la solución actual es independiente de la plataforma sobre la que se ejecuta, por lo que la implementación será similar en todas las variantes de Telegram.

7.2.1 Telegram Premium

El 21 de junio de 2022, Telegram lanzó un nuevo servicio de pago. Por 5,49 euros al mes, los usuarios de Telegram Premium tienen ventajas como mayores tamaños de subida de archivos, mayor velocidad de descarga, o, el aspecto más relevante a este proyecto, la posibilidad de convertir audios de voz a texto. Esta funcionalidad se utiliza presionando un botón que aparece junto al espectro de audio del mensaje. Al pulsar el botón, aparece el texto transcrito. La figura 7.2 es una captura sacada de la propia noticia publicada por Telegram en su página web [36].

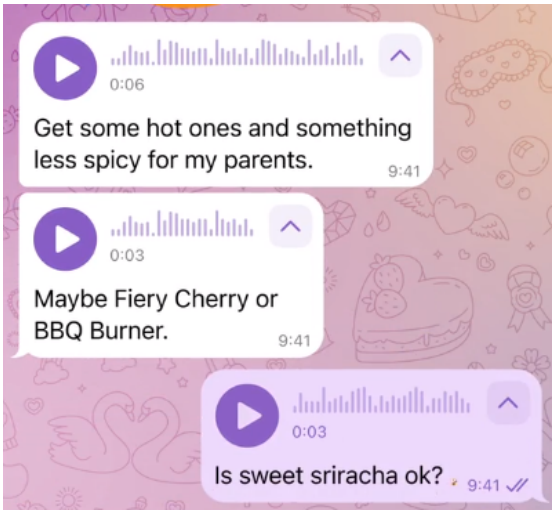


Figura 7.2: Ejemplo de transcripción en Telegram Premium

Resulta evidente, pues, que hacer una *pull request* de este proyecto para integrarlo con el repositorio oficial carece de utilidad, ya que está cubriendo una funcionalidad que no solo está implementada, sino que además es de pago (aunque el código está disponible desde el *commit* del 21 de junio de 2022 [37]).

Aún así, se podría argumentar que este trabajo sigue teniendo utilidad. Si bien no parece que vaya a integrarse con la aplicación oficial, Telegram anima a todos los desarrolladores que así lo deseen a publicar aplicaciones que puedan reemplazar a la oficial, por lo que siempre existe esa alternativa.

Por otro lado, al ser una aplicación de código abierto, nada impide a un usuario utilizar este proyecto en su día a día para hacer uso del servicio de transcripción. Es tan sencillo como crear una cuenta gratuita de Google Cloud, obtener las claves necesarias, e introducirlas en los lugares apropiados. Si bien esta clase de cambios no son tan sencillos para personas sin conocimientos de programación, tampoco son tan complejos como para que resulte imposible. Para que sea todavía más fácil, incluso sería posible automatizar dichos cambios mediante un programa adicional, o implementando la entrada de datos dentro de la propia *app*.

En última instancia, no existe el trabajo inútil, y menos en el mundo del código abierto. Este trabajo puede servir para el desarrollo de otros proyectos relacionados con Telegram, y por tanto tiene un valor inherente que se debe tener en cuenta; por no hablar de lo mucho que contribuyó a mi formación en campos como el desarrollo de aplicaciones en Android, el consumo de servicios en la nube, y la participación en proyectos *open source*.

7.3 Lecciones aprendidas

Tras experimentar con el código fuente en profundidad, hay dos aspectos relevantes que saltan a la vista, y que desafían lo que la mayoría de programadores consideran buenas prácticas.

En primer lugar, la falta más grave es la absoluta carencia de comentarios. En un proyecto de la escala de Telegram, donde existen numerosas clases anidadas dentro de ficheros que alcanzan las 20.000 líneas de código, la falta de comentarios se traduce en una dificultad descomunal a la hora de saber qué hace cada componente. A menudo ha sido necesario saltar entre tramos del código para comprender qué se intenta conseguir, para que luego resulte que los esquemas y deducciones hechos hasta el momento son erróneos. En general, los nombres de las clases y los ficheros son bastante descriptivos, pero tanto la cantidad como la complejidad de cada uno de ellos impide que esto sea suficiente para navegar el código sin problemas. No parece posible fraccionar el código en más clases, y probablemente hacer esto solo complicaría más las cosas, pero en numerosas ocasiones una simple explicación de qué se pretende conseguir con ciertos bucles y variables reduciría considerablemente el número de horas necesarias para familiarizarse con la estrucutura de la aplicación.

El segundo aspecto a mencionar es la falta de uso de propiedades inherentes al lenguaje, Java. En concreto, se observó que a la hora de añadir opciones en los menús de cada mensaje, se hace uso de *magic numbers*, es decir, se añaden números directamente a la lista de opciones; o sería más correcto decir que hay varios casos en los que es así, mientras que en otros se hace uso de constantes creadas como miembros de la clase en cuestión. En estas situaciones, la forma correcta de proceder es crear *enums* de Java, en los que se almacena cada opción con un nombre descriptivo. Este problema también sucede al identificar el tipo de cada mensaje, donde se puede observar que dentro de las sentencias condicionales hay números enteros sin ningún tipo de contexto o explicación; un ejemplo de esto está en la figura 7.3.

```
} else if (type == 4) {...} else if (type == 5) {...} else if (type == 10) {
    items.add(LocaleController.getString( key: "ApplyThemeFile", R.string.ApplyThemeFile));
    options.add(5);
    icons.add(R.drawable.msg_theme);
    if (!getMessagesController().isChatNoForwards(currentChat)) {...}
} else if (type == 6 && !getMessagesController().isChatNoForwards(currentChat)) {...} else if (type == 7) {
    if (selectedObject.isMask()) {...} else {...}
} else if (type == 8) {...} else if (type == 9) {...}
```

Figura 7.3: Ejemplo de comprobación del tipo de mensaje

Lista de acrónimos

API Application Programming Interface. 6–11, 19, 21, 23, 25, 30, 31, 35, 37, 38

JWT JSON Web Token. 6, 8, 9, 23, 25, 31, 32, 38

SDK Software Development Kit. 6, 20, 21

Bibliografía

[1] "Telegram: Telegram for Android source," consultado el 2022-11-13. [En línea]. Disponible en: https://github.com/AllInOneByte/Telegram/tree/master

[2] "BlackBerry Messenger," consultado el 2022-11-13. [En línea]. Disponible en: https://es.wikipedia.org/wiki/BlackBerry_Messenger

[3] "WhatsApp," consultado el 2022-11-13. [En línea]. Disponible en: https://www.whatsapp.com

[4] "LINE | always at your side." consultado el 2022-11-13. [En línea]. Disponible en: https://line.me

[5] "WeChat - Free messaging and calling app," consultado el 2022-11-13. [En línea]. Disponible en: https://www.wechat.com

[6] "Telegram Messenger," consultado el 2022-11-13. [En línea]. Disponible en: https://telegram.org

[7] OMS, "Sordera y pérdida de audición," 2021, consultado el 2022-11-13. [En línea]. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/deafness-and-hearing-loss

[8] A. Orús, "Transporte público: tasa de penetración 2017-2026," 2022, consultado el 2022-11-13. [En línea]. Disponible en: https://es.statista.com/estadisticas/1012265/tasa-de-penetracion-del-transporte-publico-en-el-mundo/

[9] "What is Scrum?" consultado el 2022-11-13. [En línea]. Disponible en: https://www.scrum.org/resources/what-is-scrum/

[10] D. Anderson, "Kanban - Successful Evolutionary Change for your Technology Business," 2010, consultado el 2022-11-13. [En línea]. Disponible en: https://archive.org/details/kanbansuccessful0000ande

[11] "Telegram: Telegram for Android source," consultado el 2022-11-13. [En línea]. Disponible en: https://github.com/DrKLO/Telegram

[12] "Docker: Accelerated, Containerized Application Development," 2013, consultado el 2022-11-13. [En línea]. Disponible en: https://www.docker.com

[13] "Alphatrad | Traducciones y servicios lingüísticos," consultado el 2022-11-13. [En línea]. Disponible en: https://www.alphatrad.es

[14] "Transcripciones de Audio a Texto | Transcripciones.net," consultado el 2022-11-13. [En línea]. Disponible en: https://transcripciones.net

[15] "Best Audio and Video transcription services | GoTranscript," consultado el 2022-11-13. [En línea]. Disponible en: https://gotranscript.com

[16] "Amberscript: Audio and Video Transcription," consultado el 2022-11-13. [En línea]. Disponible en: https://www.amberscript.com

[17] "VEED - Online Video Editor," consultado el 2022-11-13. [En línea]. Disponible en: https://www.veed.io

[18] "Happyscribe: Audio Transcription and Video Subtitles," consultado el 2022-11-13. [En línea]. Disponible en: https://www.happyscribe.com

[19] "IBM Cloud," 2013, consultado el 2022-11-13. [En línea]. Disponible en: https://www.ibm.com/cloud

[20] "Servicios de informática en la nube | Microsoft Azure," 2010, consultado el 2022-11-13. [En línea]. Disponible en: https://azure.microsoft.com

[21] Google, "Gboard: el teclado de Google," 2014, consultado el 2022-11-13. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin

[22] WellSource, "Speechnotes - Voz a texto," 2016, consultado el 2022-11-13. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=co.speechnotes.speechnotes

[23] Google, "Asistente de Google," 2018, consultado el 2022-11-13. [En línea]. Disponible en: https://play.google.com/store/apps/details?id=com.google.android.apps.googleassistant

[24] G. Simmons, "A survey of information authentication," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 603–620, 1988, consultado el 2022-11-13.

[25] Free Software Foundation, Inc., "GNU Library General Public License v2.0," 1991, consultado el 2022-11-13. [En línea]. Disponible en: https://www.gnu.org/licenses/old-licenses/lgpl-2.0.html

[26] "Gradle Buil Tool," 2008, consultado el 2022-11-13. [En línea]. Disponible en: https://gradle.org

[27] Telegram, "Reproducible Builds for iOS and Android," 2020, consultado el 2022-11-13. [En línea]. Disponible en: https://core.telegram.org/reproducible-builds#reproducible-builds-for-android

[28] S. Brown, "The C4 model for visualising software architecture," 2018, consultado el 2022-11-13. [En línea]. Disponible en: https://c4model.com

[29] "Using OAth 2.0 for Server to Server Applications," consultado el 2022-11-13. [En línea]. Disponible en: https://developers.google.com/identity/protocols/oauth2/service-account#httprest

[30] "Toasts overview," consultado el 2022-11-13. [En línea]. Disponible en: https://developer.android.com/guide/topics/ui/notifiers/toasts

[31] "Cloud Computing Services | Google Cloud," 2008, consultado el 2022-11-13. [En línea]. Disponible en: https://cloud.google.com

[32] "Salario para Ingeniero De Software en España," consultado el 2022-11-13. [En línea]. Disponible en: https://es.talent.com/salary?job=ingeniero+de+software

[33] "About Cloud Storage buckets," consultado el 2022-11-13. [En línea]. Disponible en: https://cloud.google.com/storage/docs/buckets

[34] J. García, "La fragmentación en Android sigue siendo un problema: apenas uno de cada diez móviles tienen Android 12," 2022, consultado el 2022-11-13. [En línea]. Disponible en: https://www.xataka.com/moviles/fragmentacion-android-sigue-siendo-problema-apenas-uno-cada-diez-moviles-tienen-android-12

[35] C. Rebato, "La fragmentación es el principal problema de Android, según los desarrolladores," 2011, consultado el 2022-11-13. [En línea]. Disponible en: https://hipertextual.com/2011/04/la-fragmentacion-es-el-principal-problema-de-android-segun-los-desarrolladores

[36] Telegram, "700 millones de usuarios y Telegram Premium," 2022, consultado el 2022-11-13. [En línea]. Disponible en: https://telegram.org/blog/700-million-and-premium

[37] “Update to 8.8.2,” consultado el 2022-11-13. [En línea]. Disponible en: https://github.com/DrKLO/Telegram/commit/96dce2c9aabc33b87db61d830aa087b6b03fe397