DESARROLLO DE UNA APLICACIÓN EDUCATIVA DESTINADA A ESTUDIANTES CON DISCAPACIDADES VISUALES

DEVELOPMENT OF AN EDUCATIONAL APPLICATION FOR STUDENTS WITH VISUAL IMPAIRMENTS

TRABAJO FIN DE MÁSTER
CURSO 2023-2024

AUTOR
HANJIE ZHU
DIRECTOR
ANTONIO SARASA CABEZUELO

CONVOCATORIA: SEPTIEMBRE 2024
CALIFICACIÓN: 6 (APROBADO)

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DESARROLLO DE UNA APLICACIÓN EDUCATIVA DESTINADA A ESTUDIANTES CON DISCAPACIDADES VISUALES

DEVELOPMENT OF AN EDUCATIONAL APPLICATION FOR STUDENTS WITH VISUAL IMPAIRMENTS

TRABAJO DE FIN DE MÁSTER EN INGENIERÍA INFORMÁTICA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

AUTOR
HANJIE ZHU

DIRECTOR
ANTONIO SARASA CABEZUELO

MÁSTER EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

21 DE SEPTIEMBRE DE 2024

“La presente memoria del Trabajo de Fin de Máster (TFM) tiene como propósito diseñar y desarrollar una aplicación de aprendizaje orientada tanto a estudiantes con discapacidades visuales como a estudiantes sin discapacidades, utilizando herramientas de inteligencia artificial y los modelos de Procesamiento del Lenguaje Natural (NLP) disponibles en el mercado.”

AGRADECIMIENTO

Quiero comenzar agradeciendo a mi tutor, Antonio Sarasa Cabezuelo, por su invaluable orientación y apoyo durante todo este proceso. Su conocimiento y disposición para ayudarme han sido cruciales para completar este Trabajo de Fin de Máster.

Asimismo, quiero agradecer a mis profesores y mentores a lo largo de estos años, cuyas enseñanzas y consejos han sido pilares fundamentales en mi formación académica y profesional. Vuestro impacto en mi vida ha sido inmenso.

Quiero expresar mi más profundo agradecimiento a mi familia. Su amor incondicional y su constante aliento han sido fundamentales para que yo pudiera alcanzar mis objetivos. Han estado a mi lado en cada paso del camino, brindándome el apoyo necesario para enfrentar cualquier desafío. Sin su respaldo, nada de esto hubiera sido posible. Gracias por ser mi pilar y por creer en mí siempre.

Además, me gustaría expresar mi gratitud a mis amigos y compañeros de la clase. Su apoyo constante y sus palabras de aliento han sido esenciales para mí, especialmente en los momentos más difíciles. Han demostrado una fe inquebrantable en mis capacidades, lo que me ha ayudado a superar muchas adversidades. Gracias por vuestra amistad y por estar siempre a mi lado.

Finalmente, quiero rendir homenaje a todas las experiencias y retos que me han formado a lo largo de este camino. Cada paso, cada caída y cada levantada han contribuido a mi crecimiento y éxito. Gracias a todos por ser parte de este viaje.

ABSTRACT

In this project, we have designed and implemented a learning application aimed at students with visual impairments. The application utilizes an AI-based chat system and various Natural Language Processing (NLP) models to support its functionality. The NLP models are categorized into three types: voice recognition and audio-to-text conversion, document recognition from visual documents, and text generation based on written context.

The primary functionality of the application is a chat generated by an AI model, which interacts with the documents provided as data. The objective is to enable users to ask specific questions about documents within a particular knowledge area, based on the provided documents. This setup facilitates note-taking, specific knowledge retrieval, and report writing for students with visual impairments.

The application enhances accessibility and learning for both visually impaired and sighted students by integrating NLP models that enable natural interaction with educational content and optimize the learning experience. Additional features include an audio reminder system, secure document storage, and the use of Firebase [38] for managing and storing important information.

The models utilized in the application are highly trained and support multiple languages, making the application accessible to students from various nationalities. NLP models are implemented through the Hugging Face community [29], while the AI-based chat model is implemented using resources from the Langchain community [45]. This integration not only reduces costs but also simplifies the implementation process, ensuring uninterrupted operation of the application.

KEYWORDS

Learning application, visually impaired students, AI-based chatbot, Natural Language Processing (NLP) models, voice recognition and audio-to-text conversion, visual document recognition, text generation from written context, optimized learning experience.

RESUMEN

En este proyecto se ha diseñado e implementado una aplicación de aprendizaje dirigida a estudiantes con discapacidades visuales, empleando un chat basado en inteligencia artificial y diversos modelos de Procesamiento del Lenguaje Natural (NLP) para el soporte de la aplicación. Los modelos NLP se clasifican en tres tipos: el modelo de reconocimiento de voz y conversión de audio a texto, el modelo de reconocimiento de documentos a partir de un documento visual, y el modelo de generación de texto a partir de un contexto escrito.

La funcionalidad principal radica en un chat generado mediante un modelo de inteligencia artificial y los documentos proporcionados como datos. El objetivo es que, partiendo de un área de conocimiento específica y aportando los documentos correspondientes, se genere un chat en el cual se puedan realizar preguntas específicas sobre los documentos de esa área delimitada. De esta manera, se facilita la toma de apuntes, la búsqueda de conocimientos específicos y la redacción de una memoria para estudiantes con discapacidades visuales.

Esta aplicación mejora la accesibilidad y el aprendizaje para estudiantes con y sin discapacidades visuales mediante la integración de modelos NLP, lo que facilita una interacción natural con el contenido educativo y optimiza la experiencia de aprendizaje. Además, incluye un sistema de recordatorios de audio, almacenamiento seguro de documentos y utiliza Firebase [38] para gestionar y guardar información importante.

Los modelos utilizados en la aplicación están altamente entrenados y ofrecen soporte en múltiples idiomas, facilitando el acceso a estudiantes de diversas nacionalidades. La implementación de los modelos de Procesamiento del Lenguaje Natural (NLP) se lleva a cabo a través de la comunidad Huggingface [29], mientras que el modelo de chat basado en inteligencia artificial se implementa utilizando los recursos de la comunidad Langchain [45]. Esta integración no solo reduce costos, sino que también simplifica la implementación, asegurando un funcionamiento ininterrumpido de la aplicación.

PALABRAS CLAVE

Aplicación de aprendizaje, estudiantes con discapacidades visuales, chatbot basado en inteligencia artificial, modelos de Procesamiento del Lenguaje Natural (NLP), reconocimiento de voz y conversión de audio a texto, reconocimiento de documentos visuales, generación de texto a partir de un contexto escrito, experiencia de aprendizaje optimizada.

ÍNDICE

CAPÍTULO 1: INTRODUCCIÓN

En la actualidad, existe una notable escasez de aplicaciones educativas verdaderamente útiles para estudiantes con discapacidades visuales o estudiantes sin discapacidades. Esta carencia refleja un estancamiento en el desarrollo de herramientas tecnológicas adaptadas a sus necesidades específicas, a pesar de los avances en otras áreas de la educación digital. Es crucial enfocar y avanzar en el desarrollo de tecnologías y aplicaciones que utilicen modelos avanzados para apoyar el estudio y aprendizaje de estos estudiantes, quienes enfrentan desafíos únicos en su acceso a la información educativa.

La creación de una tecnología innovadora que integre varios modelos avanzados podría representar un mercado significativo que aún no ha sido completamente explorado ni desarrollado. Al mejorar la accesibilidad y adaptabilidad de las aplicaciones educativas para personas con discapacidades visuales o estudiantes con otras dificultades, se contribuiría no solo a facilitar su aprendizaje, sino también a promover una sociedad más inclusiva y equitativa. Este enfoque no solo busca equiparar las oportunidades educativas, sino también enriquecer la experiencia educativa de todos los estudiantes, independientemente de sus capacidades.

Para alcanzar la verdadera igualdad en una sociedad avanzada, es esencial abordar de manera prioritaria los desafíos específicos que enfrentan las personas con discapacidades. Esto implica no solo desarrollar y lanzar nuevas aplicaciones educativas dirigidas exclusivamente a estudiantes sin discapacidades, sino también diseñar soluciones adaptativas y accesibles que satisfagan las necesidades únicas de los estudiantes con discapacidades visuales. Lamentablemente, la falta de atención y recursos dedicados a este problema puede perpetuar la discriminación y la exclusión en la sociedad actual.

Es imperativo que la sociedad tome en serio el desarrollo de tecnologías inclusivas que promuevan la igualdad de acceso y oportunidades para todos los individuos, sin importar sus capacidades físicas. Al hacerlo, no solo avanzamos hacia una sociedad más justa y equitativa, sino que también aprovechamos el potencial y las capacidades de todos los miembros de nuestra comunidad global.

El desarrollo de nuevas tecnologías orientadas a personas con discapacidades puede ser una tarea compleja y costosa, especialmente al intentar abordar todas las necesidades específicas de este grupo diverso. Dado que la mayoría de los desarrolladores no tienen discapacidades, comprender completamente estas necesidades puede ser un desafío considerable. Sin embargo, en el mercado existen varias herramientas que ya cubren las necesidades básicas para el estudio de personas con discapacidades, las cuales pueden servir como base sólida en el desarrollo de nuevas aplicaciones.

Una colaboración estrecha con estudiantes con discapacidades visuales en la creación y prueba de aplicaciones educativas puede proporcionar valiosas perspectivas y entendimiento sobre sus verdaderas necesidades en el ámbito educativo. Este enfoque incremental nos permite avanzar gradualmente hacia la creación de aplicaciones educativas más completas y adaptadas específicamente para estudiantes con discapacidades visuales.

En este contexto, surge una clara necesidad de innovar y desarrollar nuevas aplicaciones educativas que sean verdaderamente revolucionarias para estudiantes con discapacidades visuales. Estas aplicaciones no solo tienen el potencial de mejorar significativamente la experiencia educativa de estos estudiantes, sino también de optimizar sus tiempos y costos asociados con el estudio. Al proporcionar herramientas que faciliten el acceso a la información y apoyen el aprendizaje de manera

efectiva, podemos contribuir a promover la igualdad de oportunidades y el desarrollo académico de las personas con discapacidades visuales.

Es esencial que estas nuevas tecnologías sean diseñadas con un enfoque centrado en el usuario, asegurando que satisfagan de manera efectiva las necesidades específicas y diversas de los estudiantes con discapacidades visuales. Al hacerlo, no solo estamos avanzando en términos de inclusión y accesibilidad, sino también fortaleciendo nuestra capacidad colectiva para apoyar el aprendizaje y el desarrollo personal de todos los individuos, independientemente de sus capacidades físicas.

1.1. MOTIVACIÓN

La motivación subyacente de nuestro proyecto surge de la necesidad urgente de abordar los desafíos que enfrentan los estudiantes con discapacidades visuales, como se mencionó anteriormente. Nuestro objetivo principal es desarrollar un sistema que no solo sea sostenible y eficiente, sino también fácil de implementar, con el propósito de mejorar significativamente la experiencia educativa de estos estudiantes, haciendo que sea más accesible y equitativa. Esto implica reducir las barreras y los costos asociados con el estudio para este grupo específico, con el fin último de equiparar su acceso educativo con el de sus pares sin discapacidades visuales.

Nuestra visión para la aplicación incluye la implementación de un sistema de estudio altamente escalable, diseñado para adaptarse a diversas modalidades y áreas de estudio. Este enfoque nos permitirá ofrecer una solución accesible y eficaz para estudiantes con diferentes necesidades y estilos de aprendizaje. La escalabilidad es un componente crucial para alcanzar la eficiencia y la rentabilidad en la provisión de herramientas educativas, asegurando que nuestra solución pueda ser ampliamente adoptada no solo por estudiantes con discapacidades visuales, sino por una audiencia más amplia.

Al enfocarnos en la sostenibilidad y la adaptabilidad, aspiramos a superar las limitaciones actuales en el acceso a la educación para personas con discapacidades visuales. Nuestro compromiso radica en crear un entorno educativo más inclusivo y equitativo, donde todas las personas, independientemente de sus capacidades físicas, puedan alcanzar su máximo potencial académico.

1.2. OBJETIVOS

El objetivo primordial es diseñar y desarrollar una aplicación que integre varios componentes destinados a mejorar el proceso de estudio.

1. Se persigue el diseño de una aplicación que ofrezca una gestión de datos de usuario eficiente y una administración de cuenta similar a las aplicaciones disponibles en el mercado actual. Este enfoque asegurará que los usuarios puedan manejar sus datos de manera segura y realizar ajustes en su cuenta de manera intuitiva, garantizando así una experiencia de usuario fluida y confiable.
2. Se busca implementar una solución que facilite y optimice el uso de la aplicación para estudiantes con discapacidades visuales. Este enfoque está diseñado para mejorar significativamente la accesibilidad y la experiencia de usuario de este grupo específico de estudiantes.
3. El desarrollo incluirá una solución de la búsqueda simplificada de contenidos dentro de un documento a partir de una pregunta proporcionada por el estudiante. Esto permitirá una

interacción más eficiente y efectiva con los materiales educativos, mejorando la capacidad de los usuarios para obtener información relevante rápidamente.

4. Se pretende proporcionar una solución escalable y personalizable que pueda adaptarse a diversas áreas de estudio y satisfacer las necesidades específicas de cada usuario. Este enfoque garantizará que la aplicación sea versátil y capaz de responder a diferentes modalidades educativas y requerimientos individuales, promoviendo así una experiencia de aprendizaje más efectiva y personalizada.
5. La implementación de una solución que optimiza la redacción de memorias y documentos, permitiendo a los estudiantes ahorrar tiempo considerablemente en el proceso de escritura y facilitando así la elaboración de textos de manera más eficiente y efectiva.

1.3. Plan de Trabajo

Para la consecución de los objetivos planteados, se han ejecutado las siguientes tareas:

1. Desarrollo de una aplicación utilizando Ionic [39], que incorpora un sistema integral de gestión, almacenamiento y monitorización de datos basado en Firebase [38] de Google.
2. Implementación de un modelo avanzado de reconocimiento de audio, diseñado para ejecutar acciones específicas mediante comandos de voz, con el propósito de facilitar el uso de la aplicación a estudiantes con discapacidades visuales.
3. Integración de un sistema de reconocimiento de documentos, que proporciona la información necesaria de manera eficiente, mejorando así la accesibilidad y utilidad del contenido académico para los estudiantes.
4. Desarrollo de un chatbot adaptado al área de estudio de los estudiantes. Este chatbot, una vez entrenado con los documentos proporcionados por los propios estudiantes, es capaz de extraer y suministrar información específica contenida en dichos documentos, cuando los estudiantes requieran buscar datos concretos.
5. Incorporación de un modelo de generación de texto, diseñado para aumentar la eficiencia en la elaboración de memorias y otros documentos académicos, optimizando el tiempo y esfuerzo dedicados a estas tareas por parte de los estudiantes.

1.4. Organización de la Memoria

La presente memoria ha sido organizada con el fin de proporcionar una explicación exhaustiva y comprensible de los procedimientos y fundamentos subyacentes en el desarrollo del proyecto. La estructura de la memoria se articula en los siguientes capítulos, cada uno de los cuales aborda aspectos específicos del proyecto de manera detallada:

- **Capítulo 1: Introducción.** En este capítulo se presenta la motivación subyacente y los objetivos fundamentales que guían el desarrollo del presente proyecto. Además, se describe detalladamente el plan de trabajo adoptado y se proporciona una visión general de la estructura de la memoria que se ha elaborado.

- **Capítulo 2: Estado del arte.** En este capítulo efectúa un análisis de diversas herramientas y software que proporcionen funcionalidades análogas a las implementadas en el presente proyecto. una breve descripción de cada una de las herramientas identificadas durante el proceso de investigación.

- **Capítulo 3: Diseño Funcional.** En este capítulo se destina a describir el diseño inicial de concepción del proyecto, detallando los diseños iniciales.

- **Capítulo 4: Tecnologías utilizadas.** Este capítulo ofrece una descripción detallada de las tecnologías y herramientas empleadas en el desarrollo del proyecto.

- **Capítulo 5: Arquitectura y modelo de datos.** En este capítulo se detalla la estructura técnica del proyecto, describiendo la arquitectura general y los modelos de datos implementados.

- **Capítulo 6: Desarrollo del proyecto.** Se ofrece un resumen exhaustivo del desarrollo de las funcionalidades de la aplicación desarrollada.

- **Capítulo 7: Prueba.** Donde se describe la prueba de las funcionalidades desarrolladas durante el capítulo de desarrollo.

- **Capítulo 8: Discusión.** Donde se discute los resultados obtenidos, poner en valor los mismos, establecer debilidades y apuntar por donde se podría avanzar.

- **Capítulo 9: Conclusiones y trabajo futuro.** Se discuten los objetivos alcanzados y se evalúa el impacto potencial de la aplicación desarrollado en el contexto previsto. Además, se identifican áreas para futuras mejoras y desarrollo continuo, explorando las posibilidades y recomendaciones para expandir o perfeccionar el proyecto en el futuro.

- **Capítulos 10 y 11:** Traducción del capítulo 1 y el capítulo 9 a inglés, respectivamente.

- **Anexo I: Especificación de requisitos.** En este capítulo se exponen la especificación de requisitos necesarios para el desarrollo del proyecto.

- **Anexo II: Manual de usuario.** En este anexo se expone un manual de uso para los usuarios de la herramienta desarrollada.

Cada capítulo se estructura con el propósito de proporcionar una visión integral y sistemática del proyecto, desde su concepción hasta su implementación y evaluación, asegurando así una documentación completa y rigurosa del trabajo realizado.

CAPÍTULO 2: ESTADO DEL ARTE

En este segundo capítulo se presenta una revisión en diversas aplicaciones y herramientas que ofrecen funcionalidades similares a las que se han implementado en el presente proyecto. La evaluación de estas soluciones tecnológicas es fundamental para contextualizar y destacar las innovaciones y mejoras introducidas por nuestra aplicación.

2.1. GOOGLE SPEECH-TO-TEXT

En la sección de plan de trabajo del capítulo 1, podemos ver que la primera funcionalidad que vamos a desarrollar es parecida a un servicio avanzado de reconocimiento de voz desarrollado por Google, Google Speech-to-Text [1]. Este servicio utiliza algoritmos de aprendizaje automático y procesamiento del lenguaje natural (NLP) para convertir audio en texto con una alta precisión [1].

Durante un proyecto previo en una asignatura de máster, utilicé Google Speech-to-Text [1] para evaluar la tecnología avanzada disponible en el mercado. El servicio demostró ser altamente eficaz, procesando audio tanto en tiempo real como desde archivos pregrabados. Los modelos de reconocimiento de voz de Google están entrenados en extensos conjuntos de datos multilingües, ofreciendo soporte para más de 120 idiomas y dialectos [1].

Google Speech-to-Text [1] presenta una serie de características y ventajas, a continuación:

- **Soporte Multilingüe:** Ofrece más de 120 idiomas y dialectos.
- **Transcripción en Tiempo Real:** convierte el habla en texto simultáneamente a medida que se produce.
- **Personalización de Modelos:** Permite ajustar los modelos de reconocimiento de voz para incluir vocabularios específicos y mejorar la precisión en contextos especializados.
- **Filtrado de Contenido Explícito:** Funcionalidad para censurar palabras inapropiadas en la transcripción.
- **Detección de Hablantes:** Identificación de múltiples hablantes en un solo archivo de audio.
- **Escalabilidad:** Al estar basado en la infraestructura de Google Cloud, el servicio puede manejar grandes volúmenes de datos y solicitudes simultáneas.
- **Actualizaciones Constantes:** Google mejora continuamente sus modelos y servicios, asegurando que los usuarios tengan acceso a la tecnología más avanzada.

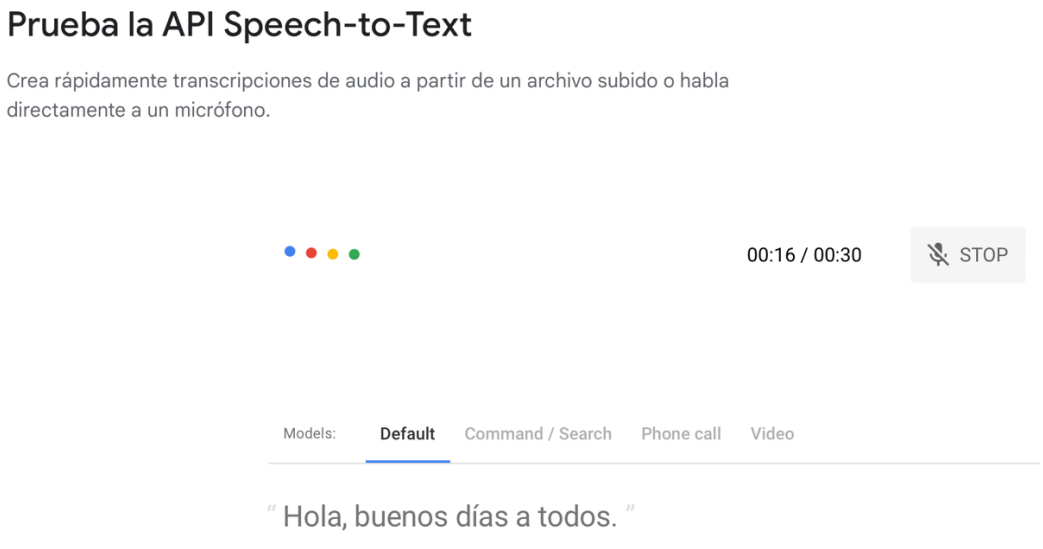


Figura 2.1.1: Google Speech-To-Text [1].

2.2. GOOGLE CLOUD DOCUMENT AI

Google Cloud Document AI [2] es una plataforma de procesamiento de documentos proporcionada por Google, diseñada para automatizar el análisis y la extracción de información a partir de documentos tanto estructurados como no estructurados [2]. Este servicio es capaz de manejar documentos en diversos formatos que sean visibles para la plataforma. Utiliza técnicas de OCR (Reconocimiento Óptico de Caracteres) y NLP (Procesamiento del Lenguaje Natural), tecnología de inteligencia artificial y aprendizaje automático para transformar documentos en datos estructurados [2].

Esta plataforma guarda una similitud con la segunda funcionalidad que deseamos implementar en nuestra aplicación. Al igual que Google Cloud Document AI [2], que es capaz de reconocer documentos no estructurados y sin formato, pero visualmente accesibles para la plataforma.

Características de Google Cloud Document AI [2]:

- **Reconocimiento Óptico de Caracteres (OCR):** Transforma imágenes de documentos escaneados y fotos en texto editable.
- **Extracción de Información:** Extrae automáticamente datos de formularios, facturas, contratos y otros documentos estructurados.
- **Análisis de Contenido:** Utiliza modelos de NLP para entender el contexto y organizar la información extraída de documentos no estructurados.
- **Análisis de Documentos:** Clasifica documentos en categorías predefinidas para facilitar su organización y búsqueda.
- **Etiquetado Inteligente:** Asocia etiquetas y metadatos relevantes a los datos extraídos, mejorando la accesibilidad y la búsqueda de información.
- **Alta Precisión:** Utiliza modelos de IA avanzados para ofrecer una alta precisión en la conversión y extracción de texto.
- **Soporte Multilingüe:** Admite múltiples idiomas, lo que facilita el procesamiento de documentos en diferentes lenguajes.

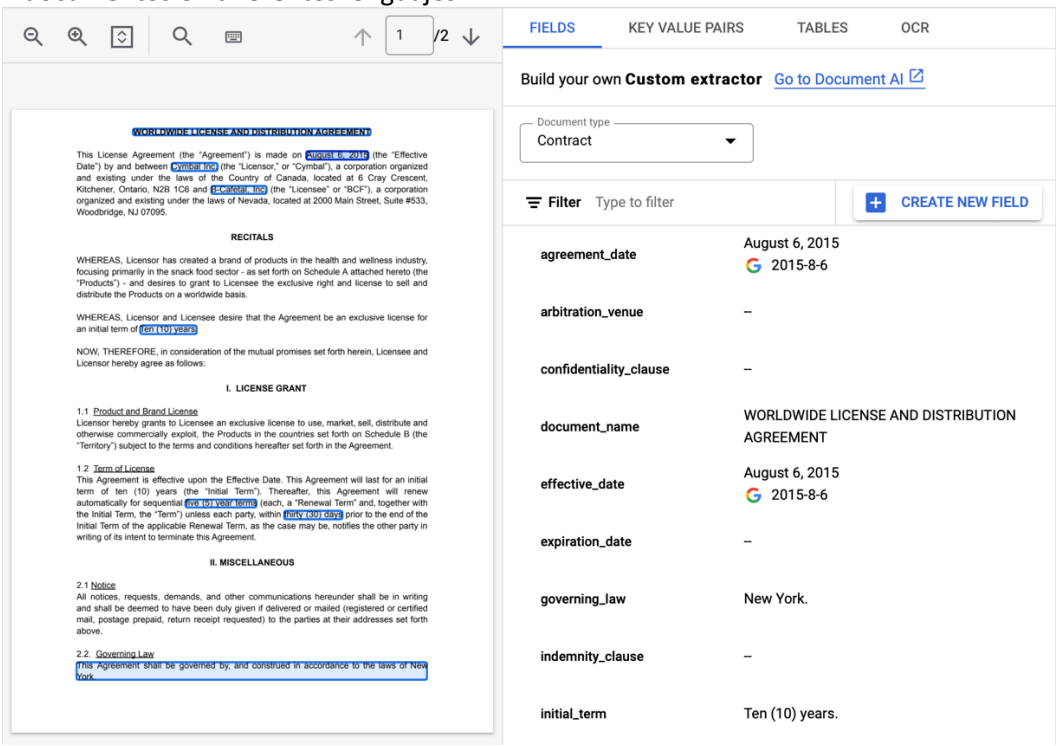


Figura 2.2.1: Google Document AI [2].

2.3. DOCLIME

Doclime [3] es un a aplicación que presenta una solución innovadora en el ámbito del reconocimiento y análisis de documentos, implementando tecnologías de reconocimiento óptico de caracteres (OCR), procesamiento del lenguaje natural (NLP) y aprendizaje automático. Esta aplicación cuenta con un chatbot integrado que permite a los usuarios hacer consultas específicas sobre el contenido de los documentos y obtener respuestas precisas y contextualizadas en tiempo real [3].

La plataforma admite la carga de documentos en diversos formatos y permite a los usuarios restringir el área de conocimiento al contenido de los documentos aportados [3]. Esto asegura que las respuestas del chatbot se limiten a la información relevante contenida en esos documentos, evitando que se salga de la temática deseada.

La característica principal de Doclime [3] radica en su interfaz conversacional, que permite a los usuarios interactuar con un chatbot para formular preguntas sobre el contenido de los documentos. Este chatbot proporciona respuestas precisas y contextualizadas basadas en las consultas realizadas, lo que facilita la obtención de información relevante de manera rápida y eficiente. Asimismo, Doclime [3] no solo ofrece respuestas generadas por el modelo de chatbot, sino que también proporciona el contenido original del documento, con la opción de abrir el documento y ubicar la información exacta solicitada por el usuario [3]. Esto asegura una referencia directa al origen de la información, mejorando la precisión y confiabilidad de las respuestas proporcionadas.

Doclime [3] también ofrece una serie de características adicionales que mejoran su funcionalidad y utilidad. Entre estas, se destacan los servicios en la nube para el almacenamiento seguro y accesible de documentos, lo que garantiza la protección de los datos y su disponibilidad en cualquier momento. Además, la aplicación cuenta con herramientas avanzadas para la clasificación y organización de documentos, facilitando su búsqueda y recuperación eficiente [3]. La infraestructura en la nube de Doclime [3] permite una escalabilidad significativa y el manejo de grandes volúmenes de datos, asegurando un rendimiento óptimo incluso en entornos de alta demanda. Adicionalmente, ofrece APIs y SDKs que permiten su integración con otros sistemas y aplicaciones, lo que facilita la personalización y extensión de sus capacidades para satisfacer necesidades específicas de los usuarios y organizaciones.

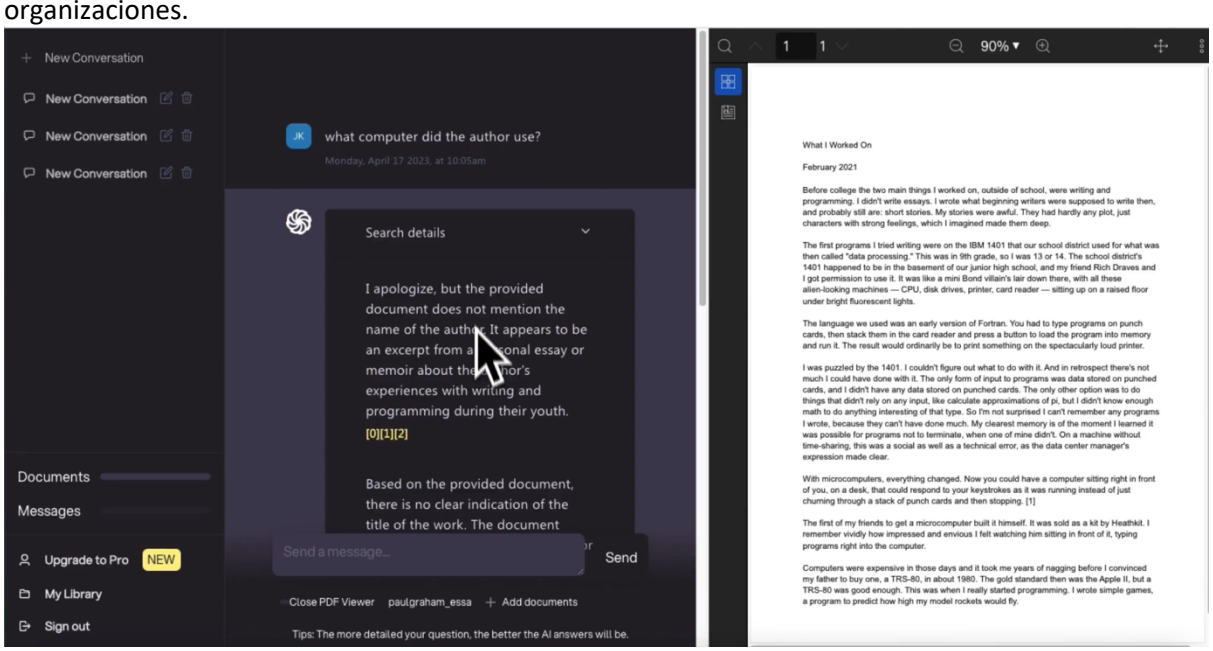


Figura 2.3.1: Doclime [3].

2.4. CHATGPT

ChatGPT [4] es una aplicación basada en el modelo de lenguaje con la arquitectura GPT-4 (Generative Pre-trained Transformer 4) desarrollado por OpenAI [41] [4]. Este modelo de inteligencia artificial utiliza una arquitectura de transformador de última generación para analizar y generar texto de manera fluida y coherente. ChatGPT [4] es capaz de comprender las consultas de los usuarios, responder con información relevante y mantener una conversación consistente incluso en diálogos extensos [4]. Esta tecnología permite una interacción natural y adaptativa, proporcionando respuestas precisas y contextuales a las necesidades de los usuarios.

La funcionalidad que se implementará en nuestra aplicación se centra en una aplicación particular del modelo ChatGPT [4], específicamente en la generación de contenido basada en un contexto previo. Aprovechando el procesamiento avanzado del lenguaje natural del modelo, esta función permitirá la creación de textos de manera automatizada. La generación de contenido puede incluir la producción de diversos textos que, aunque basados en un contexto inicial, pueden ser elaborados de manera aleatoria para ofrecer una variedad de respuestas y resultados textuales.

ChatGPT [4] ofrece una serie de características avanzadas en la generación de texto al producir contenido que fluye de manera natural y coherente manteniendo la consistencia con el contexto previo proporcionado, lo que asegura respuestas lógicas y relevantes, mientras adapta y personaliza las respuestas utilizando la información contextual del usuario, recordando detalles previos para ajustar el contenido con precisión, siendo capaz de generar una amplia variedad de textos que van desde respuestas a preguntas y explicaciones detalladas hasta textos creativos como cuentos y ensayos, abordando una gama extensa de temas y ajustando sus respuestas según el conocimiento contextual proporcionado, permitiendo además la creación de textos variados y aleatorios dentro del marco del contexto inicial, facilitando la generación de contenido novedoso, y demostrando su versatilidad y capacidad de adaptación mediante ajustes con datos especializados para mejorar la precisión en contextos específicos [4].

Figura 2.4.1: ChatGPT [4].

CAPÍTULO 3: DISEÑO FUNCIONAL

En este capítulo del proyecto se detallará el diseño inicial de la aplicación propuesta, así como la presentación de sus componentes y funcionalidades correspondientes.

3.1 DESCRIPCIÓN DEL DISEÑO

En esta sección se realizará un análisis retrospectivo de las ideas fundamentales que inicialmente inspiraron el proyecto, así como de las modificaciones que ha experimentado a lo largo de su desarrollo, influenciadas por diversos contratiempos o decisiones estratégicas de diseño.

Este proyecto surge de la imperiosa necesidad de desarrollar una alternativa educativa que mejore significativamente el proceso de estudio para estudiantes con discapacidades visuales, al mismo tiempo que se ofrece como una solución inclusiva para estudiantes en general. Desde el inicio, se ha concebido como un sistema altamente escalable capaz de gestionar integralmente la información contenida en documentos relevantes para el área de estudio de cada estudiante. Esta capacidad permite responder preguntas específicas y proporcionar información contextualizada, facilitando así la búsqueda y comprensión de contenidos de manera accesible y clara.

Además, se contempla la implementación de funcionalidades de soporte destinadas a mejorar el aprendizaje. Esto incluye el uso de audio para el control del interfaz de la aplicación, una herramienta especialmente beneficiosa para estudiantes con discapacidades visuales. También se incluye la gestión avanzada de archivos y cuentas de usuario de manera intuitiva y segura.

Otra funcionalidad clave del sistema comprenden la integración de un modelo inteligente para la generación de texto con el contenido contextual educativo proporcionado. Estas herramientas están diseñadas para optimizar el aprendizaje y la experiencia del usuario, promoviendo un entorno educativo inclusivo y adaptable a las necesidades individuales de cada estudiante.

3.2 DESCRIPCIÓN DE LOS COMPONENTES

En esta sección se ofrece una descripción de los componentes integrados en el desarrollo de la aplicación. Se detallan sus funciones específicas, el motivo de integración y las tecnologías empleadas para su incorporación efectiva en la aplicación. Se analizan los usos de cada componente dentro del contexto de la aplicación, así como la forma en que interactúan entre sí para cumplir con los requisitos del proyecto.

3.2.1 Gestión, almacenamiento y monitorización de los archivos.

Poco después de la concepción inicial del proyecto, se determinó prioritario establecer un sistema eficaz para el almacenamiento y manejo de archivos, fundamental para optimizar la gestión de documentos de los estudiantes. En consecuencia, se tomó la decisión de implementar Firestore Service [26] para el almacenamiento centralizado de documentos y archivos. Además, se optó por utilizar Filesystem [11] para el manejo operativo de los documentos dentro del entorno de Ionic Angular [9]. Pero finalmente se optó la implementación de un explorador de archivo en Python [49] como backend.

Firestore Service [26] se seleccionó debido a su robusta capacidad para almacenar y monitorizar datos de manera eficiente. Esta plataforma proporciona una solución escalable y segura que facilita el almacenamiento y recuperación de archivos esenciales para el funcionamiento integral de la aplicación educativo propuesto.

3.2.2 Modelo del reconocimiento de voz

Inicialmente, se contempló la posibilidad de implementar una interfaz operativa que permitiera controlar la aplicación mediante comandos de voz, motivado por consideraciones de escalabilidad y accesibilidad. Sin embargo, debido a las limitaciones asociadas con el entorno de desarrollo Ionic [39] y el impotente de mi dispositivo, se tomó la decisión de descartar esta funcionalidad en el ámbito de desarrollo futuro del proyecto.

En su lugar, se optó por enfocar el uso del modelo de reconocimiento de voz en aplicaciones específicas, como la toma de notas o la redacción de memorias, especialmente dirigido a mejorar la experiencia de estudiantes con discapacidades visuales. Siendo de esta forma, no solo aprovecha la potencia del modelo de reconocimiento de voz de manera efectiva, sino que también realza otras funcionalidades clave del sistema diseñadas para optimizar el aprendizaje y la interacción con el contenido educativo.

3.2.3 Explorador de archivos

Para abordar el análisis del contenido de los archivos, se ha tomado la decisión de desarrollar un explorador de archivos. La finalidad es crear una interfaz que permita visualizar y gestionar archivos de manera eficiente, incluyendo la capacidad de crear carpetas para organizar documentos. Este enfoque facilitará considerablemente el uso de la aplicación para los estudiantes, al proporcionar una manera centralizada de almacenar y administrar sus documentos.

El explorador de archivos no solo optimizará la accesibilidad a los documentos, sino que también preparará el terreno para la integración del modelo de reconocimiento de documentos. Esta integración permitirá analizar el contenido de los documentos almacenados de manera efectiva, apoyando así las funcionalidades educativas específicas diseñadas para mejorar la experiencia de aprendizaje de los usuarios.

3.2.4 Modelo de análisis de documentos especiales.

Desde el inicio, el proyecto se ha enfocado en desarrollar un modelo capaz de analizar documentos, adaptándose a las áreas de estudio específicas de los estudiantes. Sin embargo, se ha identificado que muchos documentos presentan formatos que son difíciles de reconocer o que no se reconocen correctamente para un modelo contextual. En respuesta a esta problemática, se ha decidido implementar un modelo que pueda analizar documentos en formato visual, utilizando inteligencia artificial entrenada con un amplio conjunto de documentos visuales.

Este modelo está diseñado para reconocer el contenido de documentos capturados como imágenes, permitiendo al sistema formular preguntas sobre el contenido y proporcionar respuestas con una evaluación de viabilidad y precisión. Mejorando la accesibilidad y usabilidad del sistema para los usuarios, y facilitando el análisis detallado y preciso del contenido educativo, contribuyendo así a optimizar la experiencia de aprendizaje de los estudiantes.

3.2.5 Modelo de análisis de documentos adaptado.

El objetivo más fundamentar es incorporar un modelo de análisis que pueda procesar diversos tipos de documentos textuales y analizar su contenido de manera profunda mediante un chatbot. Aportando los documentos con los conocimientos del área que quiere el estudiante, adaptando el análisis a esta área de conocimiento específica. Es decir, el chatbot de inteligencia artificial será capaz de responder preguntas sobre la información contenida en los documentos, aprovechando el conocimiento contextual del área temática de los documentos proporcionados.

Esta integración busca mejorar significativamente la capacidad de la aplicación para interactuar de manera efectiva con el contenido educativo, ofreciendo respuestas precisas y relevantes basadas en el análisis profundo realizado por el modelo de inteligencia artificial. Este enfoque aumenta la utilidad y eficiencia del sistema para los usuarios, y también fortalece su capacidad de apoyo al aprendizaje adaptativo y personalizado en diversas áreas de estudio.

3.2.6 Modelo de generación del texto.

La última funcionalidad del proyecto se centra en una herramienta diseñada para ayudar en la generación de memorias y notas por parte de los estudiantes. Este modelo utiliza contextos previos para generar textos que ofrecen sugerencias de contenido, facilitando así el proceso de escritura. Esta funcionalidad está diseñada para aumentar la eficiencia del aprendizaje al proporcionar una ayuda significativa en la creación de documentos académicos mediante la generación automática de partes del texto basadas en el contexto proporcionado, tanto para los estudiantes con discapacidades visuales o para los estudiantes sin discapacidades. Optimizando el tiempo y esfuerzo dedicados a la redacción, además también asegura la coherencia y relevancia del contenido final.

CAPÍTULO 4: TECNOLOGÍAS UTILIZADAS

En este capítulo, se demostrará las herramientas clave utilizadas en el proyecto: Ionic [39] y Python [49] para el desarrollo multiplataforma, Firebase [38] de Google Cloud para almacenamiento seguro de datos, modelos de inteligencia artificial de Huggingface [29] para procesamiento avanzado del lenguaje natural, y Langchain [45] y la API de OpenAI [41] para integrar un chat de inteligencia artificial en el proyecto.

4.1. FIREBASE

Firebase [38] es una plataforma para el desarrollo de aplicaciones móviles y web que ofrece soluciones robustas para la autenticación, gestión y almacenamiento de datos [5]. Desarrollada por Google, Firebase[38] proporciona herramientas integradas que permiten a los desarrolladores crear aplicaciones seguras y escalables de manera eficiente.

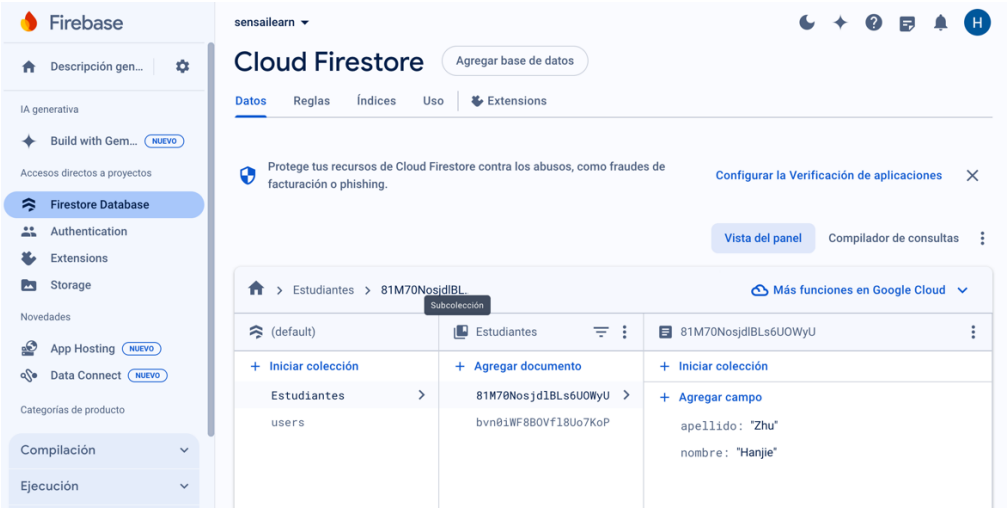


Figura 4.1.1: Panel de Firebase [5].

Configurando un proyecto en Firebase [38] para utilizar los servicios en la nube, incluyendo Firestore [26]. En este contexto, los datos del usuario se almacenan en las colecciones de Firebase Database [38], mientras que los archivos son gestionados en Firebase Storage [26].

4.2. IONIC

Ionic [39] es un framework de código abierto diseñado para el desarrollo de aplicaciones móviles y web, que permite la creación eficiente de aplicaciones multiplataforma mediante el uso de tecnologías web estándar como HTML [53], CSS [52] y JavaScript [51] [6]. Sobresale por su capacidad para habilitar el desarrollo de aplicaciones con una única base de código, la cual puede ser desplegada en diversas plataformas, incluyendo iOS, Android y la web [6]. Ionic [39] optimiza el proceso de desarrollo al facilitar la reutilización del código y asegurar la coherencia de la experiencia del usuario en todas las plataformas soportadas.

Ionic [39] ofrece ventajas significativas como la integración fluida con Capacitor y Cordova [40][42], así como componentes UI de alta calidad [7]. Angular CLI [8] simplifica la configuración del entorno y la gestión del proyecto, permitiendo a los desarrolladores enfocarse en la codificación [8]. Ionic [39] se integra perfectamente con Angular [8], lo que facilita el uso de Angular CLI [8] para la configuración

y gestión del proyecto [9]. Esta combinación ofrece una experiencia de desarrollo coherente y fluida, ya que ambos frameworks están diseñados para trabajar bien juntos, permitiendo a los desarrolladores beneficiarse de las capacidades avanzadas de Angular [8] y la flexibilidad de Ionic [39].

Ionic [39] ofrece diversas funcionalidades y plugins nativos a través de Capacitor (o Cordova) [40][42], que permiten manejarlas de manera eficiente:

- **Voice Recorder:** Plugin para la grabación de audio desde el micrófono del dispositivo [10].
- **FileSystem:** Plugin que facilita a las aplicaciones la lectura, escritura y manipulación de archivos en el dispositivo del usuario [11][40].
- **Firebase Authentication:** Plugin que proporciona servicios de autenticación mediante múltiples proveedores (correo electrónico, Google, Facebook, etc.) [12][38].
- **Haptics:** Plugin que permite agregar retroalimentación háptica a las aplicaciones [13][40].

Estos recursos nativos y funcionalidades integradas en Ionic [39] amplían las capacidades de las aplicaciones móviles desarrolladas, asegurando una experiencia de usuario rica y funcionalidades avanzadas de manera eficiente.

Figura 4.2.1: Ionic Angular CLI [9].

4.3. Visual Studio Code

Visual Studio Code [50] es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft, ampliamente reconocido por su flexibilidad y compatibilidad con diversos lenguajes de programación [14]. Este IDE ha sido seleccionado por su capacidad para integrar eficazmente HTML [53] y TypeScript [24], los dos lenguajes principales empleados en el desarrollo de la aplicación web.

La elección de Visual Studio Code [50] para el desarrollo de la aplicación usando Ionic [39] se fundamenta en su facilidad de uso, su interfaz intuitiva y su ecosistema extensible de librerías y extensiones. Entre las características destacadas se incluyen su sistema de control de versiones integrado, soporte para depuración en tiempo real, y su robusto sistema de gestión de extensiones que permite personalizar y ampliar las funcionalidades del IDE según las necesidades del proyecto [14].

4.4. HUGGINGFACE

HuggingFace [29] es una plataforma avanzada que proporciona tecnología especializada en inteligencia artificial y procesamiento del lenguaje natural (NLP) [15]. Es conocida principalmente por su biblioteca Transformers, que permite a los desarrolladores implementar y utilizar modelos de aprendizaje profundo para tareas de NLP de manera sencilla y eficiente.

Dentro del proyecto, utilizaremos varios modelos y tecnologías para el procesamiento del lenguaje natural (NLP) disponibles en el mercado a través de HuggingFace [29]. Estos modelos se aplican en una amplia gama de tareas de NLP, incluyendo el reconocimiento automático de voz, la respuesta a preguntas basadas en documentos, y la generación de textos.

El modelo de reconocimiento automático de voz que emplearemos es Whisper [33], proporcionado por OpenAI [41]. Esta tecnología permite a las máquinas convertir la voz humana en texto escrito de manera automatizada. La implementación se basa en modelos de aprendizaje automático avanzados, tales como redes neuronales recurrentes (RNN), redes neuronales convolucionales (CNN) y modelos de atención como Transformer, para llevar a cabo esta tarea de manera eficiente y precisa [16][33].

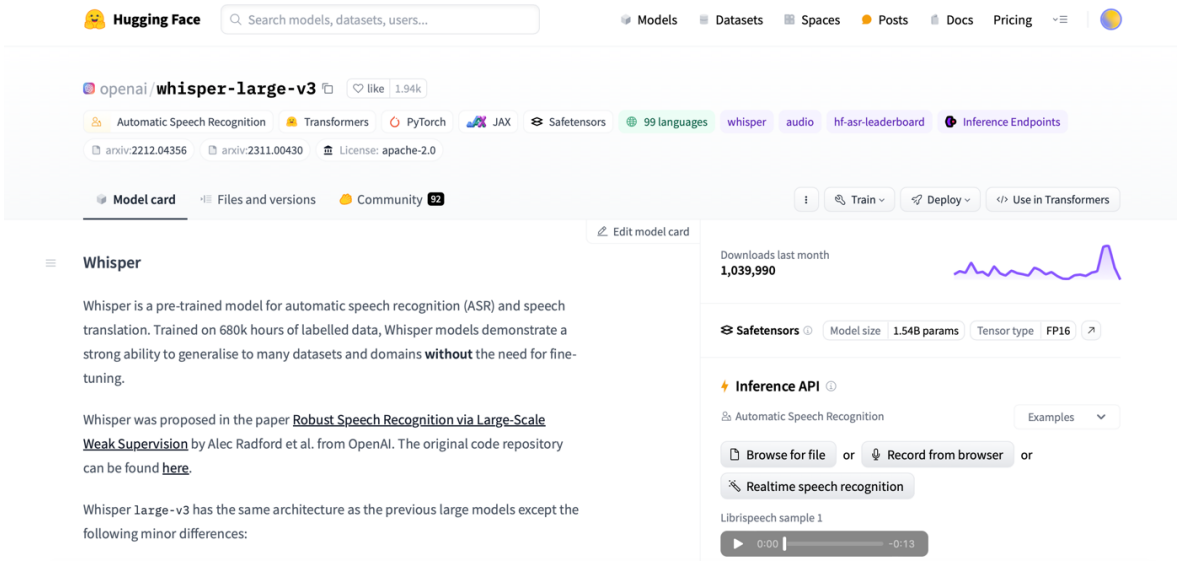


Figura 4.4.1: Whisper-large-v3 [16].

Otro modelo que empleamos es el "layoutlm-document-qa" [17][18], perteneciente a la categoría de Document Question Answering. Este modelo se basa en el modelo LayoutLM (Layout-aware Language Modeling) [17] desarrollado por Microsoft Research.

LayoutLM es un enfoque que combina la información de la estructura visual de un documento con su contenido textual para mejorar el rendimiento en tareas de procesamiento de lenguaje natural (NLP) que involucran documentos estructurados visualmente, tales como formularios, recibos, PDFs, entre otros [17].

El modelo "layoutlm-document-qa" [17][18] se especializa en responder preguntas sobre documentos. Ha sido entrenado para comprender y responder preguntas basadas en el contenido de los documentos que procesa, lo cual es especialmente útil en aplicaciones donde se necesite extraer información específica de documentos, como encontrar respuestas dentro de un formulario o un archivo PDF [17] [18].

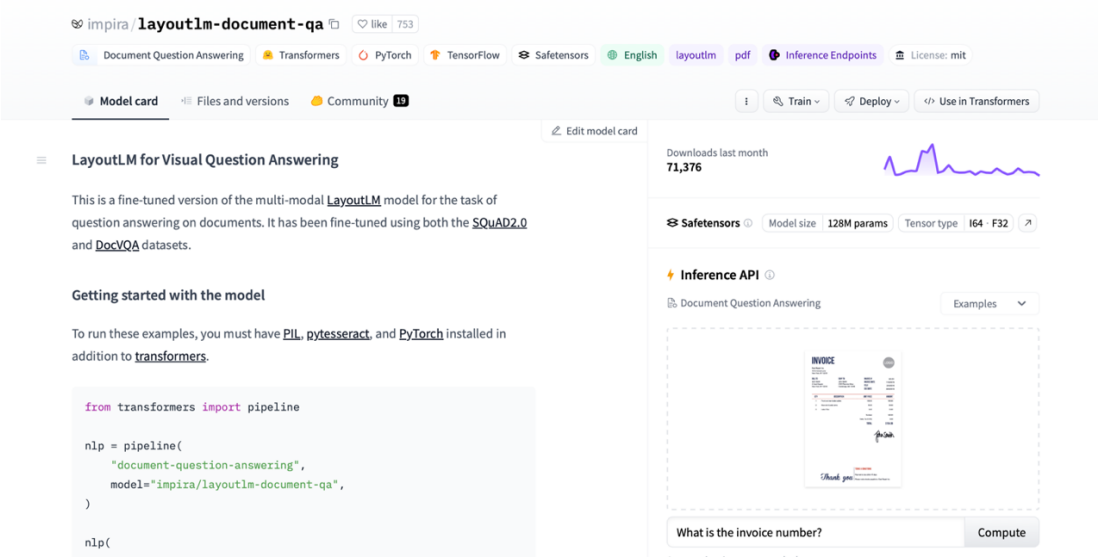


Figura 4.4.2: layoutlm-document-qa [18].

El último modelo integrado en el proyecto es "gemma-7b" [19][25] de la categoría Text Generation. Este tipo de modelo de inteligencia artificial está diseñado para generar texto de manera automática y coherente, a menudo basándose en un contexto dado. Utiliza técnicas avanzadas de aprendizaje automático, tales como redes neuronales recurrentes (RNN), transformers, o variantes más recientes como GPT (Generative Pre-trained Transformer), para producir secuencias de texto que pueden ser convincentes y relevantes en diversas aplicaciones [19][25].

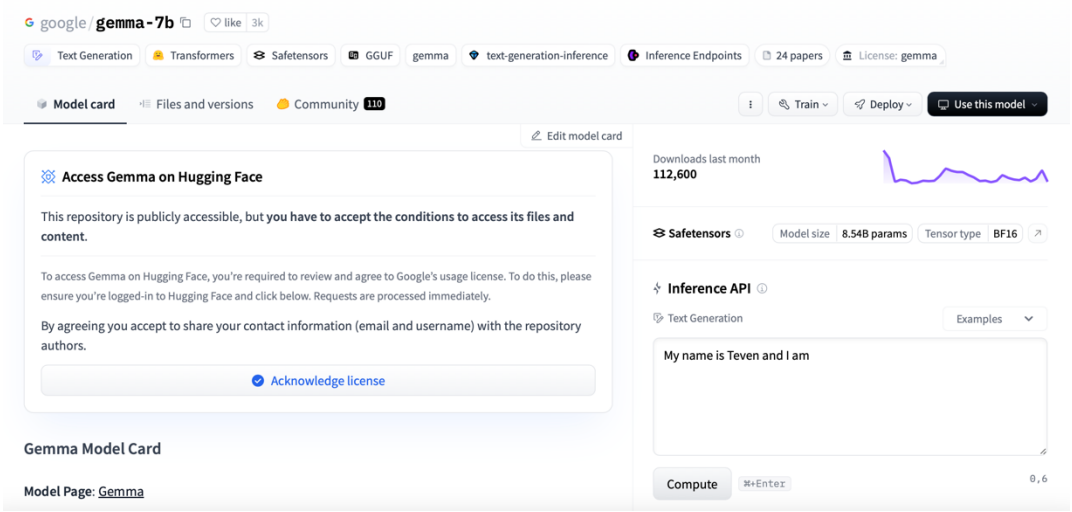


Figura 4.4.3: gemma-7b [19].

4.5. LANGCHAIN

LangChain [45] es un framework diseñado para facilitar el uso de inteligencias artificiales multilingües y multimodales (LLMs) y su propósito principal es simplificar la creación y despliegue de aplicaciones que utilizan modelos LLMs [20].

El framework también integra capacidades multimodales, permitiendo el manejo de datos de diferentes modalidades, como texto, imágenes, audio y video, lo que posibilita la creación de aplicaciones más complejas y ricas en contenido [20]. LangChain [45] busca simplificar el proceso de desarrollo de aplicaciones de inteligencia artificial mediante la provisión de herramientas y APIs que abstraen las complejidades técnicas. Esto incluye interfaces para modelos preentrenados, herramientas de etiquetado y entrenamiento de datos, entre otros recursos.

El objetivo principal de LangChain [45] es facilitar el desarrollo de aplicaciones que dependen de LLMs, proporcionando a los desarrolladores herramientas, APIs y frameworks que abstraen gran parte de la complejidad involucrada en el uso efectivo de estos modelos.

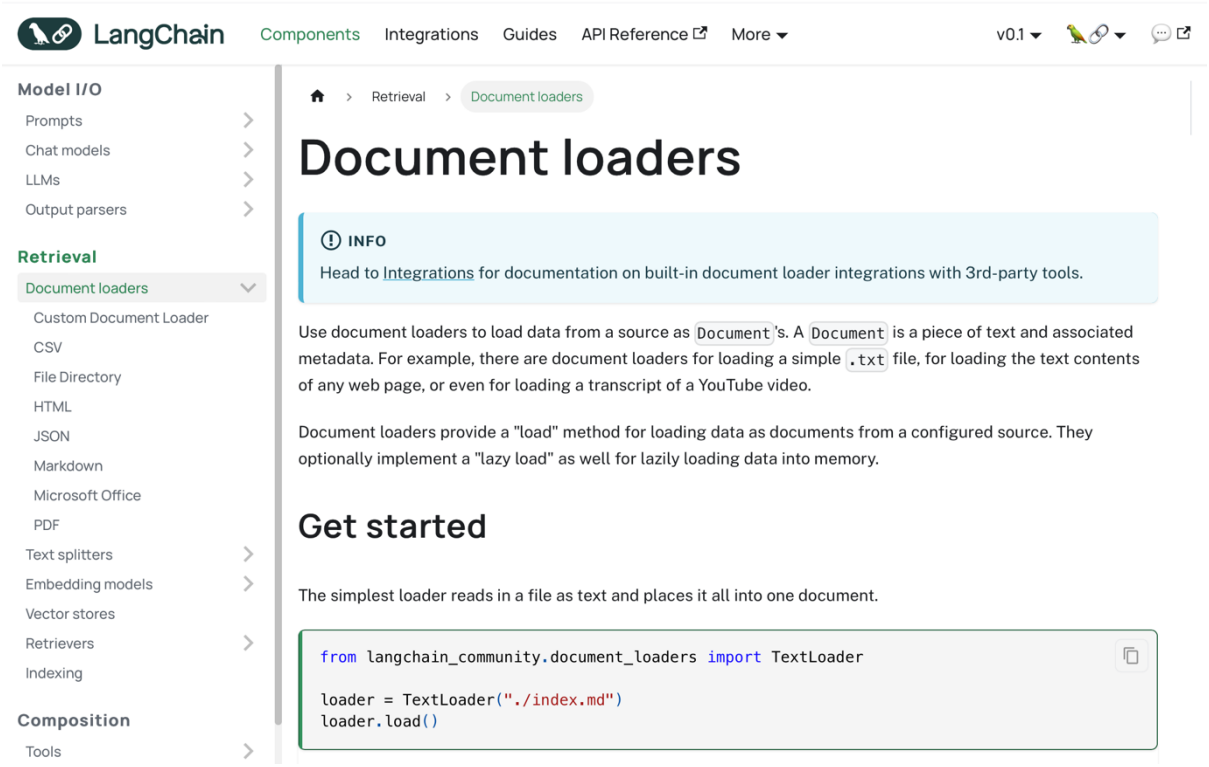


Figura 4.5.1: Langchain [21].

4.6. OPENAI API

Dentro del proyecto, utilizamos un modelo LLM (Large Language Model) de OpenAI [41] para entrenar el chatbot, adaptándolo a los documentos específicos de un área de estudio. La integración de capacidades de inteligencia artificial, basadas en modelos de lenguaje grande (LLM), se realiza mediante las herramientas proporcionadas por LangChain [45]. Esto permite que la aplicación aproveche de manera efectiva las avanzadas técnicas de procesamiento de lenguaje natural, mejorando la interacción y la capacidad de respuesta del chatbot en relación con los contenidos proporcionados.

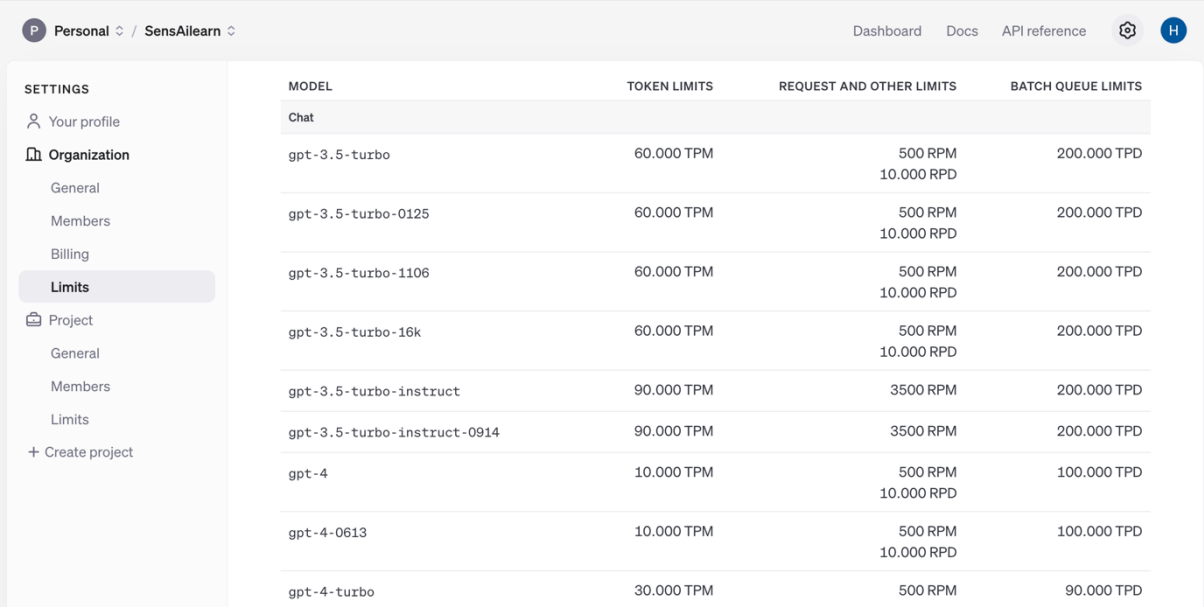


Figura 4.6.1: Modelos de OpenAI [22].

OpenAI [41] ofrece varios modelos con diferentes límites de tokens y rendimientos [22]. Por lo tanto, es fundamental crear un proyecto en la página oficial de OpenAI [41] y probar diversos modelos para determinar cuál se adapta mejor a las necesidades de la aplicación.

4.7. PYTHON

Python [49] es un lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su claridad en la sintaxis. Su diseño se orienta hacia la legibilidad del código, lo que facilita su comprensión y mantenimiento. Python [49] es versátil y se emplea en una variedad de campos, incluyendo desarrollo web, análisis de datos, inteligencia artificial, automatización y muchos otros.

La adecuada gestión del entorno de ejecución es esencial para asegurar que las dependencias y configuraciones específicas de cada proyecto se manejen de forma independiente, evitando interferencias con otros proyectos. En este contexto, utilizaremos entornos virtuales en Python [49]. Un entorno virtual es una herramienta que proporciona un espacio aislado para la instalación de paquetes y la gestión de dependencias, sin afectar el entorno global de Python [49] instalado en el sistema [49].

Esta herramienta es particularmente valiosa para proyectos que requieren versiones particulares de bibliotecas o para evitar conflictos entre distintos proyectos. Al crear un entorno virtual, se asegura que las dependencias y configuraciones sean específicas para cada proyecto, promoviendo así un desarrollo más ordenado y libre de conflictos.

CAPÍTULO 5: ARQUITECTURA Y MODELO DE DATOS

Este capítulo está dedicado a la arquitectura de los sistemas y la organización de los datos en el presente proyecto. Se explicará en detalle la estructura del sistema, la clasificación de los datos, los métodos de tratamiento aplicados y las estrategias de almacenamiento empleadas.

5.1. ARQUITECTURA

La aplicación del proyecto está compuesta por varias componentes con diferentes funcionalidades para lograr los objetivos planteado en el capítulo del diseño funcional. Pero entre ellos solo usa dos tipos de arquitectura para poder funcionar las componentes compuestas.

El desarrollo de los modelos proporcionados por HuggingFace [29] no requiere una arquitectura compleja. Solo es necesaria la integración de Ionic [39] con el servicio de Firebase [38]. Es decir, la aplicación web desarrollada en Ionic [39] actúa tanto para la interfaz visual y sus funciones, como para las conexiones con los modelos a través de API, así como la transformación de los modelos de datos en TypeScript [24] dentro de Ionic [39]. Sin embargo, la gestión y almacenamiento de los modelos de datos requiere una conexión entre Ionic [39] y la base de datos de Firebase [38].

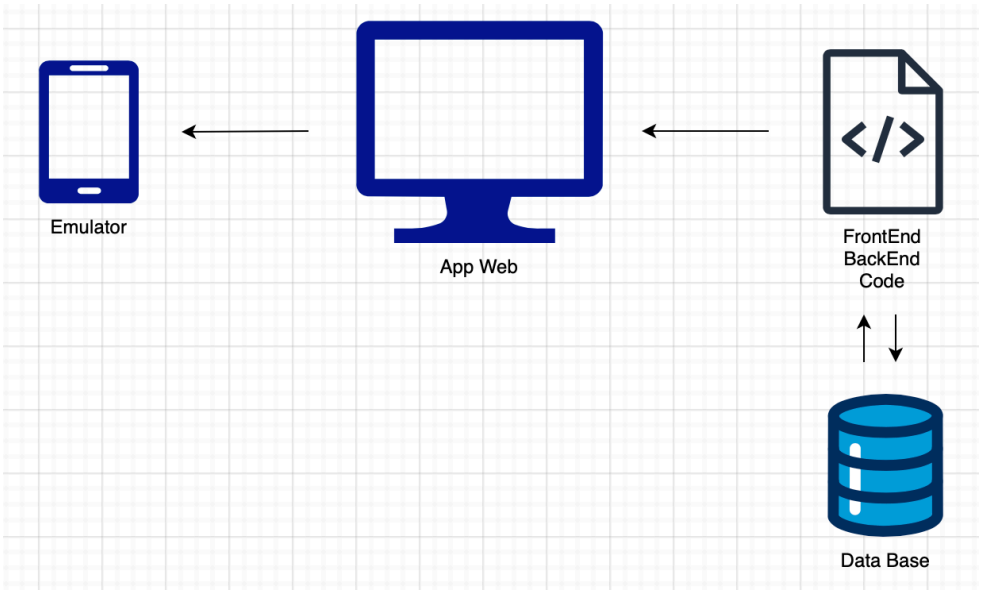


Figura 5.1.1: Arquitectura 1

La segunda arquitectura está diseñada para el funcionamiento de un chatbot que permite cargar documentos y realizar preguntas sobre su contenido. Esta arquitectura se basa en un backend desarrollado en Python [49], el cual se conecta con el frontend mediante llamadas HTTPS. El frontend está desarrollado en Ionic [39], donde se implementa la interfaz visual y sus funciones, y se gestionan las solicitudes a las funciones del backend.

Para habilitar esta comunicación, es necesario ejecutar el servidor de backend con un entorno virtual de Python [49], lo que permite la llamada de las funciones desarrolladas en Python [49] desde el servidor web creado en Ionic [39]. Al igual que en la arquitectura anterior, es posible integrar la base de datos con Firebase [38]. A pesar de, para pruebas iniciales de los documentos en la aplicación web, estos se almacenarán en el sistema de archivos de npm [34] (npm filesystem) [11] para el uso en el desarrollo de la aplicación web. Pero en esta nueva arquitectura, se almacenan en el directorio creado en backend y en Firebase [38].

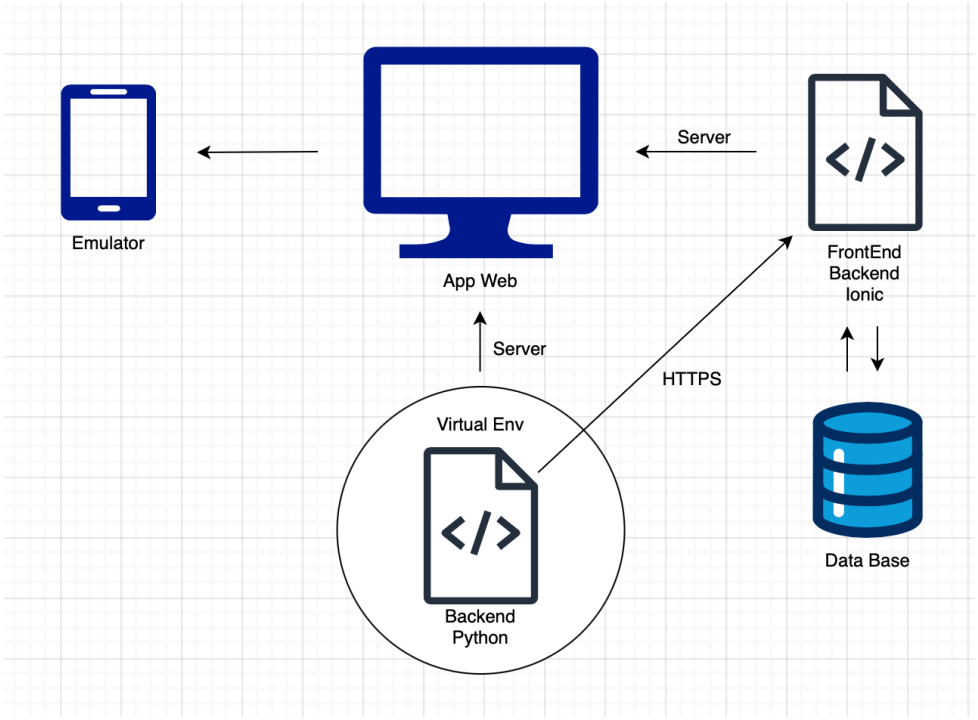


Figura 5.1.2: Arquitectura 2

5.2. MODELO DE DATOS

Los modelos de datos de este proyecto no están presentados en una gran complejidad, dado que se limitan a la gestión y almacenamiento de los datos de los alumnos. Con los usos en el procesamiento de audios mediante un modelo específico, así como el procesamiento de documentos a través de otro modelo y la implementación de un chatbot. No es necesario estructurarlos en múltiples categorías ni almacenarlos en un registro específicamente definido.

5.2.1. Modelo de datos de los datos del usuario

El primer modelo de datos consiste en una colección que incluye la información de los alumnos, así como las credenciales de usuario y contraseña para el registro en la aplicación y los datos guardados en el perfil de los usuarios. Estos datos se almacenan en Firebase [38] tras la autenticación (almacena en la autentificación del Firebase) [5] y el registro de los usuarios, constituyendo una colección de información. El avatar del usuario, utilizado como prueba del FireStorage [26], claramente se almacena en el FireStorage [26] del servicio Firebase [38].

Figura 5.2.1.1: FireBase Authentication [38].

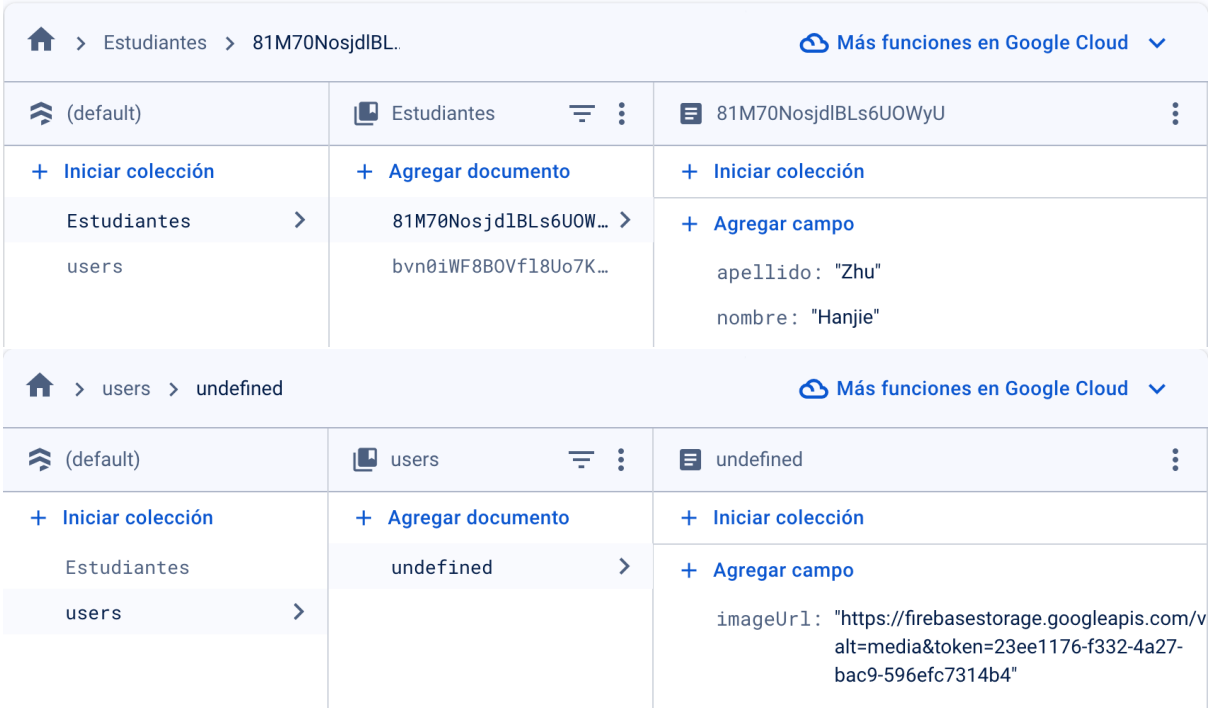


Figura 5.2.1.1: Firebase Collection [38].

5.2.2. Modelos de datos del audio

El modelo de datos del audio se encuentra codificado en formato Base64 debido al uso del módulo VoiceRecorder [10] de Capacitor [40] en Ionic [39]. Base64 es un método de codificación que convierte datos binarios en una cadena de texto ASCII, lo que resulta útil para la transferencia de datos a través de sistemas que solo manejan texto. Posteriormente, estos datos se convierten al formato MP3, que es un formato de compresión de audio digital ampliamente utilizado y que mantiene una calidad aceptable, permitiendo la reproducción en la mayoría de los reproductores y sistemas de audio para el uso de usuario.

Pero he desarrollado una función en la aplicación que permite la reproducción directa de audio en formato Base64. Esta función utiliza una referencia de audio construida a partir de datos en Base64, representados como una cadena de texto ASCII. Al emplear esta cadena Base64 en una URL de datos (data:audio/aac;base64,...), la función facilita la reproducción del contenido de audio sin necesidad de conversión adicional al formato MP3 u otros formatos de audio.

Al analizar y comprender el modelo de reconocimiento de audio automático Whisper [33], he observado que es necesario convertir el dato de audio de una cadena en formato Base64 a un formato binario, ya sea en bytes o en un buffer, que es una región de memoria destinada a manejar datos binarios. Esta conversión es esencial porque el modelo de reconocimiento de audio requiere que el audio sea interpretado y procesado en su forma binaria original para funcionar correctamente.

En consecuencia, en el capítulo de desarrollo se detalla que el modelo de datos del audio se presenta en tres formatos distintos: formato Base64, formato MP3 y dato binario.

5.2.3. Modelos de datos del documento

Para el modelo de reconocimiento de documentos impira/layoutlm-document-qa [17][18], la entrada debe consistir en un documento visual, como una imagen que puede ser una captura de pantalla, un escaneo de un documento en papel, o cualquier otra imagen que contenga texto y gráficos que el modelo pueda analizar. La entrada se debe proporcionar en formato JSON, que debe incluir tanto la consulta específica que se desea realizar sobre el contenido del documento como los datos del documento leído, que deben estar en formato Base64 o como un blob de datos.

En consecuencia, el modelo es compatible con una amplia gama de documentos visuales, los cuales pueden ser procesados y convertidos a cadenas en Base64 o a buffers de datos. Esto garantiza que diversos tipos de documentos visuales puedan ser utilizados con el modelo, siempre que se ajusten al formato de entrada requerido.

El modelo de datos para documentos textuales en el chatbot ofrece una mayor flexibilidad en los tipos de documentos que puede procesar, al centrarse en documentos en formato textual en lugar de visual. La lectura y conversión de estos documentos se realiza utilizando herramientas de LangChain [45], que ajustan los datos al formato requerido para su procesamiento en el modelo de chatbot.

LangChain [45] proporciona una serie de herramientas especializadas conocidas como **cargadores de documentos** (Document Loaders) [21], que permiten extraer y transformar el contenido de documentos textuales provenientes de diversas fuentes y formatos [21]. Estos cargadores convierten los datos a un formato compatible con los modelos de inteligencia artificial de OpenAI [41], facilitando así la integración del conocimiento contenido en los documentos en el chatbot. Los tipos de documentos compatibles con las herramientas de lectura de LangChain [45] incluyen [21]:

- **TextLoader**: Para la carga y lectura de archivos de texto plano (.txt) [21].
- **PDFLoader**: Para la extracción de contenido de archivos PDF (.pdf) [21].
- **CSVLoader**: Para la lectura de archivos de valores separados por comas (.csv) [21].
- **JSONLoader**: Para la carga y procesamiento de archivos JSON (.json) [21].
- **XMLLoader**: Para la extracción de datos de archivos XML (.xml) [21].
- **DOCXLoader**: Para la carga de documentos de Microsoft Word (.docx) [21].

Dado que el modelo no cuenta con tecnología OCR (Reconocimiento Óptico de Caracteres), es necesario que los documentos contengan información textual que pueda ser procesada sin categorización visual [22]. Por lo tanto, el modelo requiere que los documentos estén en formatos de texto plano, PDF, CSV, JSON, XML o DOCX para poder extraer y utilizar efectivamente la información contenida en ellos.

En consecuencia, el modelo de datos empleado en el chatbot se basa en documentos textuales cuyos contenidos se extraen utilizando los cargadores de documentos proporcionados por LangChain [45] [21], adaptados a cada tipo específico de documento. El proceso comienza con la creación de una cadena denominada prompt_of_file, que compila el contenido de los archivos procesados, incluyendo el nombre del archivo y su contenido en un formato estructurado.

A continuación, se preparan los mensajes para el modelo de inteligencia artificial combinando esta información extraída con el mensaje proporcionado por el usuario. Así, el prompt_of_file se integra con el mensaje del usuario en una sola solicitud. Finalmente, estos mensajes combinados se envían al modelo de IA, que los utiliza para generar una respuesta contextualizada, basada tanto en la información contenida en los archivos como en la consulta del usuario.

CAPÍTULO 6: DESARROLLO DEL PROYECTO

En esta sección del proyecto se presentará el proceso de desarrollo de la aplicación. Se describirán los pasos seguidos para implementar la aplicación y se explicará cómo se han establecido las comunicaciones entre los distintos componentes, así como las transformaciones de datos utilizados. El objetivo de esta sección es proporcionar una visión detallada de todo el proceso de desarrollo del sistema, destacando los aspectos más importantes y los retos encontrados durante el mismo.

A continuación, se comentará por orden cronológico el desarrollo e integración de cada uno de los componentes que se han implementado en el proyecto. También se detallarán los inconvenientes encontrados, donde algunos de ellos han provocado cambios en el proyecto.

6.1 IONIC ANGULAR

Para configurar un proyecto Ionic [39] con Angular CLI [8], se deben seguir los siguientes procedimientos detallados [6]:

1. **Instalación de Node.js [35] y npm [34]**: Verifica la presencia de Node.js [35] y npm [34] mediante los comandos **node -v** y **npm -v**, respectivamente, para asegurar que estén instalados. En este proyecto, se utilizan las versiones **node v21.6.1** y **npm v10.7.0**. O descárgalos e instálalos desde nodejs.org [35]**.**
2. **Instalación de Ionic CLI [39]**: Para incorporar Ionic CLI [39], la interfaz de línea de comandos de Ionic [39], se ejecuta el comando **npm install -g @ionic/cli**.
3. **Creación de un nuevo proyecto Ionic [39]**: Al emplear Angular CLI [8], se utiliza el comando **ionic start sensailearn blank --type=angular** para iniciar un nuevo proyecto Ionic [39]. Este comando genera una estructura de archivos bien organizada con directorios específicos destinados a la aplicación Ionic [39], integrando Angular CLI [8] en el proceso.

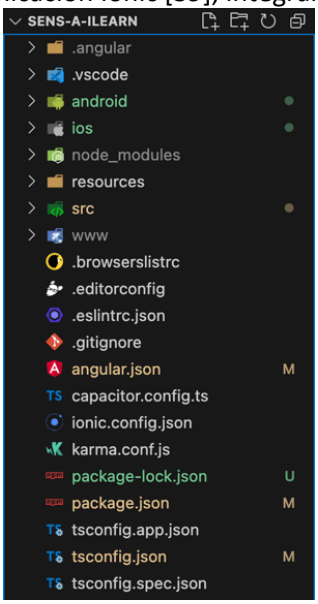


Figura 6.1.1: Archivos Ionic

4. **Ejecutación de la aplicación en el navegador:** Es necesario navegar al directorio del proyecto y ejecutar el comando **ionic serve**. Este comando se encargará de compilar y servir la aplicación, abriéndola automáticamente en el navegador predeterminado. Cualquier modificación realizada en el código se reflejará instantáneamente en la aplicación que se está ejecutando en el navegador, facilitando así el proceso de desarrollo y prueba. Este enfoque permite una iteración rápida y eficiente durante el desarrollo de aplicaciones utilizando Ionic [39] y Angular [8].

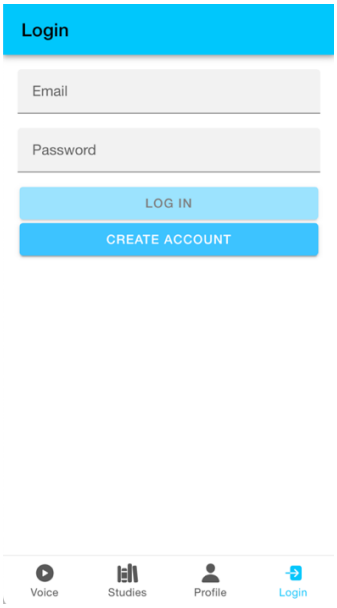


Figura 6.1.2: Aplicación Ionic

5. **Generación de la página y los servicios:** Después de haber iniciado la aplicación con **ionic serve** para crear una página inicial básica home, se utilizan los comandos **ionic g page pages/nombre de la página** y **ionic g service services/nombre del servicio** para crear nuevas páginas y archivos de servicio respectivamente, facilitando así el desarrollo de diversas funcionalidades.
6. **Instalación del Capacitor [40]:** Capacitor [40] es una herramienta fundamental en el ecosistema de Ionic [39] para la construcción de aplicaciones modernas con funcionalidades avanzadas, se realiza mediante el comando **npm install @capacitor/core @capacitor/cli**.
7. **Probar en un dispositivo físico o emulador:** Para asegurar el correcto funcionamiento de la aplicación en dispositivos físicos, se recomienda inicialmente realizar pruebas en un emulador. Esto implica agregar las plataformas deseadas utilizando los comandos **ionic capacitor add Android** y **ionic capacitor add ios**, asegurándose previamente de tener instalados los SDK correspondientes para cada plataforma. Posteriormente, se verifica el desempeño de la aplicación en dispositivos Android y iOS utilizando los comandos **ionic capacitor open Android** y **ionic capacitor open ios**, respectivamente, asegurando su correcta funcionalidad y compatibilidad con ambas plataformas móviles.

La carpeta **pages** alberga las vistas y páginas de nuestra aplicación, que corresponden a los elementos visuales que aparecen en la pantalla [23]. Cada una de estas páginas, creada con la herramienta de línea de comandos (CLI), incluye seis archivos asociados [23]:

- **nombre-routing.module.ts:** Este archivo se encarga de la configuración del enrutamiento en Angular [8], definiendo las rutas que conectan las diferentes vistas de la aplicación y gestionan la navegación entre ellas [23].
- **nombre.module.ts:** Aquí se encuentra el módulo Angular [8] específico para la página. Este módulo agrupa todas las dependencias y configuraciones necesarias para la página, facilitando su integración en el proyecto [23].
- **nombre.page.html:** El archivo HTML [53] que da forma a la estructura visual de la página. Es donde se define el contenido estático y la disposición de los elementos que los usuarios verán en la interfaz [23].
- **nombre.page.scss:** Utilizado para diseñar y aplicar los estilos específicos de la página, este archivo contiene las reglas CSS [52] que determinan cómo se verá la página a nivel de diseño y presentación [23].

- **nombre.page.spec.ts**: Un archivo destinado a pruebas unitarias. Este componente se utiliza para validar automáticamente el funcionamiento de la página mediante pruebas automatizadas, asegurando que todas las funcionalidades operen correctamente [23].
- **nombre.page.ts**: Este archivo alberga la lógica de negocio de la página, escrita en TypeScript [24]. Aquí se gestiona la funcionalidad interactiva y dinámica de la página, integrando la lógica que permite que la página responda a las acciones del usuario [23].

El archivo **package.json**, ubicado en la raíz del proyecto, desempeña un papel crucial en la gestión del mismo. Entre sus funciones principales se encuentra la administración de las dependencias de bibliotecas y paquetes del proyecto, así como la especificación de versiones asociadas. Además, contiene información detallada sobre el proyecto, incluyendo las licencias y el repositorio de código fuente. También configura diversos aspectos del comportamiento de npm [34] para el proyecto, como la selección del registro npm [34] y la configuración del entorno. Define scripts personalizados que se pueden ejecutar desde la línea de comandos, como iniciar el servidor de desarrollo con **ionic serve**, construir la aplicación con **ionic build**, y ejecutar pruebas con **npm test**, entre otros.

En ocasiones, un proyecto requiere configuraciones específicas para diferentes entornos, como desarrollo, pruebas y producción, cada uno con servidores y variables de entorno distintos. La carpeta **environment** permite establecer estas configuraciones diferenciadas de manera que, al construir la aplicación Ionic [39] utilizando una bandera específica en la línea de comandos, se utilicen automáticamente los valores correspondientes. Esto es útil para definir la URL de la API y las credenciales necesarias según cada entorno.

La carpeta **services** se utiliza para guardar los servicios del proyecto, centralizando la gestión de datos y lógica de negocio. Esto asegura una separación eficiente entre el manejo de datos y la interfaz de usuario, promoviendo un diseño más organizado y un código más fácil de mantener [23].

Para obtener más información sobre los otros CLIs y capacidades nativas contenidas en Ionic [39], se puede consultar la documentación oficial del Ionic Framework [6], donde se detallan todas estas características y funcionalidades adicionales disponibles para el desarrollo de aplicaciones web y móviles.

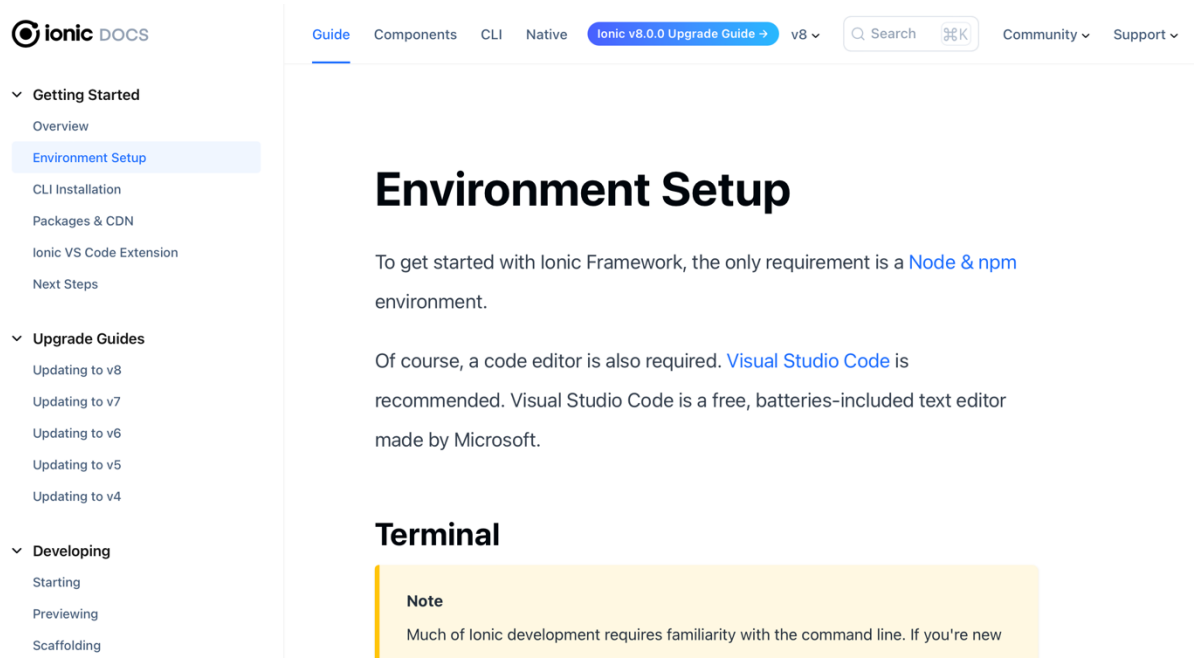


Figura 6.1.3: Ionic Docs [6].

6.2 FIREBASE

La integración de **Firebase** [38] en el proyecto se llevó a cabo en primer lugar por varias razones fundamentales. En primer lugar, su simplicidad resultó ser una ventaja significativa, permitiendo una introducción inicial al desarrollo del proyecto de forma sencilla y eficiente, con un margen de error reducido. En segundo lugar, la capacidad de Firebase [38] para el almacenamiento de archivos fue crucial, ya que inicialmente requeríamos un sistema de almacenamiento para probar los modelos de procesamiento del lenguaje natural (NLP). Finalmente, el servicio en la nube de Firebase [38] era indispensable para la gestión de los datos de los usuarios, incluyendo el inicio de sesión, el perfil y la cuenta del usuario.

La primera parte de la integración del Firebase [38] en la aplicación consta de crear un proyecto en la página oficial de Firebase [38]. Una vez creado el proyecto podemos visualizar la configuración del web.

Para gestionar usuarios, es necesario activar el servicio de **Authentication** [5]. Esto permitirá almacenar las credenciales de los usuarios (correo electrónico y contraseña) que se registren en la aplicación. Además, es imprescindible habilitar la Firebase Database [38] desde el menú correspondiente, creando una base de datos para almacenar la información de los usuarios.

Finalmente, se debe habilitar el servicio de **Storage** [26][5] desde su entrada en el menú. Se recomienda utilizar las reglas predeterminadas para asegurar que solo los usuarios autenticados puedan subir o leer archivos. De este modo, los documentos de los usuarios se almacenarán de forma segura en el almacenamiento proporcionado por Firebase [38].

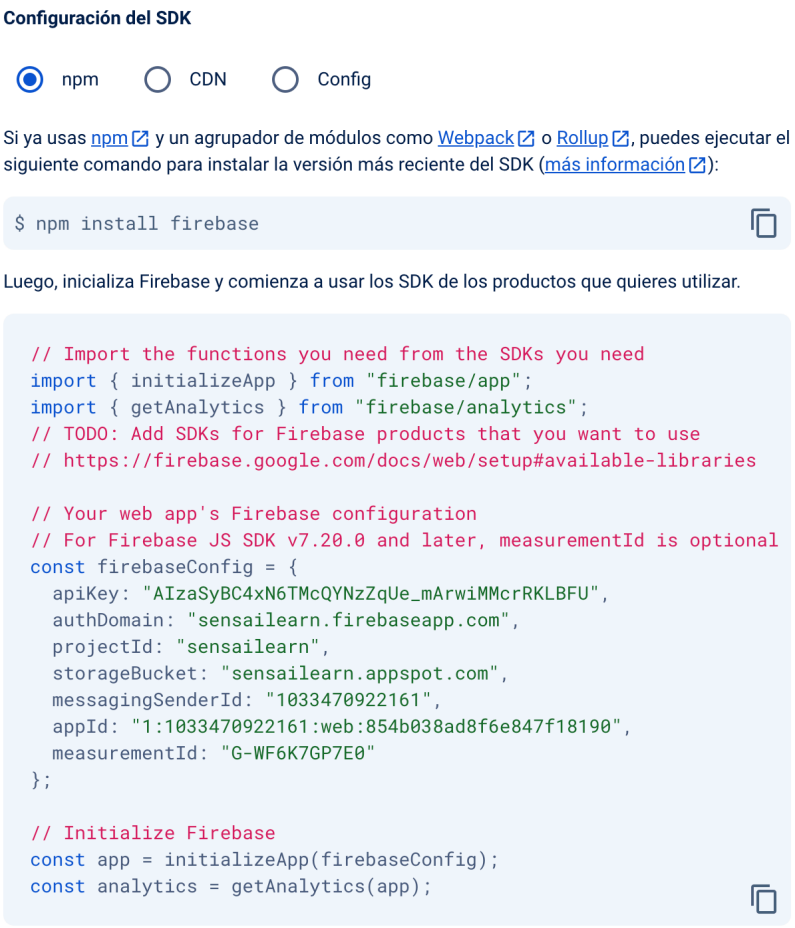


Figura 6.2.1: Configuración SDK del Firebase.

Una vez configuradas las propiedades deseadas en Firebase [38], se procede a la instalación del plugin de Firebase [5] en el proyecto **sensAIlearn** mediante el comando **ng add @angular/fire**. Durante el proceso de instalación del paquete, es necesario seleccionar las funciones que se utilizarán en el proyecto; en nuestro caso, estas funciones son **Authentication**, **Cloud Storage**, y **Firestorege [26][5]**, tal como se mencionó anteriormente.

La integración de Firebase [38] en Ionic [39] se facilita significativamente al asegurar que la configuración del entorno de Firebase [38] esté correctamente especificada en el archivo **environments/environment.ts**. Esto garantiza que la aplicación pueda comunicarse de manera efectiva con los servicios de Firebase [38] conforme a las propiedades y configuraciones definidas.

```
export const environment = {
  production: false,
  firebaseConfig: {
    apiKey: "AIzaSyBC4xN6TMcQYNzZqUe_mArwiMMcrRKLBFU",
    authDomain: "sensailearn.firebaseapp.com",
    projectId: "sensailearn",
    storageBucket: "sensailearn.appspot.com",
    messagingSenderId: "1033470922161",
    appId: "1:1033470922161:web:854b038ad8f6e847f18190",
    measurementId: "G-WF6K7GP7E0"
  },
```

Figura 6.2.2: Firebase enviroment

Una vez configuradas las propiedades deseadas en Firebase [38], se procede a inyectar las configuraciones y las inicializaciones de las tres funciones usando el archivo **src/app/app.module.ts**.

```
import { AngularFireModule } from '@angular/fire/compat';
import { environment } from 'src/environments/environment';
import { AngularFireAuthModule } from "@angular/fire/compat/auth";
import { AngularFirestoreModule } from '@angular/fire/compat/firestore';
import { initializeApp, provideFirebaseApp } from '@angular/fire/app';
import { provideAuth, getAuth } from '@angular/fire/auth';
import { provideFirestore, getFirestore } from '@angular/fire/firestore';
import { provideStorage, getStorage } from '@angular/fire/storage';
import { Capacitor } from '@capacitor/core';
import { indexedDBLocalPersistence, initializeAuth } from 'firebase/auth';
import { getApp } from 'firebase/app';

import { PreviewAnyFile } from '@ionic-native/preview-any-file/ngx';


@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    IonicModule.forRoot(),
    AppRoutingModule,
    AngularFireModule.initializeApp(environment.firebaseConfig),
    AngularFireAuthModule,
    AngularFirestoreModule,
    provideFirebaseApp(() => initializeApp(environment.firebaseConfig)),
    provideAuth(() => {
      if (Capacitor.isNativePlatform()) {
        return initializeAuth(getApp(), {
          persistence: indexedDBLocalPersistence
        });
      } else {
        return getAuth();
      }
    }),
    provideFirestore(() => getFirestore()),
    provideStorage(() => getStorage())
  ],
  providers: [
    { provide: RouteReuseStrategy, useClass: IonicRouteStrategy }
    , PreviewAnyFile
  ],
  bootstrap: [AppComponent],
})
```

Figura 6.2.3: Config de Firebase en app.module.ts

Asegurando que en el enrutamiento de nuestra aplicación se configure la página de inicio de sesión como la primera página que se muestra, y se utilice la página de inicio predeterminada como la segunda página.

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } from '@angular/router';
import {
  redirectUnauthorizedTo,
  redirectLoggedInTo,
  canActivate,
} from '@angular/fire/auth-guard';

const redirectUnauthorizedToLogin = () => redirectUnauthorizedTo(['login']);
const redirectLoggedInToHome = () => redirectLoggedInTo(['home']);

const routes: Routes = [
  { path: '', redirectTo: 'studies', pathMatch: 'full' },
  {
    path: '',
    loadChildren: () => import('./tabs/tabs.module').then(m => m.TabsPageModule)
  },
  {
    path: 'login',
    loadChildren: () => import('./tab3/tab3.module').then(m => m.Tab3PageModule),
    ...canActivate(redirectLoggedInToHome)
  },
  {
    path: 'home',
    loadChildren: () => import('./tab2/tab2.module').then(m => m.Tab2PageModule),
    ...canActivate(redirectUnauthorizedToLogin)
  },
```

Figura 6.2.4: Routing de app

A continuación, construimos la lógica de autenticación, comenzando con la codificación de las funciones de registro e inicio de sesión en el archivo **src/app/services/auth.service.ts**.

```
import { Injectable } from '@angular/core';
import {
    Auth,
    signInWithEmailAndPassword,
    createUserWithEmailAndPassword,
    signOut
} from '@angular/fire/auth';

@Injectable({
    providedIn: 'root'
})
export class AuthService {
    constructor(private auth: Auth) {}

    async register({ email, password } : { email:any, password:any }) {
        try {
            const user = await createUserWithEmailAndPassword(this.auth, email, password);
            return user;
        } catch (e) {
            return null;
        }
    }

    async login({ email, password } : { email:any, password:any }) {
        try {
            const user = await signInWithEmailAndPassword(this.auth, email, password);
            return user;
        } catch (e) {
            return null;
        }
    }

    logout() {
        return signOut(this.auth);
    }
}
```

Figura 6.2.5: Login y Register

El siguiente paso es importar el **ReactiveFormsModule**[6] en el archivo **src/app/tab3/tab3.module.ts**, lo que permite capturar la información de los usuarios durante el registro.

Finalmente, el último paso consiste en codificar las funciones de registro y login, así como una alerta para indicar cualquier fallo en el registro o inicio de sesión, en **src/app/tab3/tab3.page.ts**.

```typescript
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { AlertController, LoadingController } from '@ionic/angular';
import { AuthService } from '../services/auth.service';

@Component({
  selector: 'app-tab3',
  templateUrl: 'tab3.page.html',
  styleUrls: ['tab3.page.scss']
})
export class Tab3Page implements OnInit {
  credentials!: FormGroup;

    constructor(
        private fb: FormBuilder,
        private loadingController: LoadingController,
        private alertController: AlertController,
        private authService: AuthService,
        private router: Router,
    ) {}

    // Easy access for form fields
    get email() {
        return this.credentials.get('email');
    }

    get password() {
        return this.credentials.get('password');
    }

    ngOnInit() {
        this.credentials = this.fb.group({
            email: ['', [Validators.required, Validators.email]],
            password: ['', [Validators.required, Validators.minLength(6)]]
        });
    }

    async register() {
        const loading = await this.loadingController.create();
        await loading.present();

        const user = await this.authService.register(this.credentials.value);
        await loading.dismiss();

        if (user) {
            this.router.navigateByUrl('/home', { replaceUrl: true });
        } else {
            this.showAlert('Registration failed', 'Please try again!');
        }
    }

    async login() {
        const loading = await this.loadingController.create();
        await loading.present();

        const user = await this.authService.login(this.credentials.value);
        await loading.dismiss();

        if (user) {
            this.router.navigateByUrl('/home', { replaceUrl: true });
        } else {
            this.showAlert('Login failed', 'Please try again!');
        }
    }

    async showAlert(header: string, message: string) {
        const alert = await this.alertController.create({
            header,
            message,
            buttons: ['OK']
        });
        await alert.present();
    }

}
```

Figura 6.2.6: tab3.page.ts

Una vez construido todos anteriores, la pieza faltante es desarrollar la visual HTML [53] de la página de inicio de sesión, conectándolo con el **FormGroup** [6] definido en la página de la siguiente manera.

```
<ion-header>
    <ion-toolbar color="primary">
        <ion-title>Login</ion-title>
    </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
    <form (ngSubmit)="login()" [formGroup]="credentials">
        <ion-item fill="solid" class="ion-margin-bottom">
            <ion-input type="email" placeholder="Email" formControlName="email"></ion-input>
            <ion-note slot="error" *ngIf="email&&(email.dirty || email.touched) && email.errors"
                >Email is invalid</ion-note
            >
        </ion-item>
        <ion-item fill="solid" class="ion-margin-bottom">
            <ion-input type="password" placeholder="Password" formControlName="password"></ion-input>
            <ion-note slot="error" *ngIf="password&&(password.dirty || password.touched) && password.errors"
                >Password needs to be 6 characters</ion-note
            >
        </ion-item>

        <ion-button type="submit" expand="block" [disabled]="!credentials.valid">Log in</ion-button>
        <ion-button type="button" expand="block" color="secondary" (click)="register()"
            >Create account</ion-button
        >
    </form>
</ion-content>
```

Figura 6.2.7: tab3.page.html

Después de la conexión con Firebase [38] y la integración de la autenticación de Firebase [38] en la aplicación, procedemos a desarrollar la integración de Firestorage [26] en la aplicación, inicialmente probando con una imagen, para posteriormente utilizarlo en el almacenamiento de documentos.

Para realizar esta prueba, subimos una imagen que puede ser utilizada como avatar del perfil del usuario, almacenando la imagen en Firestorage [26]. Para ello, inicializamos Firestorage [26] y codificamos las funciones correspondientes en **src/app/services/avatar.service.ts**.

```
import { Injectable } from '@angular/core';
import { Auth } from '@angular/fire/auth';
import { doc, docData, Firestore, setDoc } from '@angular/fire/firestore';
import { getDownloadURL, ref, Storage } from '@angular/fire/storage';
import { Photo } from '@capacitor/camera';
import { Capacitor } from '@capacitor/core';
import { uploadString } from 'firebase/storage';

@Injectable({
  providedIn: 'root'
})
export class AvatarService {

  constructor(
    private auth: Auth,
    private firestore: Firestore,
    private storage: Storage
  ) { }

  getUserProfile(){
    const user = this.auth.currentUser
    const userDocRef = doc(this.firestore, `users/${user?.uid}`);
    return docData(userDocRef)
  }

  async uploadImage(cameraFile: Photo){
    const user = this.auth.currentUser;
    const path = `uploads/${user?.uid}/profile.png`;
    const storageRef = ref(this.storage, path);

    try {
      await uploadString(storageRef, cameraFile.base64String!, 'base64');
      const imageUrl = await getDownloadURL(storageRef);

      const userDocRef = doc(this.firestore, `users/${user?.uid}`);
      await setDoc(userDocRef, {
        imageUrl,
      });
      return true;

    }catch(e) {
      return null;
    }
  }
```

Figura 6.2.8: avatar.service.ts

En la página tab2 del proyecto, se implementa las funciones usando las funciones de servicio avatar creada. Se establece una suscripción a la función **getUserProfile()**, lo que permite recuperar el perfil del usuario. A continuación, se agrega una función para manejar el cierre de sesión, asegurando la correcta desconexión del usuario. Por último, se integra una función que invoca el plugin de cámara de Capacitor [40], facilitando así la actualización de la imagen del avatar.

```
27    constructor(
28      private avatarService: AvatarService,
29      private authService: AuthService,
30      private router: Router,
31      private loadingController: LoadingController,
32      private alertController: AlertController
33    ) {
34      this.avatarService.getUserProfile().subscribe((data) => {
35        this.profile = data ?? null;
36      });
37    }
38
39    async logout() {
40      await this.authService.logout();
41      this.router.navigateByUrl('/', {replaceUrl: true});
42    }
43
44    async changeImage() {
45      const image = await Camera.getPhoto({
46        quality: 90,
47        allowEditing: false,
48        resultType: CameraResultType.Base64,
49        source: CameraSource.Photos // Camera, Photos or Prompt!
50      });
51
52      if (image) {
53        const loading = await this.loadingController.create();
54        await loading.present();
55
56        const result = await this.avatarService.uploadImage(image);
57        loading.dismiss();
58
59        if (!result) {
60          const alert = await this.alertController.create({
61            header: 'Upload failed',
62            message: 'There was a problem uploading your avatar.',
63            buttons: ['OK']
64          });
65          await alert.present();
66        }
67      }
68    }
```

Figura 6.2.9: tab2.page.ts

Desarrollamos la visual HTML [53] de la página tab2, construyendo una vista para mostrar la imagen del avatar del usuario.

```
1
2   <ion-header>
3       <ion-toolbar color="primary">
4           <ion-buttons slot="start">
5               <ion-button (click)="logout()">
6                   <ion-icon slot="icon-only" name="log-out"></ion-icon>
7               </ion-button>
8           </ion-buttons>
9           <ion-title> My Profile </ion-title>
10      </ion-toolbar>
11  </ion-header>
12
13  <ion-content class="ion-padding">
14      <div class="preview">
15          <ion-avatar (click)="changeImage()">
16              <img *ngIf="profile?.['imageUrl']; else placheolder_avatar;" [src]="profile?.['imageUrl']" />
17              <ng-template #placheolder_avatar>
18                  <div class="fallback">
19                      <p>Select avatar</p>
20                  </div>
21              </ng-template>
22          </ion-avatar>
23      </div>
24  </ion-content>
```

Figura 6.2.10: tab2.page.html.

6.3 Audio Recorder y Modelo de Reconocimiento del Voz

En esta sección se detalla el desarrollo de un grabador de audio para luego su uso con un modelo de reconocimiento de voz, así como la integración del modelo utilizando la comunidad HuggingFace [29].

El primer paso consiste en la instalación del plugin capacitor-voice-recorder [10] utilizando el siguiente comando de npm [34]: **npm install --save capacitor-voice-recorder**. Para poder utilizar el plugin, es necesario añadir la clave **NSMicrophoneUsageDescription** y una descripción de su uso (de tipo string) al fichero **Info.plist**, que se genera al crear la base de la aplicación Ionic [39]. Esta configuración es esencial para solicitar permisos de uso del micrófono en dispositivos.

A continuación, se procederá al desarrollo de las funciones backend que serán utilizadas en el HTML [53] de la aplicación.

1. **Función loadFiles:** Esta función es responsable de recuperar los archivos de audio desde el sistema de archivos al inicializar la aplicación. También es necesario solicitar el permiso de uso del grabador de audio al iniciar la aplicación.
2. **Función startRecording:** Esta función inicia la grabación de audio.
3. **Función stopRecording:** Esta función detiene la grabación de audio.
4. **Función playAudio:** Esta función permite reproducir los archivos de audio grabados para pruebas y uso.

```
78   async loadFiles() {
79     Filesystem.readdir({
80       path: '',
81       directory: Directory.Data
82     }).then(result => {
83       console.log(result);
84       this.storedFileNames = result.files;
85       this.storedFileNames = this.storedFileNames.filter(file => file.name.endsWith('.mp3'))
86     });
87   }
123  startRecording(){
124    if (this.recording) {
125      return;
126    }
127    this.recording = true;
128    VoiceRecorder.startRecording();
129  }
130
131  stopRecording(){
132    if (!this.recording) {
133      return;
134    }
135    VoiceRecorder.stopRecording().then(async (result: RecordingData) => {
136      if (result.value && result.value.recordDataBase64) {
137        const recordData = result.value.recordDataBase64;
138        const time = new Date().getTime();
139        const fileName = time + '.mp3';
140        this.convertBase64ToMp3(result.value.recordDataBase64, time + '.mp3');
141        await Filesystem.writeFile({
142          path: fileName,
143          directory: Directory.Data,
144          data: recordData
145        });
146        this.loadFiles();
147        this.automaticSpeechRecognition();
148      }
149      this.recording = false;
150    })
151  }
152
153  async playFile(fileName: any) {
154    const audioFile = await Filesystem.readFile({
155      path: fileName,
156      directory: Directory.Data
157    })
158    const base64Sound = audioFile.data
159    const audioRef = new Audio(`data:audio/aac;base64,${ base64Sound }`)
160    audioRef.oncanplaythrough = () => audioRef.play()
161    audioRef.load()
162  }
```

Figura 6.3.1: audio.page.ts part1

Para optimizar la experiencia del usuario en el grabador de audio, se han implementado varias mejoras clave, que incluyen el uso del **GestureController** [27] para manejar la grabación mediante acciones táctiles, la integración del plugin **Haptics** [13] para proporcionar retroalimentación física, y la adición de funciones para calcular la duración y eliminar los archivos de audio.

El **GestureController** [27] detecta cuando el usuario mantiene presionado un botón para comenzar la grabación y detiene la grabación cuando se suelta el botón. Esta funcionalidad mejora la usabilidad al permitir un control más natural y directo de la grabación de audio.

El plugin **Haptics** [13] se utiliza para proporcionar retroalimentación táctil durante la interacción con el botón de grabación. Se activa una vibración intensa al iniciar la grabación y una vibración ligera al detenerla, lo que ofrece una confirmación táctil clara de las acciones realizadas.

```
@Component({
  selector: 'app-tab1',
  templateUrl: 'tab1.page.html',
  styleUrls: ['tab1.page.scss']
})
export class Tab1Page implements OnInit, AfterViewInit{
  recording = false;
  storedFileNames: FileInfo[] = [];
  durationDisplay = ''
  duration = 0
  @ViewChild('recordbtn', {read: ElementRef }) recordbtn!: ElementRef

  constructor(
    private firestore: FirestoreService,
    private gestureCtrl: GestureController,
    private changeDetectorRef: ChangeDetectorRef,
    ) {}

  ngOnInit(){
    this.loadFiles();
    VoiceRecorder.requestAudioRecordingPermission();
    this.automaticSpeechRecognition();
  }

  ngAfterViewInit(): void {
    const longpress = this.gestureCtrl.create({
      el: this.recordbtn.nativeElement,
      threshold: 0,
      gestureName: 'long-press',
      onStart: ev => {
        Haptics.impact({ style: ImpactStyle.Light })
        this.startRecording()
        this.calculateDuration()
      },
      onEnd: ev => {
        Haptics.impact({ style: ImpactStyle.Light })
        this.stopRecording()
      },
    }, true)

    longpress.enable()
  }
```

```
  calculateDuration() {
    if (!this.recording) {
      this.duration = 0
      this.durationDisplay = ''
      return
    }
    this.duration +=1
    const minutes = Math.floor(this.duration / 60)
    const seconds = (this.duration % 60).toString().padStart(2, '0')
    this.durationDisplay = `${ minutes }:${ seconds }`
    setTimeout(() => {
      this.calculateDuration()
    },1000)
  }
```

```
async deleteRecording(fileName: string, event: Event) {
  event.stopPropagation(); // Detiene la propagación del evento
  await Filesystem.deleteFile({
    directory: Directory.Data,
    path: fileName
  })
  this.loadFiles()
}
```

Figura 6.3.2: audio.page.ts part2

Una vez completada la configuración del grabador de audio, el siguiente paso es integrar el modelo whisper-large-v3 [33] utilizando la comunidad HuggingFace [29]. A continuación, se describen los pasos para llevar a cabo esta integración, así como las ventajas y consideraciones asociadas.

Los modelos disponibles en la comunidad HuggingFace [29] ofrecen varias ventajas significativas [15]:

1. **Gratuidad y Facilidad de Implementación:** Los modelos publicados en HuggingFace [29] son gratuitos y proporcionan una manera sencilla de ser desplegados e implementados en la aplicación. Esto facilita la incorporación de modelos avanzados sin la necesidad de desarrollar o entrenar modelos desde cero.
2. **Pruebas y Evaluación:** La plataforma permite probar los modelos directamente en su web, lo que ayuda a evaluar su rendimiento y adecuación antes de su integración completa. La comunidad HuggingFace [29] ofrece información detallada sobre cada modelo, facilitando la selección del modelo más adecuado para la aplicación.
3. **Acceso a Información Comprensible:** Los modelos en HuggingFace [29] están acompañados de documentación clara y fácil de entender, lo cual es esencial para comprender las capacidades y limitaciones del modelo elegido.

Para desplegar e integrar el modelo whisper-large-v3 [33] en la aplicación, es necesario acceder al sitio web de HuggingFace [29] donde está el modelo y seleccionar la opción **Deploy** seguida de **Inference API** [30]. Esto proporcionará la URL para realizar llamadas HTTPS al modelo publicado, así como la información necesaria para la autorización de uso de la API. Es imperativo crear una cuenta en la comunidad HuggingFace [29] y configurar los tokens de acceso, los cuales autentican la identidad en el HuggingFace Hub [29] y permiten que las aplicaciones realicen operaciones basadas en los permisos asignados a dichos tokens. Una vez completada esta configuración, se puede proceder a integrar el modelo en la aplicación utilizando la API proporcionada en el Inference API [30] y la autorización conseguida desde la configuración del access token.

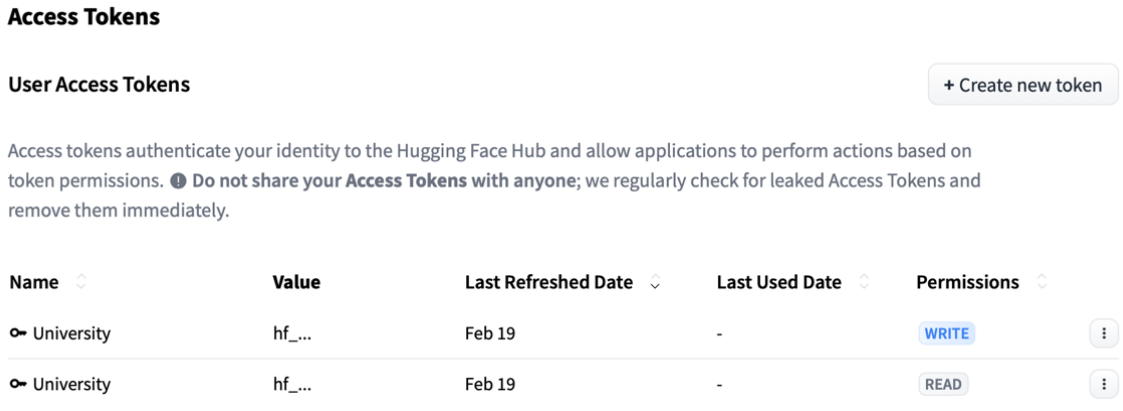


Figura 6.3.3: Access Tokens [29].

Una vez ajustadas adecuadamente las entradas y salidas de la función que integra el modelo, se podrá realizar la llamada y utilizar el modelo en la aplicación.

```
async automaticSpeechRecognition() {
  const audiofiles = await Filesystem.readdir({
    path: '',
    directory: Directory.Data
  });
  for (const file of audiofiles.files) {
    const filePath = `${''}/${file.name}`;
    const audioFile = await Filesystem.readFile({
      path: filePath,
      directory: Directory.Data
    });
    const data = audioFile.data.toString();
    const buf = Buffer.from(data, 'base64');
    const response = await fetch(
      "https://api-inference.huggingface.co/models/openai/whisper-large-v3",
      {
        headers: { Authorization: "Bearer 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" },
        method: "POST",
        body: buf,
      }
    );
    const result = await response.json();
    console.log(result);
    this.speech.push(result.text.toString());
  }
}
```

Figura 6.3.4: Modelo ASR

Importante: La transformación de los datos de entrada y la llamada a la función ASR se ejecutan directamente al terminar la grabación de audio, tal como se muestra en la función stopRecording en la imagen Figura 6.3.1.

Lo último a realizar es la implementación de HTML [53], utilizando cada una de las funciones mencionadas anteriormente.

```
<ion-header [translucent]="true">
  <ion-toolbar>
    <ion-title>
      Voice
    </ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <ion-list>
    <ng-container *ngFor="let f of storedFileNames; let i = index">
      <ion-item button (click)="playFile(f.name)" detail="false">
        {{ f.name }}
        <ion-button slot="end" (click)="deleteRecording(f.name)" fill="clear" color="danger">
          <ion-icon name="trash-outline" slot="icon-only"></ion-icon>
        </ion-button>
      </ion-item>
      <ion-item>
        <ion-label>{{ speech[i] }}</ion-label>
      </ion-item>
    </ng-container>
  </ion-list>
</ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-row class="ion-align-items-center">
      <ion-col size="10">
        <span *ngIf="!recording">
          Press and hold to record
        </span>
        <span *ngIf="recording">
          {{ durationDisplay }}
        </span>
      </ion-col>
      <ion-col size="2">
        <ion-button fill="clear" #recordbtn>
          <ion-icon name="mic-outline" slot="icon-only"></ion-icon>
        </ion-button>
      </ion-col>
    </ion-row>
  </ion-toolbar>
</ion-footer>
```

Figura 6.3.5: audio.page.html

6.4 FILE EXPLORER

Para el desarrollo del segundo modelo, que requiere la lectura de documentos, se procederá a la implementación de un explorador de archivos. Que permitirá la gestión y el almacenamiento de archivos en la aplicación, para su posterior uso con el modelo Document Question Answer**, layoutlm-document-qa** [17][18].

Primero, es necesario configurar el explorador de archivos en Capacitor [40] mediante la instalación de los siguientes plugins:

- **Plugin Capacitor Blob Writer [40]:** Este plugin facilita las operaciones de escritura de archivos. Para instalarlo, ejecute el siguiente comando: **npm install capacitor-blob-writer** [40].

- **Plugin Cordova Preview Any File [42] y Wrapper Nativo de Ionic [43]:** Este conjunto de plugins permite abrir archivos en dispositivos móviles. Instale los siguientes componentes: **npm install @ionic-native/core**, **npm install cordova-plugin-preview-any-file** y **npm install @ionic-native/preview-any-file** [42][43].

Dado que se ha incluido un wrapper nativo de Ionic [43] para el plugin de Cordova [42], es necesario añadirlo al array de providers en el archivo **src/app/app.module.ts**. Para ello, modifique el archivo **src/app/app.module.ts** e incluya **PreviewAnyFile** en la sección de providers.

Por último, en la configuración, es necesario ajustar el enrutamiento para facilitar la navegación dentro de la aplicación. Esto incluye la modificación de las rutas para aceptar un parámetro de nombre de carpeta (**:folder**). Esta configuración permitirá construir y manejar la navegación a través de los directorios de archivos en la aplicación.

```
{
  path: 'studies',
  loadChildren: () => import('./studies/studies.module').then(m => m.StudiesPageModule)
},
{
  path: 'studies/:folder',
  loadChildren: () => import('./studies/studies.module').then(m => m.StudiesPageModule)
},
```

Figura 6.4.1: studies routing file

Iniciaremos con la primera parte del desarrollo, que consiste en la visualización y creación de carpetas. Esto incluirá la carga de todos los archivos y carpetas ubicados en la página studies o en studies/folder, utilizando la función readdir del plugin Filesystem de Capacitor [11][40], para luego el uso de la visual en HTML [53].

```
40  ngOnInit() {
41    this.currentFolder = this.route.snapshot.paramMap.get('folder') || '';
42    this.loadDocuments();
43  }
44
45  async loadDocuments() {
46
47    const folderContent = await Filesystem.readdir({
48      directory: APP_DIRECTORY,
49      path: this.currentFolder
50    });
51    console.log(folderContent);
52    this.folderContent = folderContent.files.map( (file) => {
53      /*
54      const foo = await Filesystem.stat({
55        directory: APP_DIRECTORY,
56        path: this.currentFolder + '/' + file
57      });
58      console.log('STAT:', foo);
59      */
60      return {
61        name: file.name,
62        isFile: file.name.includes('.')
63      };
64    });
65  }
```

Figura 6.4.2: loadDocuments in studies page

La función de creación de carpetas se realizará utilizando **mkdir** del plugin Filesystem de Capacitor [11][40]. Esta función permite especificar el nombre de la nueva carpeta y contará con opciones para cancelar o confirmar su creación. Se definirá una ruta para la nueva carpeta dentro de la página studies. Asimismo, se procederá a cargar los archivos contenidos en la carpeta si esta ya existía previamente y contenía archivos.

```
67  async createFolder() {
68    let alert = await this.alertCtrl.create({
69      header: 'Create folder',
70      message: 'Please specify the name of the new folder',
71      inputs: [
72        {
73          name: 'name',
74          type: 'text',
75          placeholder: 'MyDir'
76        }
77      ],
78      buttons: [
79        {
80          text: 'Cancel',
81          role: 'cancel'
82        },
83        {
84          text: 'Create',
85          handler: async (data) => {
86            await Filesystem.mkdir({
87              directory: APP_DIRECTORY,
88              path: `${this.currentFolder}/${data.name}`
89            });
90            this.loadDocuments();
91          }
92        }
93      ]
94    });
95
96    await alert.present();
97  }
```

Figura 6.4.3: createFolder

Para agregar archivos a nuestro explorador, utilizamos un elemento de entrada de archivos oculto, el cual se activa para permitir la selección de archivos a través del navegador o de la aplicación nativa. El archivo seleccionado es tratado como un blob, y se emplea el plugin de escritura de blobs para almacenarlo en el sistema de archivos.

```
99  addFile() {
100   this.picker.nativeElement.click();
101 }
102
103 async fileSelected($event: any) {
104   const selected = $event.target.files[0];
105
106   await write_blob({
107     directory: APP_DIRECTORY,
108     path: `${this.currentFolder}/${selected.name}`,
109     blob: selected,
110     on_fallback(error) {
111       console.error('error: ', error);
112     }
113   });
114
115   this.loadDocuments();
116 }
```

Figura 6.4.4: addFile

Desarrollamos una función de clic que ejecuta diferentes acciones según el clic en diferentes partes realizada sobre el archivo, tales como copiar el archivo, abrir el archivo o la carpeta. Para garantizar una navegación adecuada a través de todas las carpetas, la función de clic incluye varias verificaciones. Solo si la entrada seleccionada es un directorio, se añadirá su nombre a la ruta actual. Posteriormente, se procederá a la navegación hacia dicho directorio, reinicializando la página para cargar los archivos del nuevo nivel.

```
118  async itemClicked(entry: any) {
119    if (this.copyFile) {
120      if (entry.isFile) {
121        let toast = await this.toastCtrl.create({
122          message: 'Please select a folder for your operation'
123        });
124        await toast.present();
125        return;
126      }
127      this.finishCopyFile(entry);
128    } else {
129      if (entry.isFile) {
130        this.openFile(entry);
131      } else {
132        let pathToOpen =
133          this.currentFolder != '' ? this.currentFolder + '/' + entry.name : entry.name;
134        let folder = encodeURIComponent(pathToOpen);
135        this.router.navigateByUrl(`/studies/${folder}`);
136      }
137    }
138  }
```

Figura 6.4.5: itemClicked

Para eliminar archivos y carpetas, se emplean las funciones deleteFile y rmdir del plugin Filesystem [11], proporcionando la ruta correcta del archivo o la carpeta a eliminar. Es importante considerar que, en el caso de eliminar una carpeta, es necesario eliminar primero los archivos que contiene. Esto se facilita mediante el uso de un parámetro recursivo, que permite la eliminación completa de la carpeta y su contenido.

```
193  async delete(entry: any) {
194    if (entry.isFile) {
195      await Filesystem.deleteFile({
196        directory: APP_DIRECTORY,
197        path: this.currentFolder + '/' + entry.name
198      });
199    } else {
200      await Filesystem.rmdir({
201        directory: APP_DIRECTORY,
202        path: this.currentFolder + '/' + entry.name,
203        recursive: true // Removes all files as well!
204      });
205    }
206    this.loadDocuments();
207  }
```

Figura 6.4.6: deleteFolder

La función de copia consta de dos pasos principales. En primer lugar, se activa la acción de copia en un archivo o directorio, asignando el elemento a **copyFile** para habilitar la operación de copia. En segundo lugar, al hacer clic en una carpeta seleccionada, se invoca la función **finishCopyFile()**, que efectúa la acción de pegado en la carpeta seleccionada.

Para realizar la operación de copia, es necesario obtener primero el URI absoluto del archivo y del destino mediante la función **getUri()** [11]. A continuación, estos valores se deben proporcionar a la función de copia para finalizar el proceso.

```
startCopy(file: any) {
  this.copyFile = file;
}

async finishCopyFile(entry: any) {
  // Make sure we don't have any additional slash in our path
  const current = this.currentFolder != '' ? `/${this.currentFolder}` : ''

  const from_uri = await Filesystem.getUri({
    directory: APP_DIRECTORY,
    path: `${current}/${this.copyFile?.name}`
  });

  const dest_uri = await Filesystem.getUri({
    directory: APP_DIRECTORY,
    path: `${current}/${entry.name}/${this.copyFile?.name}`
  });

  await Filesystem.copy({
    from: from_uri.uri,
    to: dest_uri.uri
  });
  this.delete(this.copyFile);
  this.copyFile = null;
  this.loadDocuments();

}
```

Figura 6.4.7: startCopyFolder

Al finalizar, se implementará la visualización de la página de studies, que se divide en dos partes principales:

1. **Menú deslizable:** La primera parte define un menú deslizable que permite el acceso a dos secciones distintas del estudio de documentos: l-studies, que se centra en el modelo de preguntas y respuestas sobre documentos visuales basado en LayoutLM [17], y d-studies, que ofrece un chat para un análisis detallado de los documentos textuales.

```
<ion-app>
  <ion-menu menuId="main-menu" contentId="main-content">
    <ion-header>
      <ion-toolbar>
        <ion-title>Menu</ion-title>
      </ion-toolbar>
    </ion-header>

    <ion-content class="ion-padding">
      <ion-list>
        <ion-menu-toggle>
          <ion-item routerLink="">
            <ion-icon name="documents" slot="start"></ion-icon>
            <ion-label>Documents</ion-label>
          </ion-item>
        </ion-menu-toggle>
        <ion-menu-toggle>
          <ion-item routerLink="l-studies">
            <ion-icon name="mail" slot="start"></ion-icon>
            <ion-label>L-studies</ion-label>
          </ion-item>
        </ion-menu-toggle>
        <ion-menu-toggle>
          <ion-item routerLink="d-studies">
            <ion-icon name="warning" slot="start"></ion-icon>
            <ion-label>D-studies</ion-label>
          </ion-item>
        </ion-menu-toggle>
      </ion-list>
    </ion-content>
  </ion-menu>
  <ion-router-outlet id="main-content"></ion-router-outlet>
```

Figura 6.4.8: studies.page.html

2. **Visual de la página de Studies:** La segunda parte corresponde a la visual de la página de studies, la cual se desarrollará utilizando las funciones previamente mencionadas para la gestión y presentación de archivos y carpetas.

```
<div class="ion-page" id="main-content">
  <ion-header>
    <ion-toolbar [color]="copyFile ? 'secondary' : 'primary'">
      <ion-buttons slot="start">
          <ng-container *ngIf="currentFolder === ''">
            <ion-menu-button></ion-menu-button>
          </ng-container>
          <ng-container *ngIf="currentFolder !== ''">
            <ion-back-button defaultHref="/"></ion-back-button>
          </ng-container>
      </ion-buttons>
      <ion-title> {{ currentFolder || 'Studies' }} </ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content class="ion-padding">
        <!-- For opening a standard file picker -->
      <input hidden type="file" #filepicker (change)="fileSelected($event)" />

      <!-- Info if the directory is empty -->
      <ion-text color="medium" *ngIf="folderContent.length == 0" class="ion-padding ion-text-center">
          <p>No documents found</p>
      </ion-text>

      <ion-list>
          <ion-item-sliding *ngFor="let f of folderContent">
              <!-- The actual file/folder item with click event -->
              <ion-item (click)="itemClicked(f)">
                  <ion-icon [name]="f.isFile ? 'document-outline' : 'folder-outline'" slot="start"></ion-icon>
                  {{ f.name }}
              </ion-item>

              <!-- The start/end option buttons for all operations -->
              <ion-item-options side="start">
                  <ion-item-option (click)="delete(f)" color="danger">
                      <ion-icon name="trash" slot="icon-only"></ion-icon>
                  </ion-item-option>
              </ion-item-options>

              <ion-item-options side="end">
                  <ion-item-option (click)="startCopy(f)" color="success"> Copy </ion-item-option>
              </ion-item-options>
          </ion-item-sliding>
      </ion-list>

      <!-- Fab to add files & folders -->
      <ion-fab vertical="bottom" horizontal="end" slot="fixed">
          <ion-fab-button>
              <ion-icon name="add"></ion-icon>
          </ion-fab-button>
          <ion-fab-list side="top">
              <ion-fab-button (click)="createFolder()">
                  <ion-icon name="folder"></ion-icon>
              </ion-fab-button>
              <ion-fab-button (click)="addFile()">
                  <ion-icon name="document"></ion-icon>
              </ion-fab-button>
          </ion-fab-list>
      </ion-fab>
  </ion-content>
</div>

</ion-app>
```

Figura 6.4.9: studies.page.html

6.5 DOCUMENT QUESTION ANSWER

En esta sección, abordaremos la integración del modelo **layoutlm-document-qa** (modelo DQA) [17] [18] en nuestra aplicación, utilizando los documentos gestionados a través del explorador de archivos descrito en la sección anterior.

La integración del modelo se desglosa en tres partes principales:

1. **Estructuración de la aplicación:** Se organiza la aplicación de manera que los usuarios puedan interactuar con el modelo de manera efectiva. Esto incluye el diseño de la interfaz y la disposición de los componentes para facilitar el uso del modelo DQA.
2. **Ajuste del tipo de entrada:** Se ajusta el tipo de entrada para que sea compatible con los requisitos del modelo, asegurando una mejor comprensión de su funcionamiento y de las directrices proporcionadas por la comunidad HuggingFace [29].
3. **Visualización de resultados:** Se desarrolla una página para mostrar los resultados del modelo y presentar la información de manera clara y accesible.

En primer lugar, se implementa una subpágina llamada **l-studies**, la cual se integra dentro de la página principal **studies** (d-studies también). La página **studies** funciona como la página de inicio (homepage) y empleando **RouterModule.forChild(routes)** [6] para gestionar la navegación hacia la subpágina **l-studies**. La subpágina **l-studies** permite a los usuarios acceder a los documentos almacenados en el sistema de archivos gestionado por la página studies y utilizarlos conforme a los requisitos del modelo.

```typescript
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { StudiesPage } from './studies.page';

const routes: Routes = [
  {
    path: '',
    component: StudiesPage
  },
  {
    path: 'l-studies',
    loadChildren: () => import('./l-studies/l-studies.module').then( m => m.LStudiesPageModule)
  },
  {
    path: 'd-studies',
    loadChildren: () => import('./d-studies/d-studies.module').then( m => m.DStudiesPageModule)
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class StudiesPageRoutingModule {}
```

Figura 6.5.1: studies-routing.module.ts

En segundo lugar, abordaremos la adaptación del tipo de entrada para el modelo. Similar al modelo anterior, HuggingFace [29] proporciona un ejemplo de despliegue del modelo utilizando **API_URL** y encabezados de autorización (User Access Tokens). En este ejemplo, el input se presenta en forma de diccionario. Sin embargo, en TypeScript [24], no se dispone de una estructura de datos equivalente, por lo que se utiliza el formato JSON como entrada para la función **docqa**.

Durante la prueba del modelo con datos en formato JSON, se ha generado un error de tipo no reconocido. Este problema se debe a que la API de HuggingFace [30] no está interpretando correctamente el formato del cuerpo de la solicitud. Para solucionar este inconveniente, es necesario

incluir la cabecera **"Content-Type": "application/json"**, que informa al servidor sobre el tipo de datos que se está enviando en el cuerpo de la solicitud. De este modo, el servidor reconocerá que el cuerpo debe ser deserializado como JSON.

```
export class LStudiesPage implements OnInit {

  documentName: string = '';
  question: string = '';
  output: any;

  constructor(private menu: MenuController) { }

  ngOnInit() {
  }

  async docqa(filename: any, qt: string) {
    console.log('Nombre del documento:', this.documentName);
    console.log('Pregunta:', this.question);
    const docFile = await Filesystem.readFile({
      path: filename,
      directory: APP_DIRECTORY
    })

    const response = await fetch(
      "https://api-inference.huggingface.co/models/impira/layoutlm-document-qa",
        {
            headers: { Authorization: "Bearer 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", "Content-Type": "application/json" },
            method: "POST",
            body: JSON.stringify({
              inputs: {
                question: qt,
                image: docFile.data
              }
            }),
        }
    );
    const result = await response.json();
    console.log(JSON.stringify(result));
    //this.output = `Nombre del documento: ${this.documentName}, Pregunta: ${this.question}, Resultado: ${JSON.stringify(res
    this.output = {
      documentName: this.documentName,
      question: this.question,
      answer: JSON.stringify(result)
    };
    return result;
  }
```

Figura 6.5.2: l-studies.page.ts

La última etapa del desarrollo consiste en diseñar la interfaz de la página **l-studies**. Esta página requiere un campo de entrada para el nombre del documento, que debe estar previamente cargado en el sistema de archivos de la página **studies**. Además, se necesita un campo para formular la pregunta sobre el documento seleccionado.

La ejecución del modelo se iniciará cuando el usuario haga clic en el botón "Send". Para ello, se deben definir dos parámetros de tipo **string**: uno para el nombre del documento y otro para la pregunta, los cuales almacenarán la información ingresada por el usuario en los campos correspondientes.

Una vez que el modelo haya terminado su ejecución, el resultado se almacenará en un parámetro de tipo **string** y se mostrará en la tarjeta, presentando el nombre del archivo, la pregunta y el resultado.

```
<ion-content [fullscreen]="true" class="content-container">
  <!-- Entrada del nombre de documento -->
  <ion-item class="custom-input">
    <ion-label position="floating">D_Name</ion-label>
    <ion-input id = "documentName" placeholder="Enter document name" type="text" [(ngModel)]="documentName"></ion-input>
  </ion-item>
  <!-- Entrada de la pregunta -->
  <ion-item class="custom-input">
    <ion-label position="floating">Question</ion-label>
    <ion-input id = "question" placeholder="Enter question" type="text" [(ngModel)]="question"></ion-input>
  </ion-item>

  <ion-button expand="block" shape="round" fill="solid" color="tertiary" class="submit-button" (click)="docqa(documentName, question)">
    Send
  </ion-button>

  <!-- Muestra la salida de docqa en una tarjeta -->
  <ion-card *ngIf="output" class="custom-card">
    <ion-card-header class="custom-card-header">
      ImpiraDocQA
    </ion-card-header>
    <ion-card-content class="custom-card-content">
      <ion-row class="custom-row">
        <ion-col size="12">
          <ion-label class="custom-label"><strong>Document Name:</strong> {{ output.documentName }}</ion-label>
        </ion-col>
      </ion-row>
      <ion-row class="custom-row">
        <ion-col size="12">
          <ion-label class="custom-label"><strong>Question:</strong> {{ output.question }}</ion-label>
        </ion-col>
      </ion-row>
      <ion-row class="custom-row">
        <ion-col size="12">
          <ion-label class="custom-label"><strong>Answer:</strong> {{ output.answer }}</ion-label>
        </ion-col>
      </ion-row>
    </ion-card-content>
  </ion-card>
</ion-content>
</div>
```

Figura 6.5.3: l-studies.page.html

6.6 CHATBOT

En esta sección, se abordará una transformación significativa en la estructura del desarrollo de la aplicación, motivada por las limitaciones encontradas al usar documentos almacenados en el sistema de archivos (filesystem) [11] y bases de datos de archivos (filebase) [38] para transmitir información al modelo de chatbot. También se presentará la implementación del chatbot y sus funcionalidades integradas, detallando el proceso paso a paso.

6.6.1. Limitaciones en el uso de Filesystem y Firebase

Debido a las desventajas asociadas con el uso del sistema de archivos desde el entorno web y base de datos, se ha decidido cambiar la estructura de desarrollo para mejorar la eficiencia y la integración con el modelo de chatbot. Las principales desventajas identificadas incluyen:

- **Interacción con el sistema de archivos:** El acceso directo a archivos desde el navegador es restringido y puede ser limitado en términos de funcionalidad y seguridad. Esto afecta la capacidad de transmitir datos de manera efectiva al modelo de chatbot.
- **Seguridad y Manejo de Archivos:** El almacenamiento de archivos en el cliente presenta riesgos en términos de seguridad y pérdida de datos. Además, manejar archivos grandes o sensibles en el cliente puede exponer información a riesgos innecesarios.
- **Transmisión de Datos:** La transferencia de datos desde sistemas de archivos o bases de datos de archivos a servicios externos como un modelo de chatbot puede ser ineficiente y propensa a errores.

6.6.2. Nueva estructura de desarrollo

Para superar las limitaciones encontradas, he decidido emplear arquitecturas B/S (Browser/Server) y C/S (Client/Server), con el objetivo de utilizar servicios basados en servidor y APIs para gestionar la interacción con el modelo de chatbot.

Dado que nuestra aplicación requiere operaciones con archivos, he observado que los navegadores presentan restricciones significativas en este aspecto. Los navegadores imponen limitaciones severas para interactuar con el sistema de archivos del dispositivo; incluso cuando permiten alguna forma de acceso, estas son bastante restringidas. Además, almacenar archivos en el cliente puede llevar a problemas como la pérdida de archivos y plantear preocupaciones de seguridad.

En contraste, optar por un servidor para manejar estas operaciones resulta ser una solución más adecuada. Por ejemplo, un servidor basado en Python [49] puede utilizar una amplia gama de APIs del sistema operativo, lo que permite la creación, modificación y eliminación de archivos, así como la configuración de red y otras acciones avanzadas. Las restricciones impuestas por los navegadores limitan significativamente la funcionalidad en comparación con lo que se puede lograr mediante un servidor.

Otro aspecto crucial para considerar es la seguridad de las claves de API. Si colocamos una clave de API en el frontend, existe el riesgo de que esta sea expuesta y utilizada de manera indebida. Utilizar un servidor para manejar estas claves no solo protege la información confidencial, sino que también previene el riesgo de abuso y garantiza una mayor seguridad en el manejo de datos sensibles.

6.6.3. Integración del modelo de chatbot

Desarrollamos una implementación para realizar llamadas a la API del modelo OpenAI [22][36][37] [41], gestionando su configuración y transmitiendo los datos de los documentos al modelo en el archivo Python [49] chat.py.

En la primera fase del desarrollo, implementé la función **get_session_history** para recuperar el historial de mensajes asociado a una sesión específica. Establecí la clave de API de OpenAI [37][41] a través de la variable de entorno **OPENAI_API_KEY** [36][37] y procedí a instanciar el modelo de lenguaje correspondiente. Además, definí un **trimmer** [20][36][47] y configuré un **ChatPromptTemplate** [20][36][47] para proporcionar instrucciones precisas al modelo sobre cómo debe responder a las consultas relacionadas con el contenido de archivos.

```python
def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = load_session_history(session_id)
    return store[session_id]


parser = StrOutputParser() # Parsear la salida del modelo de lenguaje en formato string
os.environ["OPENAI_API_KEY"] = "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"

model = ChatOpenAI(model="gpt-4o")

trimmer = trim_messages(
    max_tokens=1000000,
    strategy="last",
    token_counter=model,
    include_system=True,
    allow_partial=False,
    start_on="human",
) # Configura para recortar y ajustar la longitud de los mensajes

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are a helpful assistant. You will answer questions about file contents, i will give you file name and content.",
        ),
        MessagesPlaceholder(variable_name="messages"),
    ]
) # Proporciona instrucciones al modelo de lenguaje
```

Figura 6.6.3.1: chat.py part1

En la segunda fase, configuré la cadena de procesamiento integrando el prompt con el modelo de lenguaje e implementé la función **RunnableWithMessageHistory** [20][36][47]. Desarrollé la función chat para transmitir los datos de los documentos al modelo y que el historial de la sesión se actualice y se guarde adecuadamente tras el procesamiento del mensaje.

```python
chain = prompt | model # Cadena de procesamiento en la que el prompt se pasa al model.
# Combina la cadena de procesamiento con la func de mantener el historial de mensajes.
with_message_history = RunnableWithMessageHistory(chain, get_session_history)
config = {"configurable": {"session_id": "abc2"}}
UPLOAD_DIRECTORY = "uploaded_files/"

# Procesa el mensaje de chat y devuelve una respuesta.
def chat(session_id, prompt, filename=None):
    config = {"configurable": {"session_id": session_id}}
    print(config)
    prompt_of_file = ""

    if filename is None:  # Procesamiento sin archivos
        messages = [HumanMessage(content=prompt)]
        response = with_message_history.invoke(messages, config=config)
    else:  # Procesamiento con archivos
        for file_item in filename:
            file_content = ""
            if '.pdf' in file_item:
                loader = PyPDFium2Loader(UPLOAD_DIRECTORY + file_item)
                data = loader.load()
                page_index = 1
                for page in data:
                    file_content += f"page{page_index}: {str(page)} \n"
                    page_index += 1
            elif '.txt' in file_item:
                with open(UPLOAD_DIRECTORY + file_item, 'r') as file:
                    file_content = file.read()
            prompt_of_file += f"""
                File name: `{file_item}`
                File content: `{file_content}`
            """
        messages = [
            HumanMessage(content=prompt_of_file),
            HumanMessage(content=prompt)
        ]
        response = with_message_history.invoke(messages, config=config)

    # Guarda el historial actualizado
    save_session_history(session_id, get_session_history(session_id))

    return response.content
```

Figura 6.6.3.2: chat.py part2

En el código mostrado en la Figura anterior, se puede observar que los archivos de documentos se cargan desde la carpeta **uploaded_files**. Esta carpeta fue creada en el archivo main.py del backend, junto con el endpoint asociado a la función chat.

```python
# Directorio donde se guardarán los archivos subidos
UPLOAD_DIRECTORY = "uploaded_files"

if not os.path.exists(UPLOAD_DIRECTORY):
    os.makedirs(UPLOAD_DIRECTORY)

class ChatRequest(BaseModel):
    sessionid: Optional[str] = None # ID de sesión opcional; se genera uno nuevo si no se proporciona
    humanText: str
    filename: Optional[List[str]] = None
# Endpoint para enviar un prompt de chat y obtener una respuesta del modelo
@app.post("/chat/")
async def submit_prompt(request: ChatRequest):
    # Genera un sessionid si el id proporcionado es vacío o None
    if not request.sessionid:
        request.sessionid = str(uuid.uuid4())[0:8]

    # Llama a la función de chat con el ID de sesión, el texto del humano y el nombre del archivo
    robotMessage = chat(request.sessionid, request.humanText, filename=request.filename)
    return {"sessionid": request.sessionid, "robotMessage": robotMessage}
```

Figura 6.6.3.3: Endpoint chat

En el frontend, realizamos una llamada a este endpoint proporcionando los datos como los nombres de los documentos para la consulta, el mensaje emitido por el usuario y el ID de la sesión, en el archivo d-studies.page.ts.

```typescript
// Envía mensajes al servidor de chatbot y maneja la respuesta
sendMessage() {
  // Verificar que el mensaje nuevo no está vacío
  if (this.newMessage.trim().length > 0) {
    // Mensaje del usuario
    this.messages.push({
      message: this.newMessage,
      isHuman: true
    });
    this.newMessage = '';
    this.loading = true;
    // Enviar el solicitud POST al servidor de chatbot
    this.http.post<{sessionid: string, robotMessage: string}>(`http://localhost:8000/chat`, {
      humanText: this.messages[this.messages.length - 1].message,
      filename: this.selectedFiles.length > 0 ? this.selectedFiles.map(file =>
        file.path.replace(/\\/g, '/')
      ) : null,
      sessionid: this.selectedSessionId
    }).subscribe(data => { // La respuesta del servidor de chatbot
      this.selectedSessionId = data.sessionid;
      this.messages.push({
        message: data.robotMessage,
        isHuman: false
      });
      this.loading = false;
    });

    /// Usar setTimeout para ejecutar loadSessions después de un breve intervalo
    setTimeout(() => {
      this.loadSessions(); // Recargar sesiones después de un pequeño retraso
    }, 200);
  }
}
```

Figura 6.6.3.4: sendMessage

Gestionamos la respuesta de la solicitud POST, que incluye el ID de la sesión y el mensaje del chatbot, y la implementamos en la interfaz de usuario a través del código HTML [53].

```
<!-- Contenido principal de la página, con los mensajes del bot y humano -->
<ion-content>
  <div class="message-list">
    <!-- Itera sobre la lista de mensajes y los muestra con una clase según el tipo (humano o bot) -->
    <div *ngFor="let message of messages" [ngClass]="{'human': message.isHuman, 'bot': !message.isHuman}" class="message-item">
      <ion-icon [name]="message.isHuman ? 'person-circle' : 'happy'"></ion-icon>
      <div class="message-content">
        {{message.message}}
      </div>
    </div>
  </div>

  <!-- Muestra un spinner mientras se está cargando una respuesta -->
  <ion-spinner *ngIf="loading"></ion-spinner>
</ion-content>

<ion-footer>
  <ion-toolbar>
    <ion-item>
      <!-- Campo de entrada de texto para escribir nuevos mensajes -->
      <ion-input [(ngModel)]="newMessage" placeholder="Type a message"></ion-input>

      <!-- Botón de grabación -->
      <ion-button fill="clear" #recordbtn class="record-button">
        <ion-icon name="mic-outline" slot="icon-only"></ion-icon>
      </ion-button>

      <!-- Botón de envío -->
      <ion-button [disabled]="loading" (click)="sendMessage()" fill="clear" class="send-button">
        <ion-icon name="send" slot="icon-only" class="send-icon"></ion-icon>
      </ion-button>
    </ion-item>
  </ion-toolbar>
</ion-footer>
```

Figura 6.6.3.5: d-studies.page.html part 2

6.6.4. Operaciones de archivos

En esta nueva arquitectura, las operaciones y el almacenamiento de archivos se gestionan desde el backend en el archivo main.py de Python [49]. En consecuencia, es necesario modificar las funciones previas en **studies**. En primer lugar, se define un endpoint para listar archivos y carpetas dentro de un directorio, utilizando la ruta del directorio como dato de entrada.

```
class FileItem(BaseModel):
    name: str
    isFile: bool
    path: str
# Endpoint para listar archivos y carpetas en un directorio
@app.get("/files/", response_model=List[FileItem])
async def list_files(folder_path: str = Query("", alias="folder")):
    full_directory = os.path.join(UPLOAD_DIRECTORY, folder_path)

    # Verificación del directorio
    if not os.path.exists(full_directory):
        raise HTTPException(status_code=404, detail="Directory does not exist")

    if not os.path.isdir(full_directory):
        raise HTTPException(status_code=400, detail="Provided path is not a directory")

    # Lista los archivos y directorios
    items = []
    for item in os.listdir(full_directory):
        file_path = os.path.join(full_directory, item)
        item_path = os.path.relpath(os.path.join(full_directory, item), UPLOAD_DIRECTORY)
        item_path = item_path.replace(os.path.sep, "/")

        items.append(FileItem(name=item, isFile=os.path.isfile(file_path), path=item_path))

    return items
```

Figura 6.6.4.1: Endpoint files

A continuación, se realiza una solicitud GET desde el frontend de la página **studies** para obtener los datos de la respuesta. Estos datos se utilizan para actualizar la variable folderContent, que incluye información sobre si un elemento es una carpeta o un archivo, así como el nombre del elemento en el directorio actual. Esta variable se utiliza para mostrar los archivos y las carpetas del directorio actual en la interfaz de usuario a través del código HTML [53].

```
77   ngOnInit() {
78     // Obtiene la ruta de la carpeta actual desde la URL
79     this.currentFolder = this.route.snapshot.paramMap.get('folder') || '';
80     this.route.paramMap.subscribe((params) => {
81       const folder: string | null = params.get('path');
82       this.folder = folder || '';
83       this.currentFolder = folder || '';
84       this.loadDocuments();
85     });
86
87   }
88   // Listar el contenido de la carpeta actual desde el backend
89   async loadDocuments() {
90     this.http.get<FolderContent[]>(`http://localhost:8000/files/?folder=${this.folder}`)
91       .subscribe(data => {
92         this.folderContent = data;
93       });
94   }
```

Figura 6.6.4.2: Frontend files

En relación con la operación de carga de archivos, he definido un endpoint en main.py que acepta como entrada el archivo seleccionado desde el dispositivo local y la ruta del directorio donde se desea almacenar el archivo.

```
49  # Endpoint para subir archivos
50  @app.post("/upload/")
51  async def upload_file(file: UploadFile = File(...), folder_path: str = Form("")):
52      # Construye la ruta completa del directorio donde se guardará el archivo
53      full_directory = os.path.join(UPLOAD_DIRECTORY, folder_path) if folder_path else UPLOAD_DIRECTORY
54      print(full_directory)
55
56      # Construye la ruta completa del archivo
57      if not os.path.exists(full_directory):
58          os.makedirs(full_directory)
59
60      # Construye la ruta completa del archivo incluyendo el nombre del archivo
61      file_location = os.path.join(full_directory, file.filename)
62
63      # Guardar el archivo
64      with open(file_location, "wb+") as file_object:
65          file_object.write(file.file.read())
66
67      return {"info": f"file '{file.filename}' saved at '{file_location}'"}
```

Figura 6.6.4.3: Endpoint upload

En el frontend, se realiza una solicitud POST para cargar el archivo en el directorio especificado. No es necesario recibir información adicional en la respuesta; simplemente se realiza un nuevo listado del directorio para actualizar la vista. Además, el archivo también se carga en Firebase [38].

```
139   // Maneja el evento cuando se selecciona un archivo
140   async fileSelected($event: any) {
141     // Intenta subir el archivo a Firebase Storage
142     try {
143       this.fireBase.uploadFile($event.target.files[0], this.folder);
144     } catch (err) {
145
146     }
147     // Construir una solicitud que incluye datos de formulario, incluyendo archivos.
148     const selected = $event.target.files[0];
149     const formData = new FormData();
150     formData.append("file", selected);
151     if (this.folder) {
152       formData.append("folder_path", this.folder);
153     }
154     // Envía el archivo al Backend con una solicitud POST
155     this.http.post(`http://localhost:8000/upload/`, formData).subscribe((data) => {
156       this.loadDocuments();
157     });
158   }
```

Figura 6.6.4.4: Frontend upload

Ahora, abordemos la operación de creación de carpetas. Utilizamos el endpoint create_folder en main.py, que recibe como entrada el nombre de la carpeta que se desea crear.

```python
class FolderRequest(BaseModel):
    folder_path: str
# Endpoint para crear una nueva carpeta
@app.post("/create_folder/")
async def create_folder(request: FolderRequest):
    # Construye la ruta completa de la carpeta
    full_directory = os.path.join(UPLOAD_DIRECTORY, request.folder_path)
    print(full_directory)

    # Verifica si la carpeta ya existe
    if os.path.exists(full_directory):
        raise HTTPException(status_code=400, detail="Folder already exists")

    # Crea la nueva carpeta
    os.makedirs(full_directory)

    return {"info": f"folder '{request.folder_path}' created at '{full_directory}'"}
```

Figura 6.6.4.5: Endpoint create_folder

El endpoint crea una carpeta en el backend con el nombre proporcionado, ubicándola en el directorio padre uploaded_files. En el frontend, solo es necesario realizar la llamada POST al endpoint y actualizar el listado de archivos y carpetas en la interfaz de usuario después de hacer clic en "Create". Esta llamada POST se puede integrar en la función **createFolder** dentro de studies.

```javascript
// Muestra un cuadro de diálogo para crear una nueva carpeta
async createFolder() {
  let alert = await this.alertCtrl.create({
    header: 'Create folder',
    message: 'Please specify the name of the new folder',
    inputs: [
      {
        name: 'name',
        type: 'text',
        placeholder: 'MyDir'
      }
    ],
    buttons: [
      {
        text: 'Cancel',
        role: 'cancel'
      },
      {
        text: 'Create',
        handler: async (data) => {
          await this.http.post(`http://localhost:8000/create_folder/`, {
            folder_path: data.name
          }).toPromise();
          this.loadDocuments();
        }
      }
    ]
  });
  await alert.present();
}
```

Figura 6.6.4.6: Frontend CreateFolder

El siguiente paso es desarrollar la operación de eliminación de archivos y carpetas. En el backend, se implementa un endpoint que recibe como entrada el nombre del archivo o la carpeta que se desea eliminar. Este endpoint se encarga de eliminar el archivo o la carpeta correspondiente en el directorio uploaded_files. En el frontend, después de realizar la solicitud de eliminación, lo mismo se actualiza el listado de archivos y carpetas del directorio actual para reflejar los cambios.

```python
# Endpoint para eliminar un archivo o carpeta
@app.delete("/delete/")
async def delete_item(item_path: str = Query(...)):
    full_path = os.path.join(UPLOAD_DIRECTORY, item_path)

    # Verifica la existencia del archivo o directorio
    if not os.path.exists(full_path):
        raise HTTPException(status_code=404, detail="File or directory does not exist")

    # Elimina el archivo o directorio
    if os.path.isfile(full_path):
        os.remove(full_path)
    elif os.path.isdir(full_path):
        os.rmdir(full_path)
    else:
        raise HTTPException(status_code=400, detail="Provided path is neither a file nor a directory")

    return {"info": f"'{item_path}' has been deleted"}
```

```javascript
// Elimina un archivo o carpeta del backend y recarga la lista de documentos
async delete(entry: any) {
  this.http.delete(`http://localhost:8000/delete/?item_path=${entry.path}`).subscribe((data) => {
    this.loadDocuments();
  });

}
```

Figura 6.6.4.7: Endpoint and Frontend delete

Finalmente, se implementa la operación de copiar archivos de un directorio a otro. En el backend, se define un endpoint copy en main.py que recibe como entrada la ruta del directorio actual y la ruta del directorio de destino donde se debe realizar la copia. Este endpoint se encarga de copiar el archivo desde el directorio origen al directorio destino.

```python
# Endpoint para copiar un archivo
@app.post("/copy/")
async def copy_file(old_file_path: str = Query(...), new_file_path: str = Query(...)):
    full_old_path = os.path.join(UPLOAD_DIRECTORY, old_file_path)
    full_new_path = os.path.join(UPLOAD_DIRECTORY, new_file_path)

    # Verificar la existencia del archivo original
    if not os.path.exists(full_old_path):
        raise HTTPException(status_code=404, detail="Old file does not exist")

    # Verifica que el archivo origen sea un archivo
    if not os.path.isfile(full_old_path):
        raise HTTPException(status_code=400, detail="Old path is not a file")

    # Asegura que existe el directorio destino
    new_directory = os.path.dirname(full_new_path)
    if not os.path.exists(new_directory):
        os.makedirs(new_directory)

    # Copiar archivo
    shutil.copy2(full_old_path, full_new_path)

    return {"info": f"file copied from '{old_file_path}' to '{new_file_path}'"}
```

Figura 6.6.4.8: Endpoint copy

En el frontend, la operación de copia de archivos se gestiona en dos fases principales. Primero, la función startCopy se encarga de iniciar el proceso de copia al recibir la información del archivo, incluyendo su ruta y nombre, y almacenar estos datos en Firebase [38]. Posteriormente, la función paste recupera la información almacenada en Firebase [38], realiza una solicitud POST al endpoint correspondiente para completar la copia del archivo al directorio destino, actualiza el listado de archivos del directorio actual y restablece las variables de Firebase [38] utilizadas durante el proceso.

```
// Inicia el proceso de copia de un archivo, guardando la ruta y el nombre del archivo copiado en Firebase
startCopy(file: any) {
  this.fileService.setCopiedFilePath(file.path);
  this.fileService.setFilename(file.name);
}

// Pega el archivo copiado en la carpeta actual, obteniendo los datos desde Firebase
paste() {
  const filename = this.fileService.getFilename();
  const filePath = this.fileService.getCopiedFilePath();
  const folder = this.route.snapshot.paramMap.get('path') || '';
  const newPath = folder ? `${folder}/${filename}` : filename;
  this.http.post(`http://localhost:8000/copy/?old_file_path=${filePath}&new_file_path=${newPath}`, {}).subscribe((data) => {
    this.loadDocuments();
    this.fileService.setCopiedFilePath('');
  });
}
```

Figura 6.6.4.9: Frontend startCopy and paste

La imagen siguiente proporciona una vista general del código HTML [53] para la página "studies". En esta vista, se presenta una estructura integral de la interfaz de usuario, que incluye diversos elementos y funcionalidades mencionadas. No se realizará una explicación detallada de cada componente, ya que las funciones mencionadas se utilizan de manera directa y evidente en el contexto de la aplicación.

```
<div class="ion-page" id="main-content">
  <ion-header>
    <ion-toolbar [color]="copyFile ? 'secondary' : 'primary'">
      <ion-buttons slot="start">
        <!-- Botón de menú o botón de retroceso, dependiendo del estado de currentFolder -->
        <ng-container *ngIf="currentFolder === ''">
          <ion-menu-button></ion-menu-button>
        </ng-container>
        <ng-container *ngIf="currentFolder !== ''">
          <ion-back-button defaultHref="/"></ion-back-button>
        </ng-container>
      </ion-buttons>
      <ion-title> {{ currentFolder || 'Studies' }} </ion-title>
    </ion-toolbar>
  </ion-header>
  <ion-content class="ion-padding">
      <!-- Selector de archivos del dispositivo local -->
      <input hidden type="file" #filepicker (change)="fileSelected($event)" />

      <!-- Mensaje si el directorio está vacío -->
      <ion-text color="medium" *ngIf="folderContent.length == 0" class="ion-padding ion-text-center">
          <p>No documents found</p>
      </ion-text>

      <!-- Lista de archivos y carpetas -->
      <ion-list>
          <ion-item-sliding *ngFor="let f of folderContent">
              <!-- The actual file/folder item with click event -->
              <ion-item (click)="itemClicked(f)">
                  <ion-icon [name]="f.isFile ? 'document-outline' : 'folder-outline'" slot="start"></ion-icon>
                  {{ f.name }}
              </ion-item>

              <!-- Opciones de inicio/fin para operaciones de eliminar y copiar -->
              <ion-item-options side="start">
                  <ion-item-option (click)="delete(f)" color="danger">
                      <ion-icon name="trash" slot="icon-only"></ion-icon>
                  </ion-item-option>
              </ion-item-options>
```

Figura 6.6.4.10: new studies.page.html part 1

```
76
77                  <ion-item-options side="end">
78                      <ion-item-option (click)="startCopy(f)" color="success"> Copy </ion-item-option>
79                  </ion-item-options>
80              </ion-item-sliding>
81          </ion-list>
82
83          <!-- Botón flotante para operación de pegar -->
84          <ion-fab *ngIf="fileService.copiedFilePath" vertical="bottom" horizontal="end" slot="fixed" style="bottom
85              <ion-fab-button (click)="paste()">
86                  <ion-icon name="clipboard-outline"></ion-icon>
87              </ion-fab-button>
88          </ion-fab>
89
90          <!-- Botones flotantes para agregar archivos y carpetas -->
91          <ion-fab vertical="bottom" horizontal="end" slot="fixed">
92              <ion-fab-button>
93                  <ion-icon name="add"></ion-icon>
94              </ion-fab-button>
95              <ion-fab-list side="top">
96                  <ion-fab-button (click)="createFolder()">
97                      <ion-icon name="folder"></ion-icon>
98                  </ion-fab-button>
99                  <ion-fab-button (click)="addFile()">
100                     <ion-icon name="document"></ion-icon>
101                 </ion-fab-button>
102             </ion-fab-list>
103         </ion-fab>
104
105         <!-- Botón flotante para abrir el modal de generación de texto -->
106         <ion-fab vertical="bottom" horizontal="start" slot="fixed">
107             <ion-fab-button (click)="openTextGenerationModal()">
108                 <ion-icon name = create-outline></ion-icon>
109             </ion-fab-button>
110         </ion-fab>
111
112     </ion-content>
```

Figura 6.6.4.11: new studies.page.html part 2

6.6.5. Selector de archivos para el chatbot

En la página de **d-studies**, donde se lleva a cabo la consulta de documentos mediante un chatbot integrado con un modelo de OpenAI [22][41], se ha incorporado un selector de archivos. Este selector utiliza la información de archivos y carpetas proporcionada desde la página de **studies** (es decir, que contiene las informaciones del árbol del directorio uploaded_files de backend), permitiendo seleccionar los documentos para transmitir tanto los contenidos como los nombres de los documentos al modelo de chatbot, así podrá realizar las consultas sobre los documentos seleccionados.

Para implementar este selector de archivos, es necesario crear un endpoint que permita obtener las informaciones de archivos y carpetas del directorio uploaded_files de backend, así como las relaciones hijo-padre del árbol del directorio. Luego, transmite este dato a fronend en la variable fileData.

```python
class FileInfo(BaseModel):
    name: str
    isFile: bool
    path: str
    children: list['FileInfo'] = None
# Recorrer un directorio y sus subdirectorios para crear una lista de objetos FileInfo
def get_directory_contents(path: str, relative_path: str) -> List[FileInfo]:
    items = []
    for item in os.listdir(path):
        item_path = os.path.join(path, item)
        item_relative_path = os.path.join(relative_path, item)
        if os.path.isdir(item_path):
            items.append(FileInfo(name=item, isFile=False, path=item_relative_path,
                                  children=get_directory_contents(item_path, item_relative_path)))
        else:
            items.append(FileInfo(name=item, isFile=True, path=item_relative_path))
    return items
# Endpoint para listar archivos y carpetas de forma recursiva en el árbol de archivos
@app.get("/all-files/", response_model=List[FileInfo])
async def list_files(folder_path: str = Query("", alias="folder")):
    full_directory = os.path.join(UPLOAD_DIRECTORY, folder_path)

    # Verificación del directorio
    if not os.path.exists(full_directory):
        raise HTTPException(status_code=404, detail="Directory does not exist")

    if not os.path.isdir(full_directory):
        raise HTTPException(status_code=400, detail="Provided path is not a directory")

    return get_directory_contents(full_directory, folder_path)
```

```typescript
  // Carga el árbol de archivos desde el servidor
  async loadDocuments() {
    this.http.get<FileItem[]>(`http://localhost:8000/all-files/`)
    .subscribe(data => {
      this.fileData = data;
    });
  }
```

Figura 6.6.5.1: Endpoint and Frontend all-files

Para la implementación del selector de archivos en el chatbot en el frondend, he desarrollado un componente hijo denominado **file-selector** para la página de d-studies. Este componente se encarga de emitir la información sobre los documentos seleccionados, integrándose en el frontend para gestionar la selección de archivos.

El componente file-selector incluye funciones para la visualización de los archivos y carpetas disponibles, y estas funcionalidades se reflejan en la interfaz de usuario mediante código HTML [53]. La implementación asegura que los datos de los documentos seleccionados sean transmitidos correctamente a la página d-studies.

```ts
interface FileItem {
  name: string;
  isFile: boolean;
  children?: FileItem[];
  path: string;
}

@Component({
  selector: 'app-file-selector',
  templateUrl: './file-selector.component.html',
  styleUrls: ['./file-selector.component.scss'],
})

export class FileSelectorComponent {
  @Input() files: FileItem[] = [];
  // 1. Define el evento para emitir datos de FileItem y si está seleccionado
  @Output() onSelectFile: EventEmitter<FileItem & {checked: boolean}> = new EventEmitter();

  openedFolders: Set<string> = new Set();
  selectedFiles: Set<FileItem> = new Set();

  // Alterna el estado de apertura de una carpeta.
  toggleFolder(folder: FileItem) {
    // Si la carpeta está abierta, se cierra, y viceversa
    if (this.openedFolders.has(folder.name)) {
      this.openedFolders.delete(folder.name);
    } else {
      this.openedFolders.add(folder.name);
    }
  }
  // Alterna la selección de un archivo y
  // emite un evento con la información del archivo y su estado de selección.
  toggleFileSelection(file: FileItem, e: any) {
    // Si el archivo está seleccionado, se deselecciona, y viceversa
    console.log(e.target.checked)
    if (this.selectedFiles.has(file)) {
      this.selectedFiles.delete(file);
    } else {
      this.selectedFiles.add(file);
    }
    // 2. Emitir el evento
    this.onSelectFile.emit({...file, checked: e.target.checked});
  }

  // Verifica si una carpeta está abierta.
  isFolderOpened(folder: FileItem): boolean {
    return this.openedFolders.has(folder.name);
  }

  // Verifica si un archivo está seleccionado.
  isFileSelected(file: FileItem): boolean {
    return this.selectedFiles.has(file);
  }
}
```

Figura 6.6.5.2: file-selector.component.ts

```html
<ion-list>
  <ng-container *ngFor="let file of files">
    <!-- Si el elemento es una carpeta -->
    <ion-item *ngIf="!file.isFile" (click)="toggleFolder(file)">
      <ion-icon [name]="isFolderOpened(file) ? 'folder-open' : 'folder'" slot="start"></ion-icon>
      <ion-label>{{ file.name }}</ion-label>
      <ion-icon name="chevron-forward-outline" slot="end"></ion-icon>
    </ion-item>
    <!-- Si el elemento es un archivo -->
    <ion-item *ngIf="file.isFile" (click)="toggleFileSelection(file, $event)">
      <ion-icon name="document" slot="start"></ion-icon>
      <ion-label>{{ file.name }}</ion-label>
      <ion-checkbox slot="end" [checked]="isFileSelected(file)"></ion-checkbox>
    </ion-item>
    <!-- Lista de archivos y carpetas anidados -->
    <ion-list *ngIf="!file.isFile && isFolderOpened(file)" class="nested-list">
      <app-file-selector [files]="file?.children || []" (onSelectFile)="onSelectFile.emit($event)"></app-file-selector>
    </ion-list>
  </ng-container>
</ion-list>
```

Figura 6.6.5.3: file-selector.component.html

Los datos de los documentos seleccionados se envían a la página d-studies, donde se procesan y almacenan en la variable selectedFiles. Esta variable se utiliza para transmitir los nombres de los documentos seleccionados en la llamada POST de la función sendMessage (Figura 6.6.3.4). Además, la interfaz del componente file-selector se implementa en la página d-studies mediante HTML [53], utilizando la información del árbol de directorio uploaded_files para la visualización de los archivos y carpetas disponibles.

```
89    // 3. Desarrolla el método para actualizar la lista de archivos seleccionados
90    // según si se marca o desmarca un archivo
91    handleSelectedFiles(selectedFile: FileItem & {checked: boolean}) {
92      if (selectedFile.checked) {
93        this.selectedFiles.push(selectedFile);
94      } else {
95        this.selectedFiles = this.selectedFiles
96          .filter(file => file.path !== selectedFile.path)
97      }
98      this.sessionid = '';
99    }
```

Figura 6.6.5.4: handleSlectedFiles

```
1   <!-- Menu lateral que se despliega para seleccionar archivos -->
2   <ion-menu contentId="file-selector" *ngIf="openMenu" menu-id="file-menu" (ionDidClose)="onCloseMenu()">
3     <ion-header>
4       <ion-toolbar>
5         <ion-title>Select Files</ion-title>
6       </ion-toolbar>
7     </ion-header>
8     <ion-content>
9       <!-- Componente que muestra la lista de archivos y maneja la selección de archivos -->
10      <!-- 4.Vincula el evento del componente hijo con el método del componente padre -->
11      <app-file-selector [files]="fileData" (onSelectFile)="handleSelectedFiles($event)"></app-file-selector>
12    </ion-content>
13  </ion-menu>
```

Figura 6.6.5.5: menú de file selector

Finalmente, añadimos un botón para abrir el menú de selección en la página d-studies. Al cerrar el menú, se envía automáticamente un mensaje de instrucción al chatbot, informando que puede asistir al usuario en la comprensión de los documentos seleccionados.

```
41          <!-- Botón para abrir el menú de selección de archivos -->
42          <ion-buttons id="file-selector"  (click)="openFilesDashboard()" slot="start">
43            <ion-icon size="large" name="folder"></ion-icon>
44          </ion-buttons>
```

```
102   // Abre el menú de selección de archivos
103   openFilesDashboard() {
104     this.openMenu = true;
105     console.log('open')
106     setTimeout(() => {
107       this.menuCtrl.open('file-menu')
108     }, 800)
109   }
110
111   // Maneja el cierre del menú de file selector y muestra un mensaje de introducción de chatbot
112   onCloseMenu() {
113     if (this.selectedFiles.length > 0) {
114
115       const fileNamesStr = this.selectedFiles.map(
116         file => file.name
117       ).join(', ');
118       this.sessionid = '';
119       this.messages = [{
120         message: `Hello! I can assist you with understanding the contents of the selected document: ${fileNamesStr}.
121         isHuman: false
122       }]
123     }
124   }
```

Figura 6.6.5.6: Abre y cierre de menú

6.6.6. Sesiones de chatbot

En la página **d-studies**, se ha implementado un selector para acceder a sesiones anteriores del chatbot, permitiendo a los usuarios reutilizarlas. Estas sesiones están almacenadas en la carpeta sessions_file, que contiene el historial de mensajes. Los usuarios pueden seleccionar una sesión anterior para recargarla, facilitando así el uso continuo de las interacciones previas del chatbot.

Para la implementación del selector de sesiones anteriores, es fundamental almacenar estas sesiones en una carpeta del backend. He desarrollado las funciones necesarias para esta tarea de la siguiente manera.

```python
store = {}

SESSION_DIR = "sessions_file"
os.makedirs(SESSION_DIR, exist_ok=True)

# Convierte un mensaje en un diccionario serializable a JSON
def message_to_dict(message):
    return {
        'type': message.__class__.__name__,  # Guarda el tipo de mensaje
        'content': message.content,
        'metadata': message.response_metadata if hasattr(message, 'response_metadata') else {}
    }

# Convierte un diccionario en un objeto de mensaje.
def dict_to_message(message_dict):
    if message_dict['type'] == 'HumanMessage':
        return HumanMessage(content=message_dict['content'])
    elif message_dict['type'] == 'AIMessage':
        return AIMessage(
            content=message_dict['content'],
            response_metadata=message_dict.get('metadata', {})
        )
    else:
        raise ValueError(f"Tipo de mensaje desconocido: {message_dict['type']}")

# Convierte el historial de mensajes en un diccionario serializable a JSON.
def history_to_dict(history: BaseChatMessageHistory) -> dict:
    messages = history.messages
    return [message_to_dict(message) for message in messages]
```

```python
# Convierte un diccionario que representa el historial de mensajes
# en una instancia de BaseChatMessageHistory.
def dict_to_history(history_dict: dict) -> BaseChatMessageHistory:
    history = InMemoryChatMessageHistory()
    for message_dict in history_dict:
        message = dict_to_message(message_dict)
        history.add_message(message)
    return history

# Almacena el historial de la sesión en SESSION_DIR
def save_session_history(session_id: str, history: BaseChatMessageHistory):
    file_path = os.path.join(SESSION_DIR, f"{session_id}.json")
    with open(file_path, 'w') as file:
        json.dump(history_to_dict(history), file)
    print(f"Historial de sesión {session_id} guardado en {file_path}")

# Cargar el historial de la sesión en SESSION_DIR
def load_session_history(session_id: str) -> BaseChatMessageHistory:
    file_path = os.path.join(SESSION_DIR, f"{session_id}.json")
    if not os.path.exists(file_path):
        print(f"No se encontró historial para la sesión {session_id}")
        return InMemoryChatMessageHistory()

    with open(file_path, 'r') as file:
        history_dict = json.load(file)

    return dict_to_history(history_dict)

# Obtener el historial de la sesión en store
def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = load_session_history(session_id)
    return store[session_id]
```

Figura 6.6.6.1: Funciones para el historial de la sesión

Cada vez que se invoca la función chat, el historial de mensajes se guarda en un archivo cuyo nombre corresponde al ID de la sesión de chatbot (Figura 6.6.3.2). Para facilitar la recuperación de sesiones y la implementación de un selector de sesiones, es necesario transmitir los datos de los IDs de las sesiones y sus historiales al frontend. Por lo tanto, he desarrollado los siguientes endpoints y funciones para la página d-studies.

```python
# Endpoint para obtener la lista de sesiones ids disponibles
@app.get("/sessions/")
async def get_sessions():
    # Lista de archivos de sesión en el directorio SESSION_DIR
    session_files = [f for f in os.listdir(SESSION_DIR) if f.endswith('.json')]
    # Extrae los IDs de sesión eliminando la extensión de archivo
    session_ids = [os.path.splitext(f)[0] for f in session_files]
    return {"sessions": session_ids}

# Endpoint para obtener el historial de una sesión específica
@app.get("/sessions/{session_id}/")
async def get_session_hist(session_id: str):
    try:
        # Obtiene el historial de la sesión usando el sessionid proporcionado
        history = get_session_history(session_id)
        return {
            "session_id": session_id,
            "history": history_to_dict(history)  # Convertimos el historial a un formato serializable
        }
    except FileNotFoundError:
        raise HTTPException(status_code=404, detail="Session not found")
```

```typescript
  // Carga la lista de sesiones ids desde el servidor
  loadSessions(): void {
    this.http.get<{ sessions: string[] }>('http://localhost:8000/sessions/')
      .subscribe(response => {
        this.sessionsid = response.sessions;
        this.changeDetectorRef.detectChanges();
      }, error => {
        console.error('Error al obtener sesiones:', error);
      });
  }

  // Maneja la selección de una sesión usando la sesionid
  onSessionSelect(sessionId: string) {
    if (sessionId === '') {
      // Deseleccionar la sesión actual
      this.selectedSessionId = '';
      this.messages = [];
    } else {
      // Seleccionar una nueva sesión
      this.selectedSessionId = sessionId;
      this.loadInitialMessages(sessionId);
    }
  }

  // Carga los mensajes iniciales de la sesión seleccionada
  loadInitialMessages(sessionId: string) {
    this.http.get<{ history: Message[] }>(`http://localhost:8000/sessions/${sessionId}/`)
      .subscribe(response => {
        this.messages = response.history.map((msg: any) => ({
          message: msg.content,
          isHuman: msg.type === 'HumanMessage'
        }));
      });
  }
```

Figura 6.6.6.2: Endpoints y funciones de frontend para las sesiones

Con los datos de los IDs de sesión y la función onSessionSelect, podemos implementar el selector de sesiones en HTML [53]. Además, al proporcionar el historial de una sesión específica basado en su ID, podemos implementar la visualización de dicho historial en HTML [53] mediante la variable messages. Esta variable contendrá los mensajes asociados con la sesión seleccionada, permitiendo su presentación en la interfaz de usuario.

```
<!-- Selector de sesión -->
<ion-select [(ngModel)]="selectedSessionId" (ionChange)="onSessionSelect($event.detail.value)" style="flex: 1;">
  <!-- Opción para desmarcar sesión actual -->
  <ion-select-option [value]="''">New session</ion-select-option>
  <!-- Opciones generadas dinámicamente a partir de la lista de sesiones -->
  <ion-select-option *ngFor="let sessionid of sessionsid" [value]="sessionid">
    <ion-item>
      <ion-label>{{ sessionid }}</ion-label>
    </ion-item>
  </ion-select-option>
</ion-select>
```

Figura 6.6.6.3: Selector de sesión

He implementado también una opción para eliminar la sesión seleccionada desde el archivo session_file. Esto se realiza mediante un botón en la interfaz HTML [53] que llama a una función para eliminar la sesión. Esta función transmite el ID de la sesión al endpoint correspondiente en el backend, lo que resulta en la eliminación del archivo de sesión en el servidor. Posteriormente, se actualiza el selector de sesiones en la interfaz, se limpia el ID de la sesión y se vacía la variable messages, eliminando así el historial de la sesión de la vista. Esto permite que la interfaz refleje correctamente la eliminación de la sesión y la ausencia del historial asociado.

```
<!-- Botón para eliminar la sesión actual -->
<ion-button fill="clear" color="danger" (click)="deleteSession(selectedSessionId, $event)" slot="end">
  <ion-icon name="trash-outline"></ion-icon>
</ion-button>
```

```
// Elimina una sesión específica y actualiza la vista
deleteSession(sessionId: string, event: Event): void {
  event.stopPropagation();
  this.http.delete(`http://localhost:8000/sessions/${sessionId}`)
    .subscribe(response => {
      console.log('Session deleted successfully');
      // Volver a cargar la lista de sesiones para actualizar la vista
      this.loadSessions();
    }, error => {
      console.error('Error deleting session:', error);
    });
    this.selectedSessionId = "";
    this.onSessionSelect(this.selectedSessionId);
}
```

```
# Endpoint para eliminar una sesión específica
@app.delete("/sessions/{session_id}")
async def delete_session(session_id: str):
    # Construir el nombre del archivo basado en session_id
    file_path = os.path.join(SESSION_DIR, f"{session_id}.json")

    # Verificar si el archivo existe
    if not os.path.exists(file_path):
        raise HTTPException(status_code=404, detail=f"Session file '{session_id}.json' not found")

    try:
        # Intentar eliminar el archivo
        os.remove(file_path)
        return {"detail": f"Session '{session_id}' deleted successfully"}
    except Exception as e:
        # Manejar cualquier otro error que pueda ocurrir
        raise HTTPException(status_code=500, detail=f"Failed to delete session '{session_id}': {str(e)}")
```

Figura 6.6.6.4: Frontend y endpoint de delete sessions

6.6.7. Integración de ASR a chatbot

Para optimizar el uso de la aplicación por parte de estudiantes con discapacidades visuales, he integrado la funcionalidad de grabación de audio combinada con reconocimiento de voz a texto en la página d-studies. Esta integración facilita que los usuarios puedan dictar mensajes en lugar de escribirlos, mejorando así la accesibilidad y la experiencia de interacción con el modelo. Con esta implementación, el modelo demuestra ser adaptable a diversas partes de la aplicación, promoviendo una mayor inclusión y usabilidad para estudiantes con discapacidades visuales.

Antes de proceder con la integración, trasladé la llamada a la API del modelo de reconocimiento de voz a texto al backend, permitiendo así que los audios se almacenen en la carpeta audio_files. Además, creé un servicio speech de la funcionalidad en el frondend, que facilita el uso de la funcionalidad en la página d-studies. Esto asegura que el procesamiento del audio y la conversión a texto se gestionen de manera eficiente y segura, mejorando la funcionalidad y el rendimiento de la aplicación.

```python
class AudioRequest(BaseModel):
    audio: str
    name: str
@app.post("/asr/")
async def asr(request: AudioRequest):
    # Obtiene la cadena Base64 del campo `audio`
    audio_data = request.audio

    # Decodifica la cadena Base64 a bytes
    audio_bytes = base64.b64decode(audio_data)

    # Genera una ruta única para el archivo audio
    file_path = os.path.join(UPLOAD_DIRECTORY + "/" + AUDIO_DIR, request.name)

    # Guarda el archivo de audio en el directorio especificado
    with open(file_path, "wb") as audio_file:
        audio_file.write(audio_bytes)

    # Realiza una solicitud POST a la API de Hugging Face
    response = requests.post(
        "https://api-inference.huggingface.co/models/openai/whisper-large-v3",
        headers={
            "Authorization": "Bearer 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"
        },
        data = audio_bytes
    )
    # Verifica la respuesta de la API
    if response.status_code != 200:
        print(response.text)
        raise HTTPException(status_code=response.status_code, detail="Error calling Hugging Face API")

    result = response.json()
    return result
```

Figura 6.6.7.1: Endpoint del modelo ASR

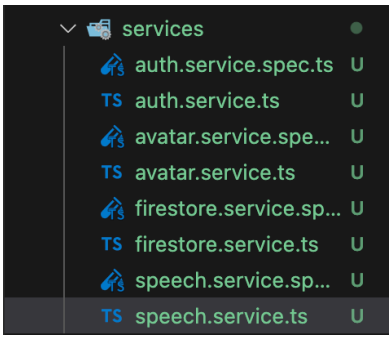


Figura 6.6.7.2: servicio de frontend

El servicio speech tiene las funciones de la grabación de audio y de la llamada al modelo ASR, donde presenta un observable de BehaviorSubject [6][8] para almacenar y emitir la lista de textos de speech.

```typescript
export class SpeechService {
  // BehaviorSubject para almacenar y emitir la lista de textos de speech
  private speechSource = new BehaviorSubject<{ name: string, text: string }[]>([]);
  // Observable para que otros componentes puedan suscribirse
  speech$ = this.speechSource.asObservable();
  recording = false;
  duration = 0;

  constructor(private http: HttpClient) { }

  async startRecording() {
    if (this.recording) {
      return;
    }
    this.recording = true;
    await VoiceRecorder.startRecording();
    this.calculateDuration();
  }

  async stopRecording() {
    if (!this.recording) {
      return;
    }
    const result: RecordingData = await VoiceRecorder.stopRecording();
    if (result.value && result.value.recordDataBase64) {
      const recordData = result.value.recordDataBase64;
      const time = new Date().getTime();
      const fileName = time + '.mp3';
      await Filesystem.writeFile({
        path: fileName,
        directory: Directory.Data,
        data: recordData
      });
      this.automaticSpeechRecognition(fileName);
    }
    this.recording = false;
  }
```

```typescript
  async automaticSpeechRecognition(fileName: string) {
    const audioFile = await Filesystem.readFile({
      path: fileName,
      directory: Directory.Data
    });
    const data = audioFile.data;

    const response = await fetch(
      "http://localhost:8000/asr",
      {
        method: "POST",
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          audio: data,
          name: fileName
        }),
      }
    );
    const result = await response.json();
      console.log(result);

    this.speechSource.next([{ name: fileName, text: result.text.toString() }]);
  }
```

Figura 6.6.7.3: speech.service.ts

En la página d-studies, es necesario importar el servicio de reconocimiento de voz (speech), utilizar las funciones de grabación de manera similar a como se hace en la página de audio, y suscribirse al observable speech$ del servicio. Esto permitirá obtener la última respuesta procesada por el modelo de reconocimiento automático de voz (ASR) y escribirla en la variable de entrada de mensaje, newMessage.

```
// Configura el gesto de "presión prolongada" para el botón de grabación
ngAfterViewInit(): void {
  const longpress = this.gestureCtrl.create({
    el: this.recordbtn.nativeElement,
    threshold: 0,
    gestureName: 'long-press',
    onStart: ev => {
      Haptics.impact({ style: ImpactStyle.Light })
      this.startRecording()
    },
    onEnd: ev => {
      Haptics.impact({ style: ImpactStyle.Light })
      this.stopRecording()
    },
  }, true)

  longpress.enable()
}

// Inicia la grabación de audio
async startRecording() {
  await this.speechService.startRecording();
}

// Detiene la grabación de audio
async stopRecording() {
  await this.speechService.stopRecording();
}
```

```
// Se suscribe al servicio speech
this.speechService.speech$.subscribe(data => {
  this.speech = data;
  if (this.speech.length > 0) {
    // Actualiza newMessage con el último elemento de la lista de speech
    this.newMessage = this.speech[this.speech.length - 1].text || '';
  }
});
```

```
<!-- Botón de grabación -->
<ion-button fill="clear" #recordbtn class="record-button">
  <ion-icon name="mic-outline" slot="icon-only"></ion-icon>
</ion-button>
```

Figura 6.6.7.4: Integración de servicio speech

6.7 MODELO TEXT GENERATION

En la página **studies**, también se ha integrado un modal accesible a través de un botón flotante ubicado en la parte inferior izquierda, diseñado para utilizar el modelo de generación de texto. Para implementar esta funcionalidad, he desarrollado un componente hijo específico para el modal. Este componente se utiliza en la página studies para facilitar la interacción con el modelo de generación de texto.

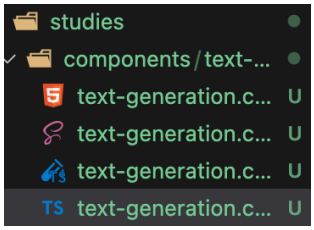


Figura 6.7.1: Componente text generation

De manera similar a los modelos anteriores, se ha implementado un endpoint en el backend que permite realizar una llamada a la API del modelo para generar textos basados en el contexto proporcionado. Los archivos generados se almacenan en el directorio uploaded_files, lo que facilita su acceso y uso en el chatbot destinado a los estudiantes.

```python
class TextRequest(BaseModel):
    texts: str
@app.post("/tg/")
async def tg(request: TextRequest):
    # Obtiene la cadena Base64 del campo `audio`
    text_data = request.texts

    # Realiza una solicitud POST a la API de Hugging Face
    response = requests.post(
        "https://api-inference.huggingface.co/models/google/gemma-7b",
        headers={
            "Authorization": "Bearer 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",
            "Content-Type": "application/json"
        },
        json={
            "inputs": text_data
        }
    )
    # Verifica la respuesta de la API
    if response.status_code != 200:
        print(response.text)
        raise HTTPException(status_code=response.status_code, detail="Error calling Hugging Face API")

    result = response.json()

    # Define la ruta del archivo
    file_name = "generated_text_" + str(uuid.uuid4())[0:8] + ".txt"
    file_path = os.path.join(UPLOAD_DIRECTORY + "/" + TEXT_DIR, file_name)


    # Guarda el texto en un archivo .txt
    with open(file_path, 'w') as file:
        file.write(result[0]["generated_text"])

    # Devuelve la respuesta
    return {"file_path": file_path, "generated_text": result[0].get('generated_text', '')}
```

Figura 6.7.2: Endpoint de text generation

El componente modal es bastante sencillo, permitiendo la entrada de texto contextual proporcionado por el usuario y la visualización de los textos generados por el modelo de generación de texto (TG). En consecuencia, se requiere implementar las siguientes funcionalidades:

1. **Abrir el Modal**: Integra una forma en la página studies para crear, presentar y abrir el modal.

```
// Método para abrir el modal
async openTextGenerationModal() {
  // Crea una instancia del modal
  const modal = await this.modalController.create({
    component: TextGenerationComponent, // El componente que se mostrará en el modal
  });
  // Presenta el modal en la pantalla
  return await modal.present();
}
```

```
    <!-- Botón flotante para abrir el modal de generación de texto -->
    <ion-fab vertical="bottom" horizontal="start" slot="fixed">
        <ion-fab-button (click)="openTextGenerationModal()">
            <ion-icon name = create-outline></ion-icon>
        </ion-fab-button>
    </ion-fab>
```

Figura 6.7.3: El botón y la función para crear y abrir el modal

2. **Utilizar el Modelo**: El componente de generación de texto maneja la interacción con el modelo TG para procesar el texto ingresado, generar respuestas, recibir y mostrar los resultados del modelo.
3. **Cerrar el Modal**: Implementa la funcionalidad para cerrar el modal en el componente de generación de texto.

```
@Component({
  selector: 'app-text-generation',
  templateUrl: './text-generation.component.html',
  styleUrls: ['./text-generation.component.scss'],
})
export class TextGenerationComponent  implements OnInit {

  textInput: string = '';
  generatedText: string = '';

  constructor(private modalController: ModalController) {}

  // Cierra el modal
  async dismiss() {
    await this.modalController.dismiss();
  }

  // LLamada API del modelo
  async tg(t: string) {
    const response = await fetch(
      "http://localhost:8000/tg",
      {
        headers: {
          "Content-Type": "application/json"
        },
        method: "POST",
        body: JSON.stringify({ texts: t }),
      }
    );
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    const result = await response.json();
    console.log(result);
    this.generatedText = result.generated_text;

  }
```

Figura 6.7.4: text-generation.componente.ts

```
<ion-header>
  <ion-toolbar>
    <ion-title>Text Generation</ion-title>
    <ion-buttons slot="end">
      <ion-button (click)="dismiss()">Close</ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <ion-item>
    <ion-textarea
      [(ngModel)]="textInput"
      auto-grow="true"
      rows="1"
      placeholder="Type a context"
    ></ion-textarea>
  </ion-item>
  <ion-button expand="full" (click)="tg(this.textInput)">send</ion-button>

  <ion-card *ngIf="generatedText" class="generated-text-card">
    <ion-card-header>
      <ion-card-title>Generated Text</ion-card-title>
    </ion-card-header>
    <ion-card-content>
      <p>{{ generatedText }}</p>
    </ion-card-content>
  </ion-card>
</ion-content>
```

Figura 6.7.5: text-generation.component.html

CAPÍTULO 7: PRUEBAS

En este capítulo, se resumirán las pruebas realizadas para verificar el correcto funcionamiento del prototipo, así como los resultados obtenidos de dichas pruebas.

7.1 CONEXIÓN DE FIREBASE

En primer lugar, se realice la prueba de inicio de sesión con una cuenta previamente registrada, comprobándose que el proceso funciona de manera óptima.

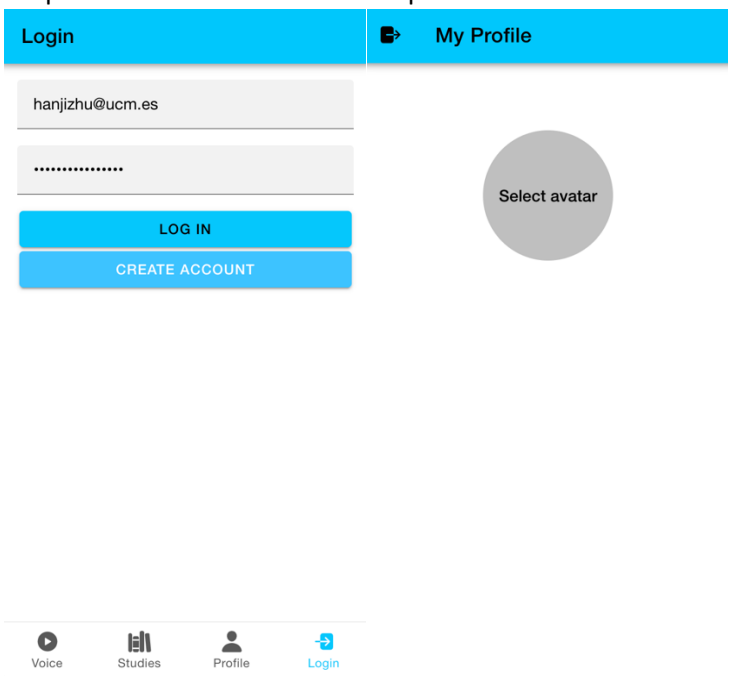


Figura 7.1.1: Login and Profile page

Posteriormente, se efectuó la prueba de creación de una nueva cuenta, empleando una dirección de correo electrónico y una contraseña diferentes. Se verificó que la información de la cuenta se almacena correctamente en el sistema de autenticación de Firebase [5][38].

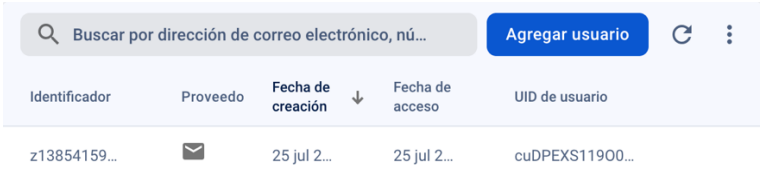


Figura 7.1.2: New user

Finalmente, se realizó una prueba de carga de un avatar para comprobar la correcta conexión con FireStorage [26], confirmándose que la imagen se almacena adecuadamente en FireStorage [5][26].

Figura 7.1.3: Avatar Data

De este modo, se ha verificado que la funcionalidad funciona correctamente y establece una conexión adecuada con Firebase [38].

7.2 GRABADORA DE AUDIO Y MODELO DE RECONOCIMIENTO DE VOZ

En esta sección, probamos el funcionamiento de grabar el audio y el modelo de reconocimiento de voz. Ambas funciones se integraron en una sola, permitiendo que el proceso se realice de manera simultánea con una única llamada. Como se puede observar en la imagen siguiente, el funcionamiento es perfecto en diversos idiomas, con posibilidad de recordar, reproducir y eliminar el audio.

Figura 7.2.1: voice tab

7.3 EXPLORADOR DE ARCHIVOS

El explorador de archivos ofrece las funciones de subir archivos a la página de studies, crear carpetas con una URL específica y descargar los archivos subidos para su visualización y conformidad por parte de los usuarios. En las imágenes siguientes, se puede observar el correcto funcionamiento de cada una de estas funciones.

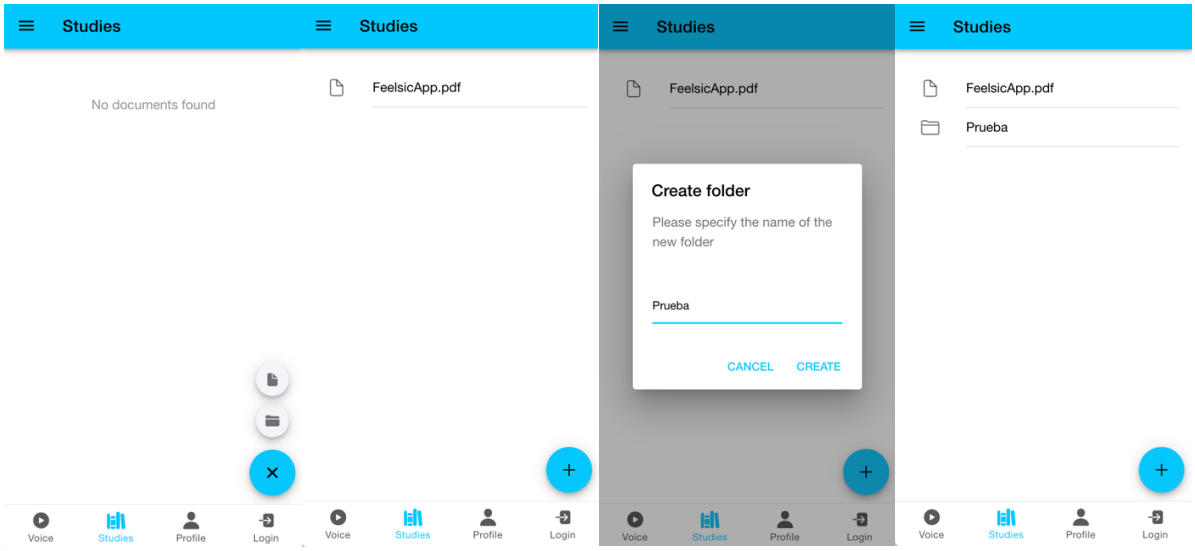


Figura 7.3.1: studies tab

En la imagen anterior, se pueden observar dos opciones en la esquina inferior derecha: subir archivos y crear carpetas. Estas acciones se pueden realizar de manera perfecta. Además, la descarga de los archivos se lleva a cabo con un solo clic, guardando el archivo en el dispositivo local.

Se genera correctamente la URL de la carpeta en la página studies y se confirma que los archivos se almacenan adecuadamente en Firebase [38], tanto en el Firestore [26] como en la colección de la base de datos.

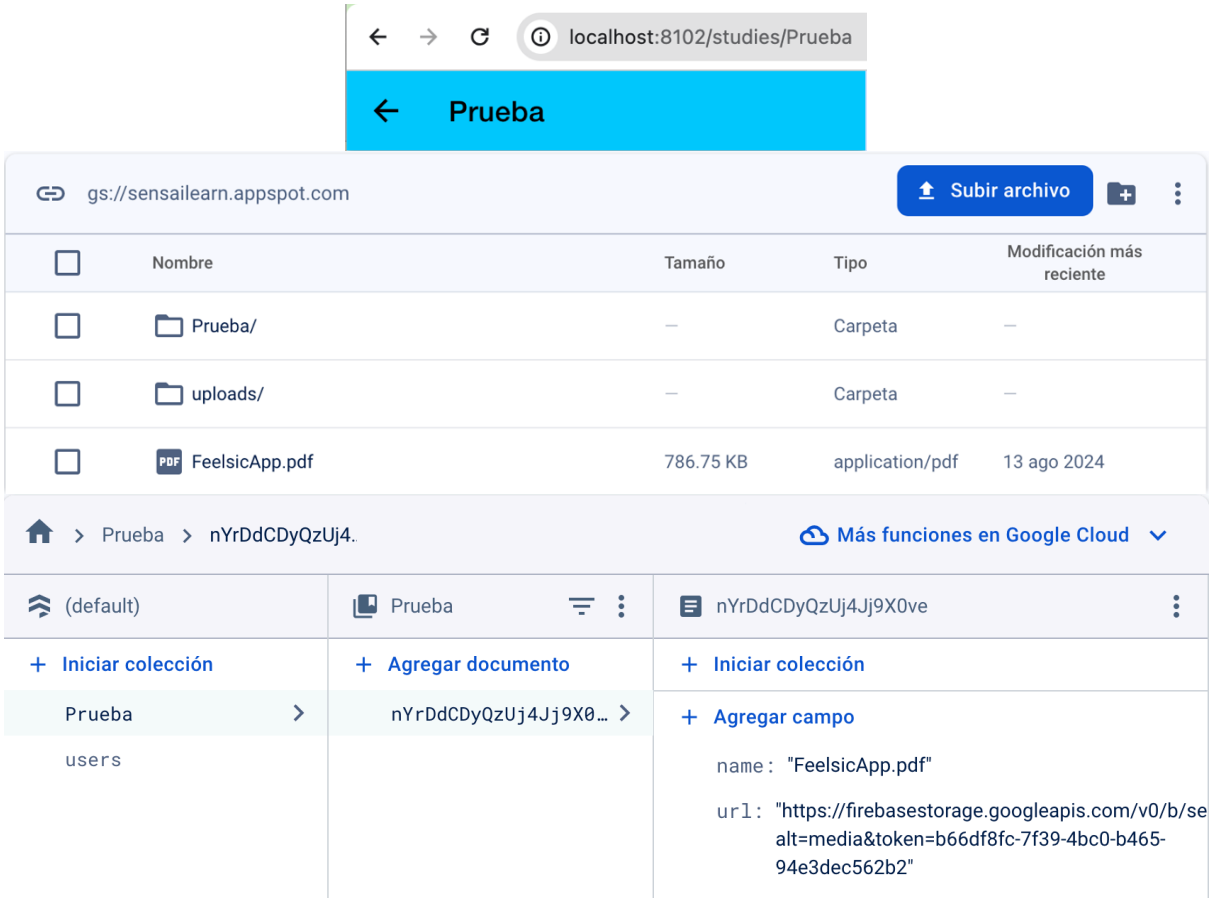


Figura 7.3.2: URL and Data

Asimismo, la eliminación y la copia de los documentos en otras carpetas se realizan perfectamente.

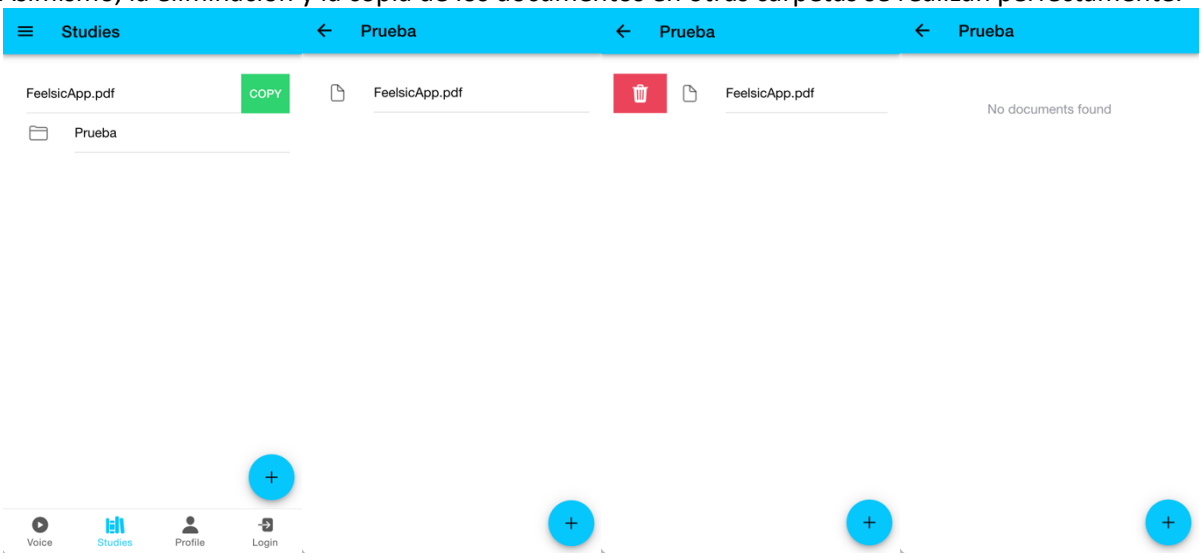


Figura 7.3.3: delete y copy

7.4 MODELO DE RECONOCIMIENTO DE DOCUMENTOS

En esta sección, se prueba la conexión de los documentos gestionados en la página de "studies" y la integración del modelo de reconocimiento de documentos. Comprobando que los documentos almacenados en el explorador de archivos de la página "studies" pueden ser utilizados como entrada para el modelo de reconocimiento de documentos y el modelo procesa la solicitud y devuelve la respuesta de manera inmediata y precisa.

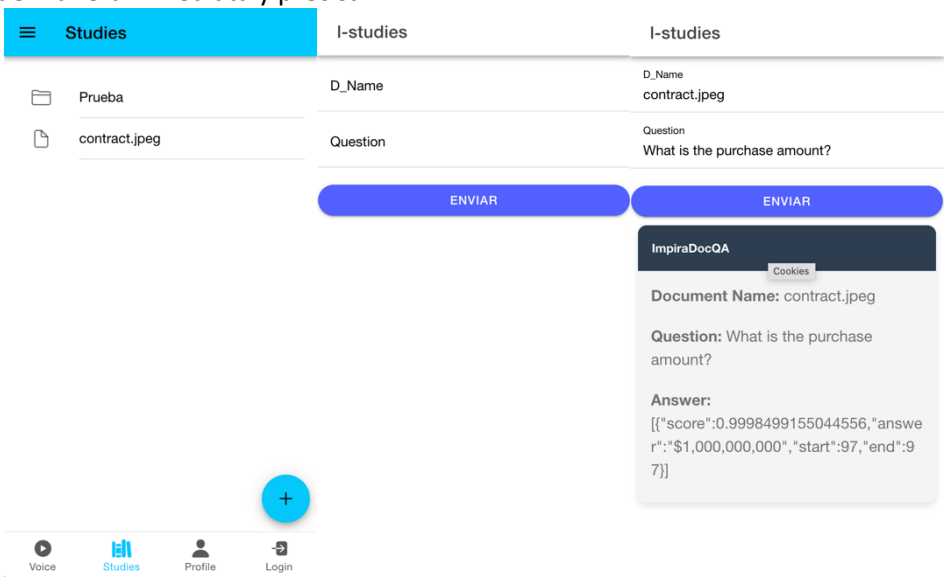


Figura 7.4.1: lstudies page

7.5 CHATBOT

El chatbot opera de manera perfecta, con la funcionalidad completa para la selección de documentos que permite aportar al chatbot a analizar el contenido de los documentos seleccionados. Las funciones para la selección, recuperación y eliminación de sesiones también funcionan correctamente. Además, la integración del modelo de Reconocimiento Automático de Voz (ASR) en la aplicación ha sido de manera satisfactoria.

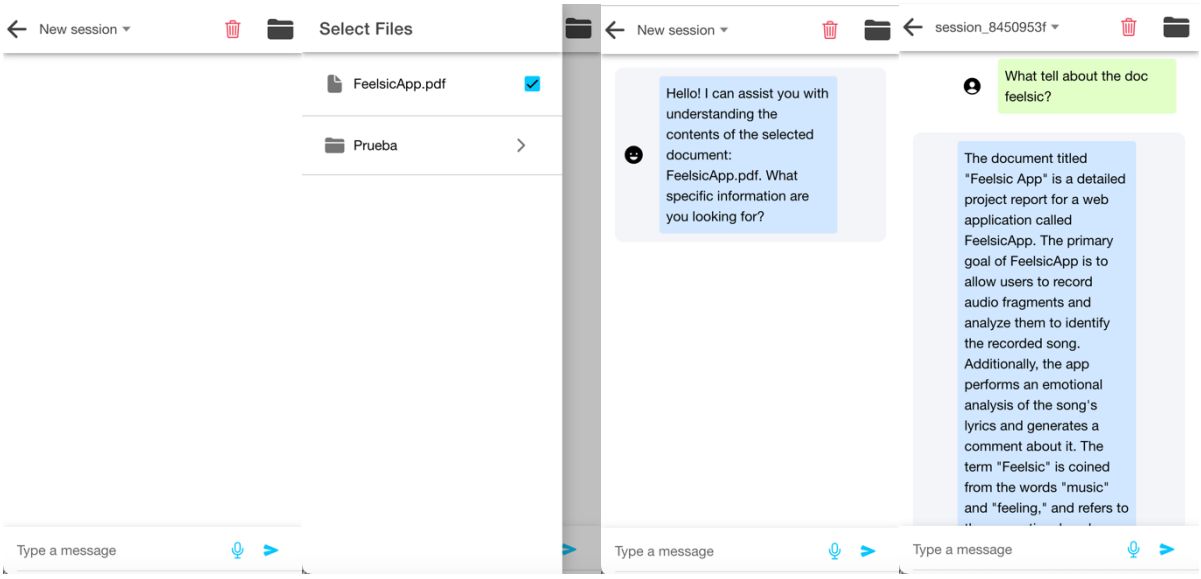


Figura 7.5.1: prueba de file selector

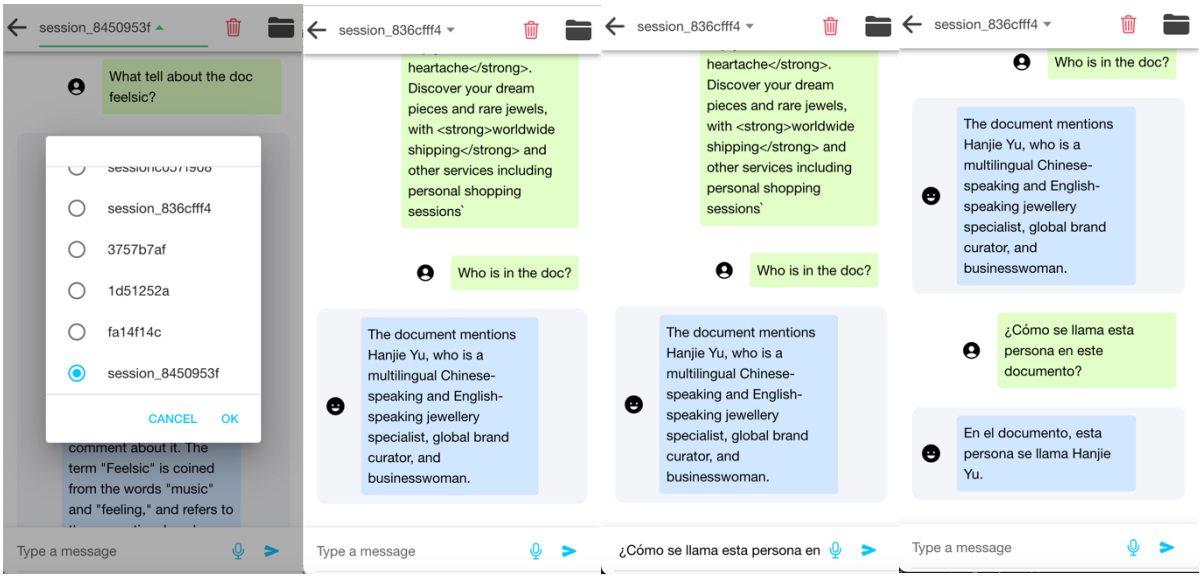


Figura 7.5.2: prueba de session selector y ASR

Las sesiones se almacenan correctamente en el archivo sessions_file, mientras que los archivos de audio se guardan de manera adecuada en el directorio audio_files.

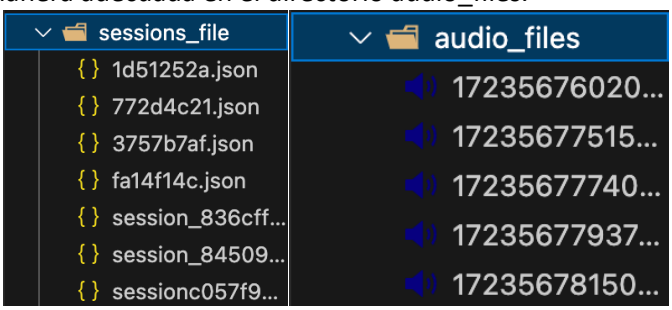


Figura 7.5.3: sessions_file y audio_files

7.6 MODELO DE GENERACIÓN DE TEXTOS

En la página studies, la funcionalidad de generación de textos, accesible a través del botón implementado, opera correctamente permitiendo proporcionar un contexto y generar textos basados en dicho contexto. Además, los textos generados se guardan correctamente en el archivo saved_texts.

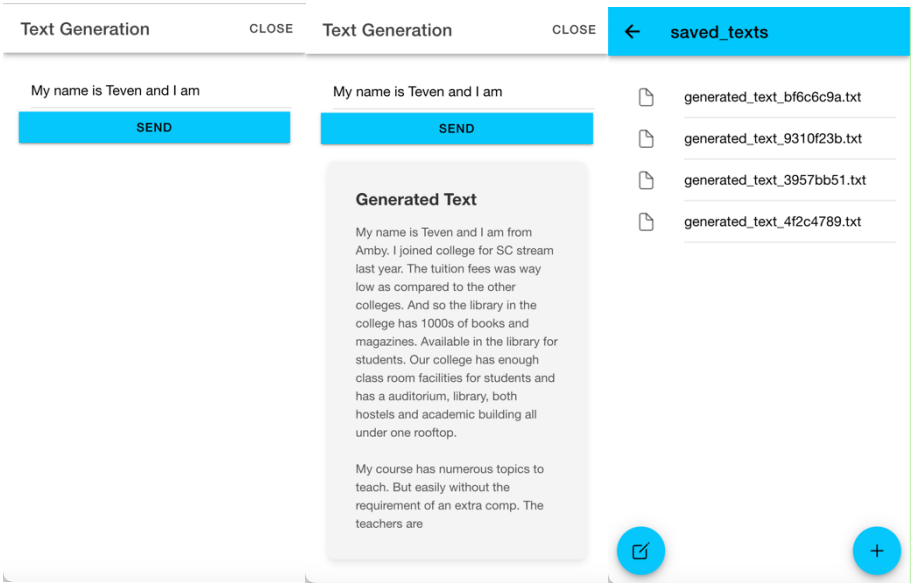


Figura 7.6.1: text generation y saved_text

CAPÍTULO 8: DISCUSIÓN

En este capítulo se presentan y evalúan los resultados obtenidos de la aplicación desarrollada, valorando su efectividad frente al diseño funcional planteado. Se examinan tanto las ventajas y desventajas, así como las limitaciones del enfoque adoptado. Además, se ofrecerá una perspectiva sobre los posibles avances futuros.

En la primera sección del capítulo anterior, se realizó la prueba de conexión con Firebase [38], así como la autenticación, gestión y almacenamiento de datos. Durante el desarrollo, estas funciones fueron probadas en múltiples ocasiones, demostrando un rendimiento eficiente en la autenticación de usuarios y en la gestión de la información, como colecciones. No obstante, en el caso de la gestión y almacenamiento de datos, como documentos en Firebase [38], se observó una latencia significativa en la recuperación de los archivos. Por esta razón, se optó por almacenar los documentos en el directorio del backend en lugar de en Firebase [38]. De este modo, el uso de Firebase [38] se ha limitado a la autenticación de cuentas, la gestión de colecciones de información de usuarios y el almacenamiento de datos de cuentas de usuario. Por consiguiente, se identifica un posible avance en esta área para optimizar el servicio de almacenamiento de documentos en Firebase [38], abordando la latencia mediante la adquisición de un nivel de servicio superior en Firebase [38] o explorando alternativas de desarrollo más complejas para reducir dicha latencia.

El grabador de audio integrado en Capacitor de Ionic [7][39][40] ha demostrado un rendimiento excepcional, capturando el audio con una claridad comparable a la de los sistemas de grabación de alta calidad en computadores. Esta herramienta facilita el uso de audio en el desarrollo al permitir la grabación en formato base64, con la opción de convertirlo fácilmente a otros formatos de audio. Gracias a esto, el modelo de reconocimiento de voz presenta una alta precisión en la conversión de audio a texto, con una tasa de error menor. Este modelo de reconocimiento de voz, similar al servicio Google Speech-to-Text [1], transforma inteligentemente los audios en textos y soporta una amplia gama de idiomas y dialectos. Sin embargo, la integración utilizando la comunidad HuggingFace [29] presenta latencia y atascos en la transmisión de datos al servicio. Esta funcionalidad podría mejorarse sustancialmente al asegurar una fluidez mediante la adquisición de un nivel de servicio superior y un desarrollo más adecuado y completo para la integración del modelo Whisper [33].

El modelo de reconocimiento de documentos, impira/layoutlm-document-qa [17][18], ofrece una conexión superior dentro de la comunidad de HuggingFace [29], lo que se traduce en un tiempo de ejecución significativamente menor en comparación con el modelo Whisper [33] en la misma plataforma. Sin embargo, presenta ciertas limitaciones en su aplicación, ya que su funcionalidad depende de la categorización de información a través de tecnología OCR (Reconocimiento Óptico de Caracteres). Este modelo permite realizar consultas sobre el documento visual una vez que la información ha sido categorizada por inteligencia artificial, pero su capacidad se ve restringida cuando se trata de preguntas sobre datos específicos que no han sido previamente categorizados.

El modelo resulta más eficaz en documentos con un formato visual estructurado. Para superar esta limitación, sería beneficioso avanzar en las tecnologías OCR y de reconocimiento de texto a partir de imágenes. Mejorar estas tecnologías permitiría una identificación más precisa de los contenidos en documentos presentados en formato de imagen, facilitando su conversión perfecta a texto editable. Esto ampliaría significativamente las posibilidades de uso y efectividad del modelo en cuestión. Por lo tanto, se vislumbra un avance potencial en el entrenamiento de este tipo de modelo, con el objetivo de mejorar su capacidad para categorizar y reconocer una mayor variedad de textos en imágenes, incluso en diferentes calidades o entornos de las imágenes.

El chatbot desarrollado mitiga la limitación del modelo anterior al ser capaz de reconocer todos los contenidos de texto proporcionados. Sin embargo, su funcionalidad no se extiende a documentos en formato de imagen, sino únicamente a aquellos en formato de texto plano. Un posible avance en esta área sería la integración de tecnología avanzada de OCR para lograr un modelo que no solo pueda leer y comprender los contenidos textuales, sino también interpretar la categorización implementada por la tecnología OCR. Este modelo debería ser entrenado para transformar la información categorizada en fragmentos de texto explicativos, reflejando tanto las informaciones categorizadas como los contenidos leídos en el documento.

De este modo, el chatbot no solo comprendería y proporcionaría información de textos puros, sino que también sería capaz de interpretar los textos presentados en las imágenes de los documentos. Esto permitiría extraer más información de dichos documentos, ampliando significativamente la utilidad y eficacia del chatbot en el procesamiento y análisis de diversos formatos documentales.

Así, se puede observar que el chatbot presenta una limitación cuando se enfrenta a documentos que contienen imágenes. En tales casos, el chatbot no tiene la capacidad para reconocer ni procesar las imágenes, lo que conduce a la pérdida de información significativa contenida en el documento. No obstante, el chatbot demuestra una notable capacidad para capturar y procesar información en documentos que están compuestos exclusivamente de texto.

En cuanto al modelo de generación de texto, la calidad de los textos producidos varía significativamente según los contextos aportados; en algunos casos, la calidad es alta, mientras que en otros es baja. Esta inconsistencia podría deberse a la presencia de datos ruidosos durante el entrenamiento del modelo. Un posible avance en esta área sería implementar un filtro para eliminar los ruidos de los datos y entrenar el modelo con un conjunto de datos de mayor calidad. Además, sería beneficioso incorporar un mecanismo que permita dirigir las instrucciones del modelo hacia un contexto específico, evitando así que los textos generados se desvíen del tema deseado por los usuarios.

La aplicación también exhibe una debilidad en términos de su interfaz y carece de adaptaciones suficientes para estudiantes con discapacidades visuales. La mejora de la interfaz y la adecuación de la aplicación para satisfacer las necesidades de estos estudiantes requerirá una investigación exhaustiva para identificar las deficiencias actuales y desarrollar soluciones efectivas. Este proceso de optimización demandará un esfuerzo considerable y un tiempo significativo para garantizar una conformidad completa con los requisitos de accesibilidad para usuarios con discapacidades visuales.

En el capítulo siguiente, se llevará a cabo un análisis y una evaluación de la aplicación desde una perspectiva alternativa, teniendo en cuenta las limitaciones identificadas. Asimismo, se formulará un plan para abordar estas deficiencias y se delinearán las posibles direcciones futuras del proyecto con el fin de mitigar las limitaciones detectadas. Donde concluirá con una reflexión final sobre el proyecto desde mi punto de vista.

CAPÍTULO 9: CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentan las conclusiones derivadas de los resultados obtenidos. Además, se proponen diversas líneas de trabajo futuras con el objetivo de ampliar el alcance del proyecto, mejorar su rendimiento y añadir nuevas funcionalidades.

9.1 CONCLUSIONES

En primer lugar, podemos concluir que nuestro proyecto ha sido exitoso en cuanto al diseño y desarrollo de una aplicación prototipo para la educación de estudiantes con discapacidades visuales y estudiantes sin discapacidades. El funcionamiento de cada componente de la aplicación, incluyendo Firebase [38], el reconocimiento de audio, el reconocimiento de documentos visuales, el chatbot que proporciona información detallada a partir del contenido de los documentos y la generación del texto a partir de un contexto proporcionado, ha funcionado de manera satisfactoria. Además, la aplicación ofrece diversos servicios que mejoran su usabilidad para los usuarios y funcionan exitosamente. Entre estos servicios se encuentran el almacenamiento seguro de la información de los usuarios, las interfaces de la aplicación accesible para interactuar con los modelos y el chatbot, y la capacidad de almacenar los documentos proporcionados por los usuarios.

Sin embargo, también es posible identificar ciertas limitaciones en nuestro prototipo actual que podrían afectar su escalabilidad. Una de las principales limitaciones es el retardo en el tiempo de ejecución de los modelos de HuggingFace [29]. Para mitigar esta desventaja, sería necesario adquirir servicios premium directamente en la página web oficial de los servicios de modelos, en lugar de utilizar los modelos presentados en HuggingFace [29]. Aunque para este proyecto específico no es esencial realizar esta mejora, se reconoce que hay margen para la optimización en esta área. Esta mejora sería especialmente relevante en el contexto de un proyecto empresarial con un presupuesto adecuado.

Otra limitación se encuentra en el entorno de desarrollo. Debido a la falta de acceso a dispositivos variados, el entorno de desarrollo Ionic [39] y a la restricción de tiempo, no pudimos desarrollar el control de la interfaz mediante entrada de voz en el entorno de Ionic [39]. Con un entorno de desarrollo más diversificado, podría haber logrado que la aplicación fuera más interactiva y adaptable para estudiantes con discapacidades visuales. No obstante, si la sociedad o grandes empresas dedicaran más tiempo al desarrollo de aplicaciones adaptables para grupos con necesidades especiales, se podría mejorar la igualdad social. Esto también serviría como ejemplo para desarrolladores interesados en esta área, proporcionando más recursos y ejemplos a aquellos con limitaciones en su entorno de desarrollo.

Finalmente, la limitación en el uso del servicio de Firebase [38] nos llevó a restringir la aplicación del proyecto únicamente a la autenticación, gestión y almacenamiento de datos. No obstante, se podría ampliar significativamente la utilización de Firebase [38] para abarcar más funcionalidades. Por ejemplo, se podría emplear Firebase [38] para el entrenamiento básico de modelos de reconocimiento de documentos, utilizando documentos de diversas áreas del conocimiento (pero necesitará de comprar los servicios de Firebase [38], para poder almacenar tantos documentos de diversas áreas de conocimientos). Esto permitiría entrenar distintos tipos de modelos predeterminados, entrenados para diferentes disciplinas, mejorando así la utilidad y efectividad de la aplicación para los estudiantes.

En conclusión, nuestro proyecto representa un excelente punto de partida para el desarrollo de aplicaciones destinadas a estudiantes con discapacidades visuales, así como a estudiantes en general. Aunque la aplicación presenta varias limitaciones debido a costos, tiempo y la falta de conocimiento

específico sobre las necesidades de personas con discapacidades visuales, ha logrado cumplir su propósito como prototipo. Este prototipo ha sido eficaz para la prueba, comprensión y aplicación de los modelos de procesamiento de lenguaje natural (NLP) de HuggingFace [29] y del chatbot en el contexto educativo. A pesar de las restricciones, la aplicación ha demostrado ser una herramienta útil tanto para estudiantes con discapacidades visuales como para aquellos sin ellas. Estoy muy satisfechos con los resultados obtenidos y creemos firmemente que este trabajo puede tener aplicaciones muy interesantes en el campo de la educación inclusiva. Este proyecto no solo sirve como ejemplo de cómo desarrollar aplicaciones educativas adaptadas a diversas necesidades, sino que también destaca el potencial de la tecnología para mejorar la accesibilidad y la igualdad en la educación.

Los archivos de códigos de la aplicación desarrollada son accesibles en las direcciones: [https://drive.google.com/drive/folders/1sz61uN8qqo-bKNf2bNZiXxV2rkz1kgQF?usp=share_link].

9.2 Trabajo Futuro

En la sección de trabajo futuro, se evaluarán las posibles mejoras y adaptaciones del prototipo actual para satisfacer de manera más efectiva las necesidades de los usuarios.

Una de las potenciales evoluciones es el desarrollo del control de la interfaz de la aplicación mediante la voz proporcionados por el usuario. Esto podría lograrse mediante el reconocimiento y la categorización de las voces, permitiendo que las acciones y operaciones en la aplicación se realicen a través de la conversión de la voz en texto usando el modelo de reconocimiento de voz y la categorización de dicho texto para el control de la interfaz de la aplicación. Sin embargo, este enfoque podría presentar varios desafíos, ya que la categorización manual de los textos puede dar lugar a errores en la ejecución de las acciones. Para mitigar estos problemas, sería necesario entrenar un modelo de categorización basado en inteligencia artificial, optimizando su precisión hasta alcanzar una tasa de error aceptablemente baja. Esta tarea representaría un proyecto significativo grande.

Otra posible evolución del prototipo sería el entrenamiento del modelo de chatbot utilizando grandes cantidades de documentos almacenados en Firebase [38], que abarcan diversas áreas del conocimiento. Este proceso implicaría la categorización de los documentos según su área temática y la incorporación de estos datos en el modelo, permitiendo el entrenamiento específico de diferentes chatbots para cada área del conocimiento. De este modo, se podrían ofrecer chatbots prediseñados y entrenados con un amplio conjunto de datos documentales para cada área temática, eliminando la necesidad de que los estudiantes busquen y realicen el entrenamiento desde cero. Esta mejora proporcionaría una mayor facilidad de uso y accesibilidad para los estudiantes que no disponen de documentos en una determinada área de conocimiento, optimizando así la experiencia educativa.

Asimismo, en el futuro se podría explorar la posibilidad de colaborar con grandes empresas para optimizar el uso de las funcionalidades desarrolladas, contando con un presupuesto adecuado y un plazo de desarrollo extendido. Esta colaboración facilitaría la mejora del tiempo de respuesta de los modelos y la perfección de las interfaces de usuario, lo que resultaría en una mayor eficiencia y usabilidad de la aplicación. De esta manera, se podría considerar la evolución de la aplicación prototipo hacia una solución comercial completa.

CHAPTER 10: INTRUDUCTION

At present, there is a notable shortage of truly useful educational applications for students with visual impairments or for students without disabilities. This lack reflects a stagnation in the development of technological tools tailored to their specific needs, despite advances in other areas of digital education. It is crucial to focus on and advance the development of technologies and applications that use advanced models to support the study and learning of these students, who face unique challenges in accessing educational information.

The creation of innovative technology that integrates various advanced models could represent a significant market that has yet to be fully explored or developed. Improving the accessibility and adaptability of educational applications for visually impaired individuals or students with other difficulties would not only contribute to facilitating their learning but also to promoting a more inclusive and equitable society. This approach aims not only to equalize educational opportunities but also to enrich the educational experience of all students, regardless of their abilities.

To achieve true equality in an advanced society, it is essential to prioritize addressing the specific challenges faced by people with disabilities. This implies not only developing and launching new educational applications aimed exclusively at students without disabilities but also designing adaptive and accessible solutions that meet the unique needs of visually impaired students. Unfortunately, the lack of attention and resources dedicated to this issue can perpetuate discrimination and exclusion in today's society.

It is imperative that society takes seriously the development of inclusive technologies that promote equal access and opportunities for all individuals, regardless of their physical abilities. By doing so, we not only move towards a more just and equitable society but also harness the potential and capabilities of all members of our global community.

The development of new technologies oriented towards people with disabilities can be a complex and costly task, especially when attempting to address all the specific needs of this diverse group. Given that most developers do not have disabilities, fully understanding these needs can be a considerable challenge. However, there are several tools on the market that already cover the basic needs for the study of people with disabilities, which can serve as a solid foundation in the development of new applications.

Close collaboration with visually impaired students in the creation and testing of educational applications can provide valuable insights and understanding of their true needs in the educational field. This incremental approach allows us to gradually advance toward the creation of more comprehensive educational applications specifically tailored for visually impaired students.

In this context, there is a clear need to innovate and develop new educational applications that are truly revolutionary for visually impaired students. These applications not only have the potential to significantly improve these students' educational experience but also to optimize their study time and associated costs. By providing tools that facilitate access to information and effectively support learning, we can contribute to promoting equal opportunities and the academic development of visually impaired individuals.

It is essential that these new technologies be designed with a user-centered approach, ensuring that they effectively meet the specific and diverse needs of visually impaired students. By doing so, we are not only advancing in terms of inclusion and accessibility but also strengthening our collective ability

to support the learning and personal development of all individuals, regardless of their physical abilities.

10.1 MOTIVATION

The underlying motivation for our project stems from the urgent need to address the challenges faced by visually impaired students, as mentioned earlier. Our primary goal is to develop a system that is not only sustainable and efficient but also easy to implement, with the purpose of significantly improving the educational experience of these students, making it more accessible and equitable. This involves reducing the barriers and costs associated with studying for this specific group, with the ultimate goal of equalizing their educational access with that of their peers without visual impairments.

Our vision for the application includes the implementation of a highly scalable study system designed to adapt to various modalities and areas of study. This approach will allow us to offer an accessible and effective solution for students with different needs and learning styles. Scalability is a crucial component for achieving efficiency and cost-effectiveness in providing educational tools, ensuring that our solution can be widely adopted not only by visually impaired students but by a broader audience.

By focusing on sustainability and adaptability, we aim to overcome the current limitations in access to education for visually impaired individuals. Our commitment lies in creating a more inclusive and equitable educational environment where all people, regardless of their physical abilities, can reach their full academic potential.

10.2 OBJECTIVES

The primary objective is to design and develop an application that integrates several components aimed at improving the study process.

1. The design of an application that offers efficient user data management and account administration like applications currently available on the market is pursued. This approach will ensure that users can securely manage their data and make adjustments to their accounts intuitively, thus guaranteeing a smooth and reliable user experience.
2. A solution that facilitates and optimizes the use of the application for visually impaired students is sought. This approach is designed to significantly improve the accessibility and user experience of this specific group of students.
3. The development will include a solution for simplified content search within a document based on a question provided by the student. This will allow for more efficient and effective interaction with educational materials, enhancing users' ability to quickly obtain relevant information.
4. A scalable and customizable solution that can adapt to various areas of study and meet the specific needs of each user is intended to be provided. This approach will ensure that the application is versatile and capable of responding to different educational modalities and individual requirements, thus promoting a more effective and personalized learning experience.

5. The implementation of a solution that optimizes the writing of reports and documents, allowing students to save considerable time in the writing process and thus facilitating the preparation of texts more efficiently and effectively.

10.3 Work Plan

To achieve the stated objectives, the following tasks have been executed:

1. Development of an application using Ionic [39], which incorporates an integrated system for managing, storing, and monitoring data based on Google's Firebase [38].
2. Implementation of an advanced audio recognition model designed to execute specific actions through voice commands, with the purpose of facilitating the use of the application for visually impaired students.
3. Integration of a document recognition system that provides the necessary information efficiently, thus improving the accessibility and utility of academic content for students.
4. Development of a chatbot tailored to the students' area of study. This chatbot, once trained with the documents provided by the students themselves, can extract and supply specific information contained in these documents when students need to search for specific data.
5. Incorporation of a text generation model designed to increase efficiency in the preparation of reports and other academic documents, optimizing the time and effort dedicated to these tasks by students.

10.4 Structure of the Report

This report has been organized to provide a comprehensive and understandable explanation of the procedures and underlying foundations in the development of the project. The structure of the report is articulated in the following chapters, each addressing specific aspects of the project in detail:

- **Chapter 1: Introduction**. This chapter presents the underlying motivation and fundamental objectives that guide the development of this project. It also provides a detailed description of the work plan adopted and gives an overview of the structure of the report.
- **Chapter 2: State of the art**. This chapter performs an analysis of various tools and software that provide functionalities analogous to those implemented in this project, offering a brief description of each tool identified during the research process.
- **Chapter 3: Functional Design**. This chapter describes the initial design conception of the project, detailing the initial designs.
- **Chapter 4: Technologies used**. This chapter offers a detailed description of the technologies and tools employed in the development of the project.
- **Chapter 5: Architecture and data model**. This chapter details the technical structure of the project, describing the overall architecture and the data models implemented.
- **Chapter 6: Project Development**. It offers a comprehensive summary of the development of the application's functionalities.

- **Chapter 7: Testing**. Where the testing of the functionalities developed during the development chapter is described.
- **Chapter 8: Discussion**. Where the obtained results are discussed, highlighting their value, identifying weaknesses, and pointing out where further progress could be made.
- **Chapter 9: Conclusions and future work**. The objectives achieved are discussed, and the potential impact of the developed application in the intended context is evaluated. Additionally, areas for future improvements and continuous development are identified, exploring possibilities and recommendations for expanding or refining the project in the future.
- **Chapter 10 and 11:** Translations into English of Chapter 1 and Chapter 9, respectively.
- **Appendix I: Requirements specification**. This chapter presents the specification of the requirements necessary for the development of the project.
- **Appendix II: User manual**. This appendix provides a user manual for the developed tool.

Each chapter is structured to provide a comprehensive and systematic view of the project, from its conception to its implementation and evaluation, thus ensuring complete and rigorous documentation of the work performed.

CHAPTER 11: CONCLUSIONS AND FUTURE WORK

This chapter presents the conclusions derived from the results obtained. Additionally, various future work directions are proposed with the aim of expanding the project's scope, improving its performance, and adding new functionalities.

11.1 CONCLUSIONS

First, we can conclude that our project has been successful in the design and development of a prototype application for the education of both visually impaired and non-disabled students. The functioning of each component of the application, including Firebase [38], audio recognition, visual document recognition, the chatbot that provides detailed information from document content, and text generation based on provided context, has been satisfactory. Additionally, the application offers various services that enhance its usability for users and operate successfully. These services include secure storage of user information, accessible application interfaces for interacting with the models and chatbot, and the ability to store user-provided documents.

However, certain limitations in our current prototype can be identified that may affect its scalability. One of the main limitations is the delay in execution time of the HuggingFace models [29]. To mitigate this drawback, it would be necessary to acquire premium services directly from the official model service website, rather than using the models available on HuggingFace [29]. Although this improvement is not essential for this specific project, it is acknowledged that there is room for optimization in this area. This enhancement would be particularly relevant in the context of a commercial project with an appropriate budget.

Another limitation lies in the development environment. Due to the lack of access to a variety of devices, the Ionic [39] development environment, and time constraints, we were unable to develop voice input interface control within the Ionic environment [39]. With a more diversified development environment, the application could have been made more interactive and adaptable for visually impaired students. Nevertheless, if society or large companies dedicated more time to developing adaptable applications for groups with special needs, social equality could be improved. This would also serve as an example for developers interested in this area, providing more resources and examples to those with limitations in their development environment.

Finally, the limitation in the use of the Firebase service [38] led us to restrict the project's application solely to authentication, data management, and storage. However, Firebase's [38] use could be significantly expanded to cover more functionalities. For example, Firebase [38] could be used for basic training of document recognition models using documents from various fields of knowledge (although purchasing Firebase services [38] would be necessary to store such a large number of documents from different knowledge areas). This would allow training of different types of pre-designed models, tailored for various disciplines, thereby improving the application's utility and effectiveness for students.

In conclusion, our project represents an excellent starting point for the development of applications aimed at visually impaired students as well as students in general. Although the application presents several limitations due to costs, time, and a lack of specific knowledge about the needs of visually impaired individuals, it has successfully fulfilled its purpose as a prototype. This prototype has been effective in testing, understanding, and applying HuggingFace's [29] natural language processing (NLP) models and the chatbot in an educational context. Despite the constraints, the application has proven to be a useful tool for both visually impaired students and those without disabilities. We are very satisfied with the results obtained and firmly believe that this work could have significant applications

in the field of inclusive education. This project not only serves as an example of how to develop educational applications adapted to diverse needs but also highlights the potential of technology to improve accessibility and equality in education.

The code files for the developed application are accessible at the following links:
[https://drive.google.com/drive/folders/1sz61uN8qqo-bKNf2bNZiXxV2rkz1kgQF?usp=share_link].

11.2 Future Work

In the future work section, potential improvements and adaptations of the current prototype to more effectively meet users' needs will be evaluated.

One possible evolution is the development of voice-controlled application interfaces. This could be achieved through voice recognition and categorization, allowing actions and operations within the application to be carried out by converting voice to text using a speech recognition model and categorizing that text to control the application interface. However, this approach may present several challenges, as manual text categorization could lead to errors in action execution. To mitigate these issues, it would be necessary to train a categorization model based on artificial intelligence, optimizing its accuracy until an acceptably low error rate is achieved. This task would represent a significant and large-scale project.

Another potential evolution of the prototype would be training the chatbot model using large amounts of documents stored in Firebase [38], covering various fields of knowledge. This process would involve categorizing documents by subject area and incorporating this data into the model, enabling the specific training of different chatbots for each area of knowledge. In this way, pre-designed and trained chatbots could be offered with a comprehensive set of documentary data for each subject area, eliminating the need for students to search for and train the models from scratch. This improvement would provide greater ease of use and accessibility for students who do not have documents in a specific area of knowledge, thereby optimizing the educational experience.

Furthermore, in the future, the possibility of collaborating with large companies to optimize the use of developed functionalities, with an adequate budget and extended development timeframe, could be explored. This collaboration would facilitate the improvement of model response times and the refinement of user interfaces, resulting in greater efficiency and usability of the application. Thus, the evolution of the prototype application into a full commercial solution could be considered.

BIBLIOGRAFÍA Y REFERENCIA

[1] Google, "Google Cloud Speech-to-Text: Convierte voz en texto con la IA de Google," [Online]. Available: https://cloud.google.com/speech-to-text?hl=es_419. [Accessed Aug 2023].

[2] Google, "Google Cloud Document AI," [Online]. Available: https://cloud.google.com/document-ai?hl=es. [Accessed Aug 2023].

[3] Doclime, "Doclime: Chat with any PDF," [Online]. Available: https://doclime.com. [Accessed Aug 2023].

[4] OpenAI, "ChatGPT," [Online]. Available: https://chatgpt.com. [Accessed Aug 2023].

[5] Firebase, "Firebase Documentation," [Online]. Available: https://firebase.google.com/docs. [Accessed Mar 2023].

[6] I. Framework, "Ionic Documentation," Ionic Framework, [Online]. Available: https://ionicframework.com/docs. [Accessed Feb 2023].

[7] I. Framework, "Capacitor Ionic Framework Documentation," [Online]. Available: https://ionicframework.com/docs/native. [Accessed Aug 2023].

[8] Google, "Angular CLI Documentation," [Online]. Available: https://v17.angular.io/cli. [Accessed May 2023].

[9] I. Framework, "Ionic Angular," [Online]. Available: https://ionicframework.com/docs/angular/overview. [Accessed Mar 2023].

[10] A. Harush, "capacitor-voice-recorder," tchvu3, [Online]. Available: https://github.com/tchvu3/capacitor-voice-recorder. [Accessed May 2023].

[11] I. Framework, "Filesystem Plugin Documentation," [Online]. Available: https://ionicframework.com/docs/native/filesystem. [Accessed May 2023].

[12] Chemerisuk, "cordova-plugin-firebase-authentication Github repository," [Online]. Available: https://github.com/chemerisuk/cordova-plugin-firebase-authentication. [Accessed May 2023].

[13] I. Framework, "Haptics Plugin Documentation," [Online]. Available: https://ionicframework.com/docs/native/haptics. [Accessed May 2023].

[14] Microsoft, "Visual Studio Code Documentation," [Online]. Available: https://code.visualstudio.com/docs. [Accessed May 2023].

[15] HuggingFace, "HuggingFace Documentations," [Online]. Available: https://huggingface.co/docs. [Accessed Jun 2023].

[16] OpenAI, "openai/whisper-large-v3 Model," HuggingFace , [Online]. Available: https://huggingface.co/openai/whisper-large-v3. [Accessed Jul 2023].

[17] Microsoft Research, "Document AI," Microsoft Research, [Online]. Available: https://www.microsoft.com/en-us/research/project/document-ai/. [Accessed: Aug. 2024].

[18] impira, "impira/layoutlm-document-qa Model," HuggingFace , [Online]. Available: https://huggingface.co/impira/layoutlm-document-qa. [Accessed Aug 2023].

[19] Google LLC, "GEMMA-7B Model Documentation," Hugging Face, [Online]. Available: https://huggingface.co/google/gemma-7b. [Accessed: Feb. 2023].

[20] LangChain, "LangChain Documentation," [Online]. Available: https://python.langchain.com/v0.2/docs/introduction/. [Accessed Jan 2024].

[21] LangChain, "LangChain Document Loaders Documentation," [Online]. Available: https://python.langchain.com/v0.1/docs/modules/data_connection/document_loaders/. [Accessed Jan 2024].

[22] OpenAI, "Models Documentation," OpenAI, [Online]. Available: https://platform.openai.com/docs/models. [Accessed: Oct 2023].

[23] I. Academy, "How to Build Your First Ionic 6 App with API Calls," [Online]. Available: https://ionicacademy.com/ionic-6-app-api-calls/. [Accessed Apr 2023].

[24] TypeScript, "TypeScript," [Online]. Available: https://www.typescriptlang.org. [Accessed: May 2023].

[25] Google, "Gemma AI Documentation," Google AI, [Online]. Available: https://ai.google.dev/gemma/docs?hl=es-419. [Accessed: Aug. 2024].

[26] Google, "Firestore," Firebase, [Online]. Available: https://firebase.google.com/products/firestore?hl=es-419. [Accessed: Aug. 2024].

[27] Ionic Framework, "Gestures," Ionic Framework Documentation, [Online]. Available: https://ionicframework.com/docs/utilities/gestures. [Accessed: Sep. 2024].

[28] I. Platform, "Voice Communications with Ionic," [Online]. Available: https://platform.ionic.io/multi-experience/voice. [Accessed May 2023].

[29] HuggingFace, "HuggingFace," [Online]. Available: https://huggingface.co. [Accessed Jun 2023].

[30] HuggingFace, "Api-inference documentation," [Online]. Available: https://huggingface.co/docs/api-inference/quicktour. [Accessed Jun 2023].

[31] HuggingFace, "Inference Endpoints (dedicated) documentation," [Online]. Available: https://huggingface.co/docs/inference-endpoints/index. [Accessed Jun 2023].

[32] HuggingFace, "Transformers documentation," [Online]. Available: https://huggingface.co/docs/transformers/index. [Accessed Jun 2023].

[33] OpenAI, "Whisper," OpenAI, [Online]. Available: https://openai.com/index/whisper/. [Accessed: Jul 2024].

[34] npm, Inc., "npm," npm, [Online]. Available: https://www.npmjs.com. [Accessed: Aug. 2024].

[35] OpenJS Foundation, "Node.js," Node.js, [Online]. Available: https://nodejs.org/en. [Accessed: Aug. 2024].

[36] OpenAI, "OpenAI Documentation," [Online]. Available: https://platform.openai.com/docs/overview. [Accessed Feb 2024].

[37] OpenAI, "OpenAI API reference Documentation," [Online]. Available: https://platform.openai.com/docs/api-reference/introduction. [Accessed Feb 2024].

[38] Google, "Firebase," Firebase, [Online]. Available: https://firebase.google.com. [Accessed: May 2023].

[39] Ionic, "Ionic Framework," Ionic Framework, [Online]. Available: https://ionicframework.com. [Accessed: Feb 2023].

[40] Ionic, "Capacitor," Capacitor, [Online]. Available: https://capacitorjs.com. [Accessed: Aug. 2024].

[41] OpenAI, "OpenAI," OpenAI, [Online]. Available: https://openai.com. [Accessed: Aug. 2024].

[42] [1] Apache Software Foundation, "Apache Cordova," Apache Cordova, [Online]. Available: https://cordova.apache.org. [Accessed: Feb 2024].

[43] Ionic Framework, "Ionic Native," Ionic Framework Documentation, [Online]. Available: https://ionicframework.com/docs/native/. [Accessed: Sep 2023].

[44] LangChain, "LangChain Python API reference documentation," [Online]. Available: https://api.python.langchain.com/en/latest/langchain_api_reference.html. [Accessed Jan 2024].

[45] LangChain, "LangChain," LangChain, [Online]. Available: https://www.langchain.com. [Accessed: Aug. 2024].

[46] S. E. Maameri, "smaameri/multi-doc-chatbot Github Repository," [Online]. Available: https://github.com/smaameri/multi-doc-chatbot. [Accessed Feb 2024].

[47] LangChain, "LangChain Python Tutorial Chatbot Documentation," [Online]. Available: https://python.langchain.com/v0.2/docs/tutorials/chatbot/. [Accessed Feb 2024].

[48] S. Maameri, "Building a Multi-Document Reader and Chatbot With LangChain and ChatGPT," [Online]. Available: https://betterprogramming.pub/building-a-multi-document-reader-and-chatbot-with-langchain-and-chatgpt-d1864d47e339. [Accessed Feb 2024].

[49] Python Software Foundation, "Python," Python.org, [Online]. Available: https://www.python.org. [Accessed: Aug. 2024].

[50] Microsoft, "Visual Studio Code," Visual Studio Code, [Online]. Available: https://code.visualstudio.com. [Accessed: Aug. 2024].

[51] Mozilla, "JavaScript," Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Accessed: Aug 2023].

[52] Mozilla, "CSS," Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/es/docs/Web/CSS. [Accessed: Aug 2023].

[53] Mozilla, "HTML," Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTML. [Accessed: Aug 2023].

ANEXO I: ESPECIFICACIÓN DE REQUISITOS

En este Anexo I se presenta una descripción detallada de los actores involucrados y la especificación de los requisitos que intervienen en la aplicación desarrollada.

I.A ACTORES

A continuación, se enumeran los principales actores del sistema y se proporciona una descripción detallada de cada uno de ellos.

- **Usuario General:** Los usuarios generales pueden navegar por la aplicación y utilizar la mayoría de sus funcionalidades. Sin embargo, no tienen acceso a la funcionalidad de autenticación de cuentas, ni a la gestión y almacenamiento de datos de usuarios. En términos generales, los usuarios pueden utilizar las funciones de reconocimiento de voz, reconocimiento de documentos y el chatbot que proporciona información a partir de los documentos cargados en el directorio de backend.

- **Usuario Registrado:** Los usuarios que se registren en la aplicación tendrán acceso a una base de datos de usuarios proporcionada por Firebase [38]. Esta funcionalidad permite la autenticación, gestión y almacenamiento de datos, evitando así la pérdida de documentos y permitiendo la personalización de la cuenta con la información necesaria.

- **Administrador:** El administrador será responsable de la gestión de las cuentas de usuario, incluyendo la desactivación y modificación de dichas cuentas. Además, tendrá la capacidad de administrar los datos almacenados en la aplicación, configurar los componentes inteligentes y realizar análisis de los componentes para evaluar su rendimiento.

I.B ESPECIFICACIÓN DE REQUISITOS

En esta sección se van a explicar los casos de uso definidos para la aplicación. Los casos de uso se han agrupado modularmente de la siguiente manera:

1. Cuenta de usuario
2. Perfil de usuario
3. Componente de la voz
4. Componente de los documentos
5. Componente del Chatbot
6. Componente de la generación del texto

I.B.1 Cuentas de Usuario

Requisito	**Registro**	
Identificador	1.1	
Prioridad	Alta	
Precondición	NA	
Descripción	Los usuarios tienen la posibilidad de crear una cuenta en esta aplicación.	
Entrada	Usuario (Correo Electrónico) y Contraseña	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña Login de la aplicación.
	2	La pestaña muestra los campos de usuario y contraseña, permitiendo al usuario ingresar esta información para completar el registro.
	3	El usuario completa todos los campos.
	4	El usuario hace clic en el botón “Create Account” y el sistema valida la información.
	5	El sistema autentifica la cuenta y almacena las informaciones en Firebase [38].
Postcondición	La cuenta del usuario ha sido creada.	
Excepciones	Paso	Acción
	4	Se muestra el mensaje de error si la contraseña no cumple con los requisitos.
	4	Se muestra el mensaje de error si el correo electrónico no es correcto.
	4	Se muestra el mensaje de error si el correo electrónico o el nombre de usuario ya existen.
Comentarios	NA	
Actores	Usuario no registrado	

Tabla 1: Registro del usuario

Requisito	**Login**	
Identificador	1.2	
Prioridad	Alta	
Precondición	El usuario ya tiene registrada una cuenta.	
Descripción	Los usuarios registrados pueden iniciar sesión de su cuenta.	
Entrada	Usuario (Correo Electrónico), Contraseña	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace clic en la pestaña "Login".
	2	El sistema muestra los campos requeridos, solicitando al usuario toda la información necesaria para iniciar sesión.
	3	El usuario completa todos los campos (usuario y contraseña).
	4	El usuario hace clic en el botón "Login" y el sistema valida las informaciones.
	5	El sistema muestra la pestaña perfil.
Postcondición	La sesión de la cuenta del usuario ha sido inicializada.	
Excepciones	Paso	Acción
	4	No se ha podido realizar el inicio de sesión.
Comentarios	NA	
Actores	Usuario registrado	

Tabla 2: Login del usuario

Requisito	**Logout**	
Identificador	1.3	
Prioridad	Alta	
Precondición	El usuario ha iniciado sesión en la cuenta.	
Descripción	Los usuarios pueden cerrar la sesión iniciada para iniciar una nueva sesión o navegar sin estar registrados.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede al perfil en la aplicación y hace clic en el ícono “Logout”.
	2	El sistema cierra la sesión de la cuenta y vuelve a la pantalla principal.
Postcondición	La sesión de la cuenta del usuario ha sido cerrada.	
Excepciones	Paso	Acción
	2	NA
Comentarios	NA	
Actores	Usuario registrado	

Tabla 3: Logout

Requisito	**Dar de baja al usuario**
Identificador	1.4
Prioridad	Media
Precondición	NA
Descripción	El administrador puede eliminar las cuentas de usuario correspondientes.
Entrada	UID usuario, Nombre de Usuario o Correo electrónico.
Salida	NA

Secuencia normal	Paso	Acción
	1	El administrador accede a la pestaña de autenticación de Firebase [38] de este proyecto.
	2	Busca el usuario correspondiente mediante el UID o el correo electrónico del usuario.
	3	Clic en “Ver más opciones”
	4	Clic en “Borrar Cuenta”
Postcondición	La cuenta del usuario correspondiente a UID o correo electrónico proporcionada ha sido eliminada.	
Excepciones	Paso	Acción
	2	No se ha encontrado ningún usuario.
	4	El administrador hace click en “Cancelar”.
Comentarios	NA	
Actores	Administrador	

Tabla 4. Dar de baja al usuario

Requisito	**Reestablecer contraseña del usuario**	
Identificador	1.5	
Prioridad	Media	
Precondición	NA	
Descripción	El administrador puede reestablecer la contraseña del usuario.	
Entrada	UID usuario, Nombre de Usuario o Correo Electrónico	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El administrador accede a la pestaña de autenticación de Firebase [38] de este proyecto.
	2	Busca el usuario correspondiente mediante el UID o el correo electrónico del usuario.

	3	Clic en “Ver más opciones”
	4	Clic en “Restablecer la contraseña” y envía un correo electrónico de restablecimiento de la contraseña.
	5	El usuario restablece la contraseña en el correo electrónico
Postcondición	La contraseña del usuario ha sido restablecida.	
Excepciones	Paso	Acción
	2	No se ha encontrado ningún usuario.
	4	El administrador hace click en “Cancelar”.
	6	No se han podido modificar los datos del usuario.
Comentarios	NA	
Actores	Administrador, Usuario	

Tabla 5: Restablecer la contraseña

I.B.2 Perfil de usuario

Requisito	**Mostrar y modificar el perfil del usuario**	
Identificador	2.1	
Prioridad	Alta	
Precondición	El usuario debe haber completado el proceso de registro y haber iniciado sesión en la aplicación.	
Descripción	El usuario registrado debe poder ver su perfil y modificar los datos en él.	
Entrada	NA	
Salida	Perfil del usuario modificado	
Secuencia normal	Paso	Acción
	1	El usuario inicializa la sesión de la cuenta en la aplicación.

	2	El sistema muestra la pestaña del perfil en la aplicación.
	3	El usuario hace clic en el campo que quiere modificar.
	4	El usuario ingresa el nuevo dato en el campo que desea modificar.
	5	El sistema actualiza el campo con el dato proporcionado.
Postcondición	El usuario puede visualizar los nuevos datos actualizado en su perfil.	
Excepciones	Paso	Acción
	1	Si el usuario no ha iniciado sesión, debe ser redirigido a la página de inicio de sesión de la aplicación y realiza la inicialización de la sección.
	4	En caso de que no sea posible actualizar el perfil del usuario debido a problemas técnicos, se deberá mostrar un mensaje de error.
Comentarios	NA	
Actores	Usuario registrado	

Tabla 6: Mostrar y modificar el perfil del usuario

Requisito	**Gestionar los datos del perfil del usuario**	
Identificador	2.2	
Prioridad	Media	
Precondición	El usuario debe haber ingresado algunos datos en el perfil.	
Descripción	El administrador tiene la capacidad de visualizar y gestionar la información del perfil de los usuarios.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El administrador accede a la pestaña Database de Firebase [38].
	2	Hace clic en “Edit” para modificar datos.
	3	Clic en “Delete” para borrar
	4	Clic en “Agregar” para añadir datos.

	5	El administrador puede jugar con las herramientas a las colecciones de datos.
Postcondición	El administrador podrá visualizar los datos gestionados.	
Excepciones	Paso	Acción
	2	Tiene la opción de “cancelar”
	3	Con la opción de “cancelar”
	4	Podrá cancelar la agregación de los datos.
Comentarios	NA	
Actores	Administrador	

Tabla 7: Gestionar los datos del perfil

I.B.3 Componente de la voz

Requisito	**Reconocimiento de Voz**	
Identificador	3.1	
Prioridad	Alta	
Precondición	La aplicación debe estar operativa, y el usuario debe contar con un dispositivo equipado con capacidad de entrada de voz, como un micrófono.	
Descripción	La aplicación integra la capacidad de reconocimiento de voz para optimizar la interacción de usuarios con discapacidades visuales, facilitando la conversión inteligente de audio a texto para el manejo de comandos y tomar notas.	
Entrada	Comandos de voz del usuario	
Salida	Textos traducidos desde un audio	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Voice” de la aplicación.
	2	El usuario mantiene presionado el icono del micrófono para iniciar la grabación de audio.
	3	El sistema transmite el audio al modelo una vez que se libera el icono del micrófono, indicando el final de la grabación.

	4	La aplicación presenta el audio grabado y su correspondiente traducción a texto en la sección inferior del audio.
Postcondición	Éxito en la funcionalidad de reconocimiento de voz para interactuar con la aplicación, logrando la traducción del audio a texto.	
Excepciones	Paso	Acción
	3	Si el dispositivo del usuario no soporta la entrada de voz o si el reconocimiento de voz no está disponible debido a razones técnicas, se debe mostrar un mensaje de error en la consola.
Comentarios	Requisitos Específicos: • El entorno ofrece una manera de entrada de voz por parte de los usuarios. • Es necesario actualizar el modelo en la plataforma HuggingFace [29] para asegurar una conexión estable con el modelo.	
Actores	Usuario General	

Tabla 8: Reconocimiento de Voz

Requisito	**Salida de Voz**	
Identificador	3.2	
Prioridad	Alta	
Precondición	La aplicación debe estar en funcionamiento, y el usuario debe haber grabado al menos un audio.	
Descripción	La aplicación ofrece una funcionalidad de salida de voz que permita a los usuarios verificar la exactitud de la traducción del audio.	
Entrada	Audio grabado.	
Salida	Audio reproducido.	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Voice” de la aplicación.
	2	El usuario hace clic en el audio que desea reproducir.
	3	El sistema reproduce el archivo de audio seleccionado a través del sistema de salida de sonido del entorno.
Postcondición	El sistema reproduce el archivo de audio correspondiente a la selección realizada.	

Excepciones	Paso	Acción
	3	Si el usuario opta por desactivar la salida de voz, la aplicación no podrá realizar la reproducción del audio.
Comentarios	NA	
Actores	Usuario General.	

Tabla 9: Salida de Voz

Requisito	**Eliminar el audio**	
Identificador	3.3	
Prioridad	Media	
Precondición	La aplicación debe estar en funcionamiento, y el usuario debe haber grabado al menos un audio.	
Descripción	La aplicación ofrece una funcionalidad de eliminar el archivo de audio correspondiente.	
Entrada	Audio grabado.	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Voice” de la aplicación.
	2	El usuario hace clic en el icono de la basura del audio que desea eliminar.
	3	El sistema elimina el audio correspondiente tanto de la interfaz como del almacenamiento.
Postcondición	El audio correspondiente ha sido eliminado.	
Comentarios	NA	
Actores	Usuario General.	

Tabla 10: Eliminar audio

I.B.4 Componente de los documentos

Requisito	**Guardar los documentos**	
Identificador	4.1	
Prioridad	Alta	
Precondición	La aplicación debe estar operativa y el usuario debe tener acceso a los documentos que desea cargar desde su dispositivo local.	
Descripción	La aplicación cuenta con un explorador de archivos integrado que permite a los usuarios cargar y guardar documentos.	
Entrada	Documentos	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Studies” de la aplicación.
	2	Haz clic en el icono “+” en la parte inferior derecha.
	3	Clic en la primera opción del icono de archivos.
	4	El sistema abre el explorador de archivos del dispositivo local, permitiendo al usuario seleccionar un documento para cargar en la aplicación.
	5	El usuario selecciona el documento que desea cargar en la aplicación.
	6	El sistema guarda el documento en el explorador de archivos de la aplicación y lo almacena en Firebase [38] para prevenir su pérdida.
Postcondición	El documento cargado desde el dispositivo local está almacenado en el explorador de archivos de la aplicación y podré visualizarlo en la interfaz.	
Comentarios	NA	
Actores	Usuario General.	

Tabla 11: Guardar los documentos

Requisito	**Crear las carpetas**	
Identificador	4.2	
Prioridad	Bajo	
Precondición	La aplicación debe estar operativa.	
Descripción	La aplicación cuenta con un explorador de archivos integrado que permite a los usuarios crear las carpetas.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Studies” de la aplicación.
	2	Haz clic en el icono “+” en la parte inferior derecha.
	3	Clic en la segunda opción del icono de carpetas.
	4	Aparece una ventana que solicita al usuario ingresar el nombre de la carpeta, y el usuario proporciona la información correspondiente.
	5	El sistema genera una carpeta en el explorador de archivos con el nombre proporcionado por el usuario. Además, crea una URL específica para la carpeta recién creada.
Postcondición	La carpeta ha sido creada en el explorador de archivos. El usuario podrá visualizarla en la interfaz y tendrá la opción de acceder a dicha carpeta.	
Comentarios	NA	
Actores	Usuario General.	

Tabla 12: **Crear las carpetas**

Requisito	**Eliminar y copiar los documentos y las carpetas**
Identificador	4.3
Prioridad	Media
Precondición	La aplicación debe estar en funcionamiento y debe tener al menos un documento y una carpeta almacenados.
Descripción	El usuario tendrá la capacidad de eliminar documentos y copiar documentos a las carpetas deseadas.
Entrada	Documentos

Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña “Studies” de la aplicación.
	2	Desliza el documento o la carpeta hacia la derecha
	3	Aparecerá un ícono de papelera. Haga clic en este ícono para eliminar el documento o la carpeta.
	4	Desliza el documento o la carpeta hacia la izquierda.
	5	Aparecerá un ícono con la letra "copy". Haga clic en este ícono para copiar el documento o la carpeta.
	6	Haga clic en la carpeta en la que desea copiar el documento o la carpeta, y el sistema realizará la copia dentro de esa carpeta.
Postcondición	El documento y la carpeta se han eliminado del almacenamiento y ya no aparecen en la interfaz. El documento y la carpeta se han copiado dentro de la carpeta seleccionada con la nueva ubicación URL.	
Comentarios	NA	
Actores	Usuario General.	

Tabla 13: Eliminar y copiar los documentos y las carpetas

Requisito	**Reconocimiento de documentos**	
Identificador	4.4	
Prioridad	Alta	
Precondición	La aplicación debe estar en funcionamiento y debe tener al menos un documento almacenado en el explorador de archivos.	
Descripción	La aplicación incorpora un modelo de reconocimiento de documentos que permite analizar los documentos subidos de manera estructurada e interrogar sobre su contenido.	
Entrada	Documentos	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña "Studies" de la aplicación y navega a la página "l-studies" mediante el menú situado en la parte superior izquierda.

	2	Rellene los dos campos de entrada de la página con el nombre de un documento que haya sido subido en la página "Studies" y con una pregunta relacionada con el contenido de dicho documento.
	3	Clic en el botón “Enviar”
	4	El modelo recibe las informaciones y ejecuta el modelo.
	5	En la interfaz de la página “l-studies”, se muestra la respuesta a la pregunta formulada, junto con los resultados del análisis del documento seleccionado.
Postcondición	NA	
Comentarios	NA	
Actores	Usuario General.	

Tabla 14: Reconocimiento de documentos

I.B.5 Componente del Chatbot

Requisito	**Selector de documentos de chatbot**	
Identificador	5.1	
Prioridad	Media	
Precondición	La aplicación debe estar en funcionamiento y debe tener al menos un documento almacenado en el explorador de archivos.	
Descripción	La aplicación incorpora un selector de documentos en la página “d-studies”, permitiendo al usuario elegir entre los documentos almacenados en el explorador de archivos.	
Entrada	Documentos	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña "Studies" de la aplicación y navega hacia la página "d-studies" utilizando el menú ubicado en la parte superior izquierda.
	2	Haz clic en el ícono de archivo ubicado en la parte superior derecha.
	3	Selecciona los documentos sobre los cuales desea realizar la consulta a través del chatbot.
	4	El sistema envía el contenido de los documentos seleccionados al modelo de chatbot para su análisis preliminar.
Postcondición	NA	

Comentarios	NA
Actores	Usuario General.

Tabla 15: Selector de documentos

Requisito	**Chatbot**	
Identificador	5.2	
Prioridad	Alta	
Precondición	La aplicación debe estar operativa y contar con al menos un documento almacenado en el explorador de archivos, así como al menos un documento seleccionado mediante el selector de archivos en la página "d-studies".	
Descripción	La aplicación incorpora un chatbot en la página “d-studies” que puede analizar, interactuar e interrogar sobre los contenidos de los documentos almacenados en el explorador de archivos de la página “studies”, después de que estos hayan sido seleccionados mediante el selector de archivos.	
Entrada	Documentos	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El chatbot analiza los contenidos de los documentos específicos seleccionados mediante el selector de archivos.
	2	Introduce una pregunta en el campo "Type message".
	3	Haz clic en el botón “send”
	4	El chatbot proporciona una respuesta detallada basada en el análisis de la información contenida en los documentos seleccionados, y muestra los resultados en la interfaz tras la muestra de la pregunta.
Postcondición	NA	
Comentarios	NA	
Actores	Usuario General.	

Tabla 16: Chatbot

Requisito	**El promp del chatbot**	
Identificador	5.3	
Prioridad	Alta	
Precondición	NA	
Descripción	En el backend de la aplicación, se ofrece un campo configurable que permite modificar el prompt del chatbot, facilitando así su adaptación a diferentes áreas de estudio y otras necesidades específicas.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	Accede al único backend de la aplicación en Python [49].
	2	Proporciona una descripción del prompt con la información que el chatbot debe recibir y cómo debe actuarse en la aplicación.
	3	Ejecuta en el entorno virtual de Python [49] la nueva script.
	4	Ejecuta el servidor de Ionic [39].
Postcondición	NA	
Comentarios	De esta manera, podemos obtener un chatbot diferente con distintas actuaciones y diferentes áreas de conocimientos.	
Actores	Administrador.	

Tabla 17: Prompt Chatbot

Requisito	**Selector de sesiones de chatbot**	
Identificador	5.4	
Prioridad	Media	
Precondición	En la carpeta de las sesiones hay que tener al menos una sesión guardado	
Descripción	La aplicación incorpora un selector de sesiones de chatbot en la página “d-studies”, permitiendo al usuario elegir y recuperar una sesión anterior.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción

	1	El usuario accede a la pestaña "Studies" de la aplicación y navega hacia la página "d-studies" utilizando el menú ubicado en la parte superior izquierda.
	2	Clic en el selector de las sesiones que pone new sesión por defecto.
	3	Clic en la sesión deseada para la recuperación
	4	Clic en ok y se muestra las historiales de esta sesión
Postcondición	NA	
Comentarios	Así recuperando la historial de la sesión elegida y podrá ir preguntando sobre los contenidos aportados historialmente.	
Actores	Usuario general.	

Tabla 18: Selector de sesiones de chatbot

Requisito	**Eliminar una sesión del chatbot**	
Identificador	5.5	
Prioridad	Baja	
Precondición	En la carpeta de las sesiones hay que tener al menos una sesión guardado	
Descripción	La aplicación tiene la opción de eliminar las sesiones de chatbot	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña "Studies" de la aplicación y navega hacia la página "d-studies" utilizando el menú ubicado en la parte superior izquierda.
	2	El usuario selecciona la sesión que desea eliminar usando el selector anterior
	3	Clic en el botón del icono papelera
Postcondición	NA	
Comentarios	Al clic en el botón la sesión actual seleccionada se elimina.	
Actores	Usuario general.	

Tabla 19: Eliminar una sesión del chatbot

Requisito	**Dictador de chatbot**	
Identificador	5.6	
Prioridad	Alta	
Precondición	NA	
Descripción	En la página del chatbot, d-studies, tiene la opción de dictar las preguntas del usuario en el campo de la entrada de los mensajes usuarios.	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña "Studies" de la aplicación y navega hacia la página "d-studies" utilizando el menú ubicado en la parte superior izquierda.
	2	Mantiene presionado el botón del micrófono y empieza a dictar la pregunta.
	3	Suelta el botón y genera la pregunta en texto en el campo de la entrada.
Postcondición	NA	
Comentarios	NA	
Actores	Usuario general.	

Tabla 20: Dictador de chatbot

I.B.6 Componente de la generación del texto

Requisito	**Generación del texto**	
Identificador	6.1	
Prioridad	Alta	
Precondición	NA	
Descripción	En la página studies, hay un botón para acceder al modal para la generación del texto a partir de un contexto proporcionado por el usuario	
Entrada	NA	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la pestaña studies de la aplicación y clic en el botón izquierta inferior para acceder al modal.

	2	Introduce el contexto en el campo de la entrada
	3	Clic en send y se genera los textos a partir del contexto aportado.
Postcondición	NA	
Comentarios	NA	
Actores	Usuario general.	

Tabla 21: Generación del texto

ANEXO II: GUÍA DEL USUARIO

En este anexo II, se proporcionan paso a paso las instrucciones detalladas para la ejecución y el uso de la aplicación web. Asimismo, se describen la utilización de las funcionalidades disponibles.

II.A EJECUCIÓN DEL BACKEND Y FRONTEND

En esta sección, se describen los pasos necesarios para ejecutar tanto el backend en Python [49] como el frontend en Ionic [39] de la aplicación web.

II.A.1 Ejecución de Python

El primer paso consiste en abrir una terminal dentro del directorio **Langchain-Model**. A continuación, debe ejecutar el comando **"pip install virtualenv"** para instalar el entorno virtual de Python [49]. Posteriormente, se crea un entorno virtual con el comando **"virtualenv myenv"** y proceda a activarlo mediante el comando **"source myenv/bin/activate"**.

Figura II.A.1.1: Entorno virtual

El siguiente paso consiste en instalar las dependencias especificadas en el archivo **requirements.txt** dentro del entorno virtual, utilizando el comando **"pip install -r requirements.txt"**. Una vez completada la instalación, ejecute el script **main.py** en el mismo entorno virtual mediante el comando **"python main.py"**, con el fin de ejecutar el script e iniciar y activar el servidor backend de Python [49].

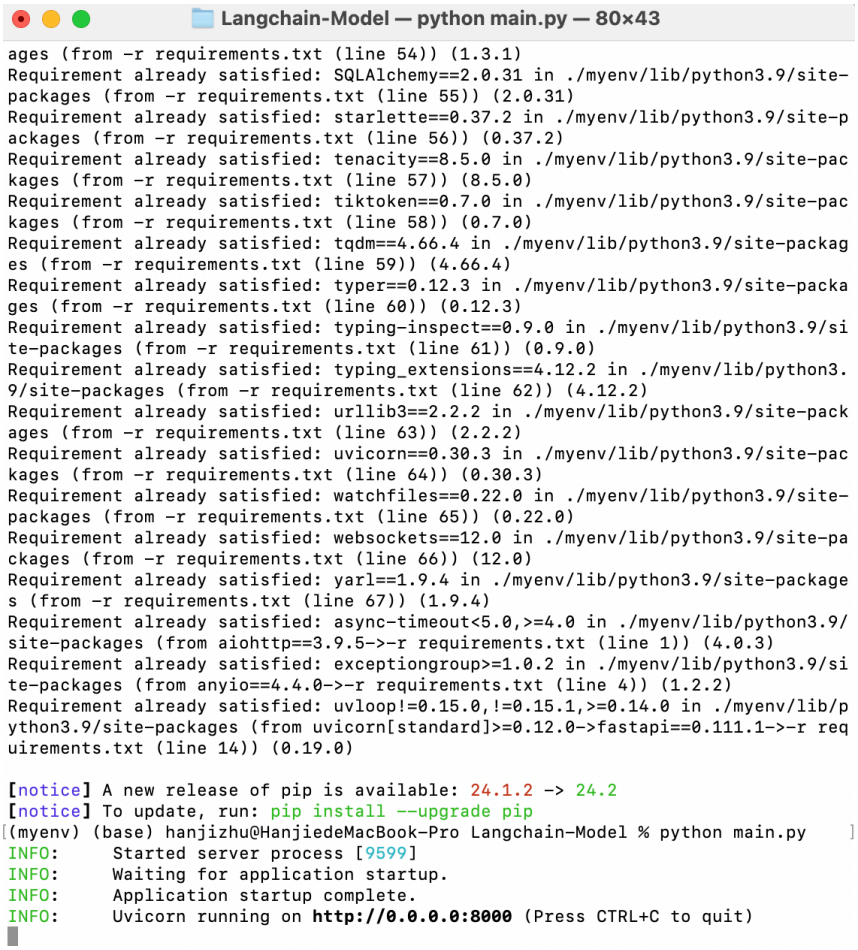


Figura II.A.1.2: Ejecución de Python

II.A.2 Ejecución de Ionic

En Visual Studio Code [50], se debe abrir el directorio **sens-a-ilearn** y acceder a la consola de terminal dentro de dicho directorio. Después, ejecute el comando **"npm install"** para instalar las dependencias necesarias de npm [34]. Luego, inicie el servidor web del frontend en Ionic [39] utilizando el comando **"npm start"**.

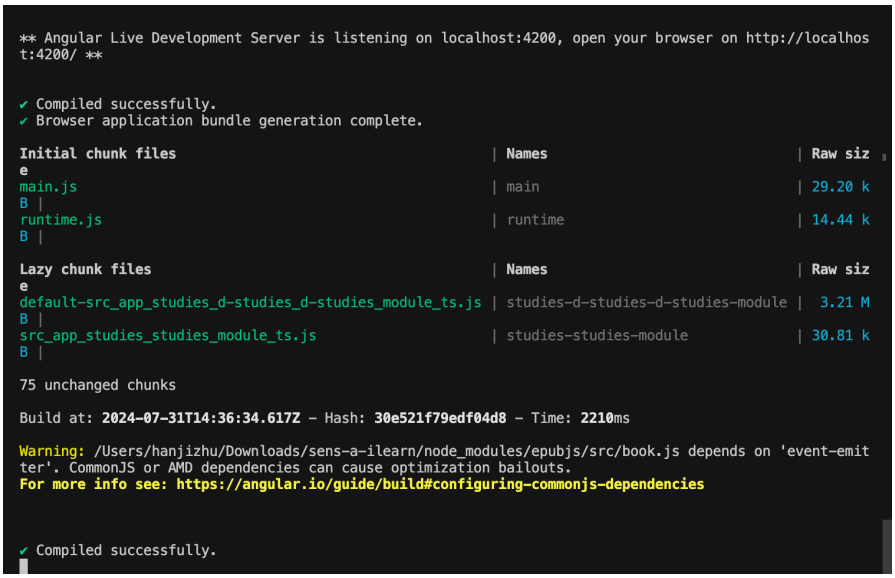


Figura II.A.2.1: Ejecución de Ionic

II.B Aplicación Web

En la sección anterior, se ha iniciado un servidor Angular [8] en localhost:4200, el cual puede ser accedido a través de la dirección http://localhost:4200/ utilizando cualquier navegador web. En esta sección, se proporcionará una guía sobre el uso de cada funcionalidad disponible en la aplicación web.

II.B.1 Login and Register

Accedemos a la dirección http://localhost:4200/tabs/tab3 de la aplicación web, que corresponde a la sección destinada al inicio de sesión y al registro de usuarios.

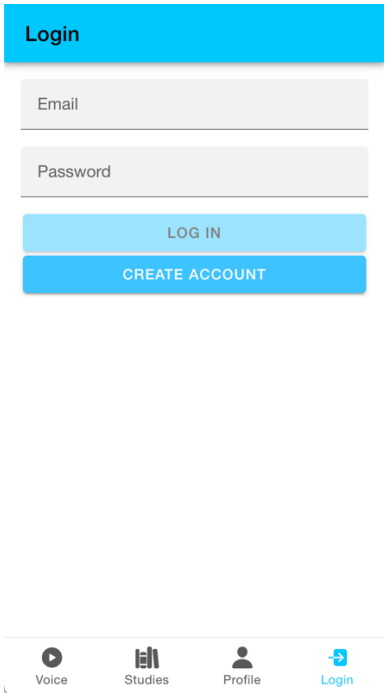


Figura II.B.1.1: Interfaces de Login and register.

Introduzca una dirección de correo electrónico y una contraseña válidas, y haga clic en el botón "Create Account" para proceder con la creación de la cuenta.

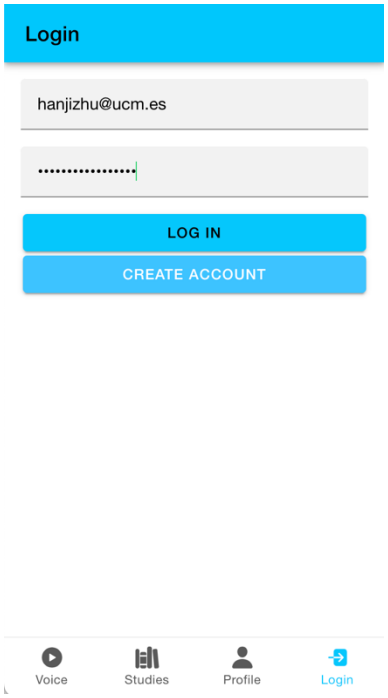


Figura II.B.1.2: Create Account.

Si la dirección de correo electrónico y la contraseña son válidas, la aplicación web redirige al usuario a su perfil y registra la autenticación del usuario en Firebase [38].

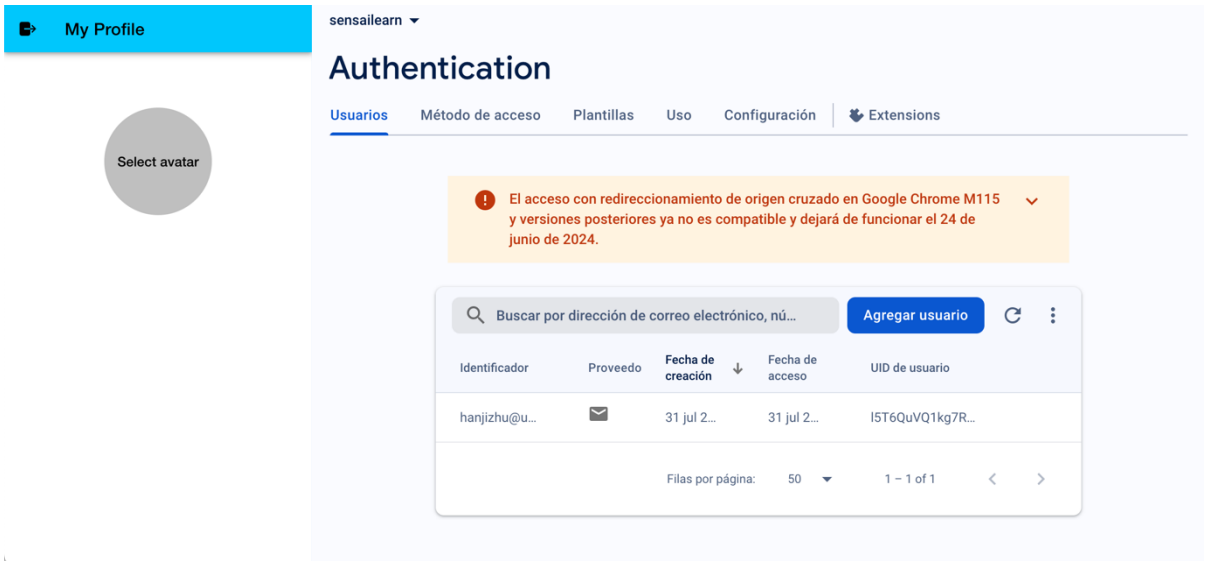


Figura II.B.1.3: Profile and authentication.

El proceso para iniciar sesión sigue los mismos pasos que el registro, pero el usuario debe tener una cuenta previamente registrada con la información de autenticación en Firebase [38]. El usuario debe introducir su correo electrónico y contraseña de la cuenta registrada, y luego hacer clic en el botón "Log in" en la página mostrada en la Figura II.B.1.2.

En caso de que el usuario introduzca un correo electrónico que no corresponda a una cuenta registrada o una contraseña incorrecta durante el inicio de sesión, la aplicación mostrará una ventana indicando que el inicio de sesión ha fallado. De manera similar, si el usuario intenta registrarse con un correo electrónico ya registrado, se mostrará una ventana informando del error de la registración.

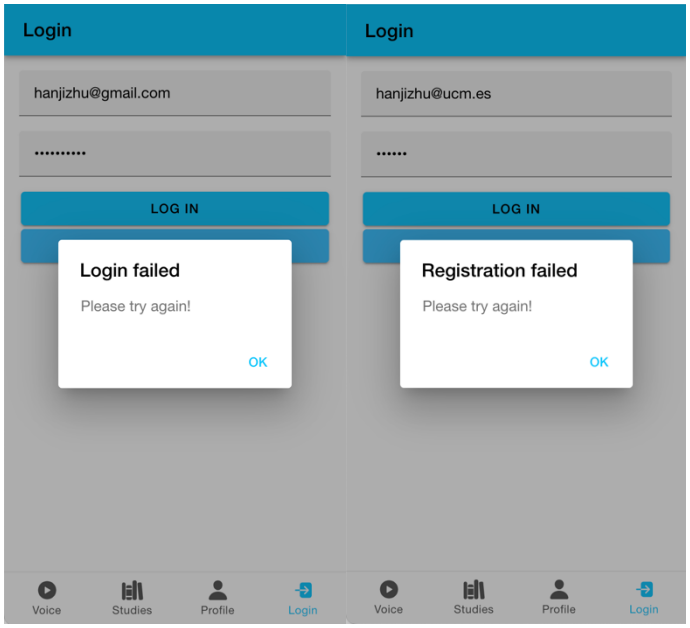


Figura II.B.1.4: Login and Registration failed.

Además, en el caso de que el usuario introduzca un correo electrónico o una contraseña no válidos, los botones "Log in" y "Create Account" se desactivarán.

II.B.2 Profile

Al iniciar sesión en la cuenta, se accede a la página de perfil del usuario. En esta página, no hay ningún contenido, pero incluye una funcionalidad de prueba para la conexión de Database de Firestore [26]: la carga de un avatar. Esta función permite la visualización del avatar y el almacenamiento de la imagen en la base de datos de Firestore [26].

La carga de avatar sigue los siguientes pasos: en la página de perfil, haga clic en "Select avatar" como se muestra en la Figura II.B.1.3, elija una imagen para subir como avatar y luego haga clic en "Abrir" de la Figura II.B.2.1.

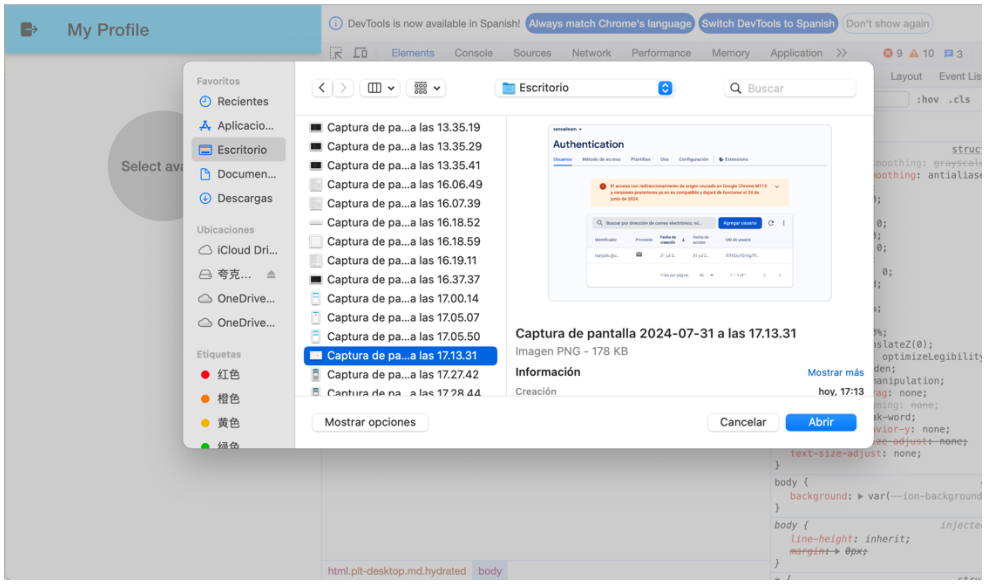
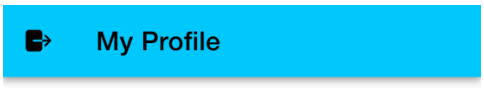


Figura II.B.2.1: Subir avatar.

Como resultado, podemos observar en la Figura II.B.2.2, que el avatar se ha subido correctamente al perfil y la imagen se ha almacenado en Firestore Database [26].



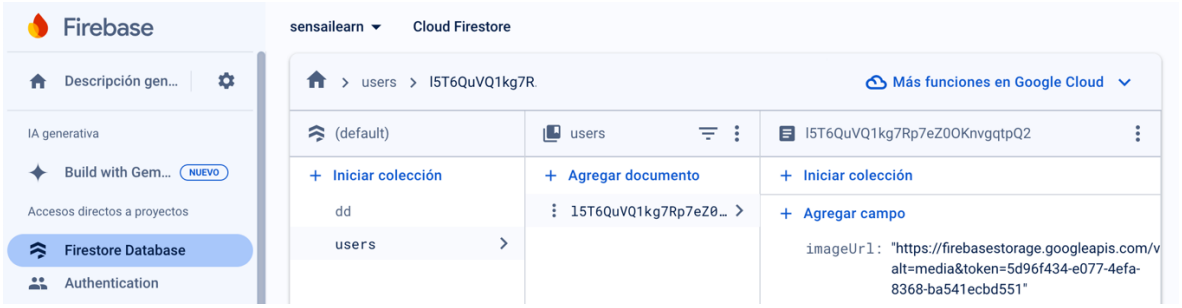


Figura II.B.2.2: Database de avatar.

II.B.3 Voice

Dirigimos a la página Voice de la aplicación usando la dirección http://localhost:4200/tabs/tab1 de web o con el tab de la aplicación situado en la parte inferior.

Figura II.B.3.1: Interfaz de Voice.

En esta página, podemos utilizar la funcionalidad de grabación de audio y su traducción a texto mediante el modelo de inteligencia artificial Whisper [33] con un simple proceso: mantenga presionado el ícono de micrófono y comience a hablar. Una vez que suelte la presión, el audio se transmitirá al modelo para su procesamiento y se mostrará el audio grabado. Al finalizar el procesamiento, la interfaz mostrará la traducción a texto del audio correspondiente en la parte inferior.

Figura II.B.3.2: Func ATT.

También se puede eliminar el audio con un clic en el ícono de la papelera y reproducir el audio con un clic en el archivo correspondiente.

II.B.4 Studies

En esta página, se cargan los documentos que se utilizarán como datos para las funcionalidades integradas en las páginas l-studies y d-studies. Se puede acceder a esta página mediante la pestaña en la parte inferior o a través de la dirección http://localhost:4200/tabs/tab4 en la web.

Figura II.B.4.1: Interfaz de Studies.

Utilizando el ícono "+" situado en la parte inferior derecha (clic en el botón "+"), podrá cargar documentos o crear carpetas. Para cargar documentos, haga clic en el primer botón con el ícono de archivo dentro del menú desplegable del ícono "+" y seleccione el documento que desea cargar desde su dispositivo local.

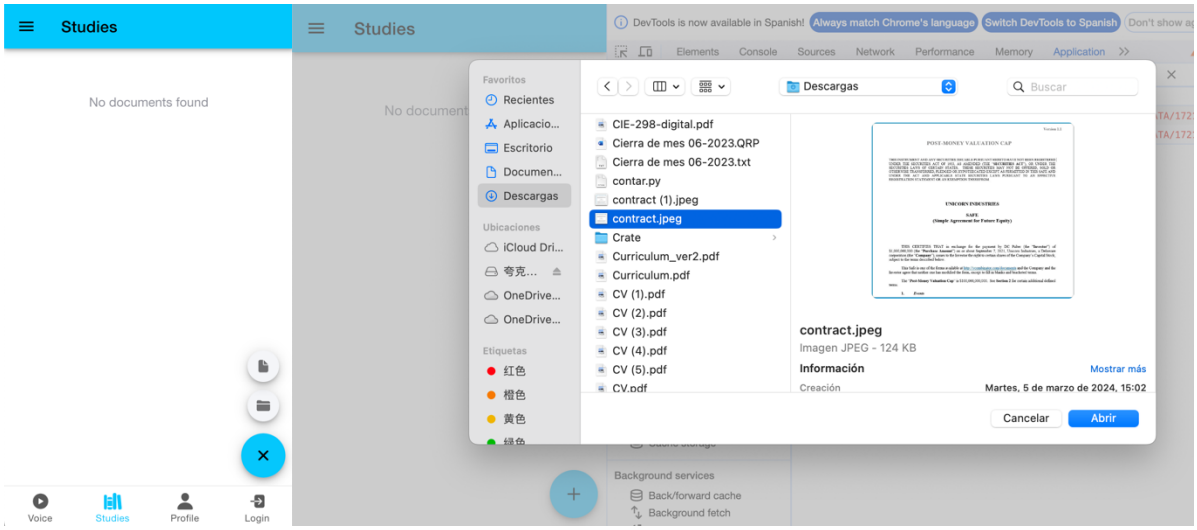


Figura II.B.4.2: Subir docs en Studies.

De esta forma, el documento se integrará en el sistema del explorador de archivos de la aplicación y será almacenado en el Storage de Firebase [26][38].

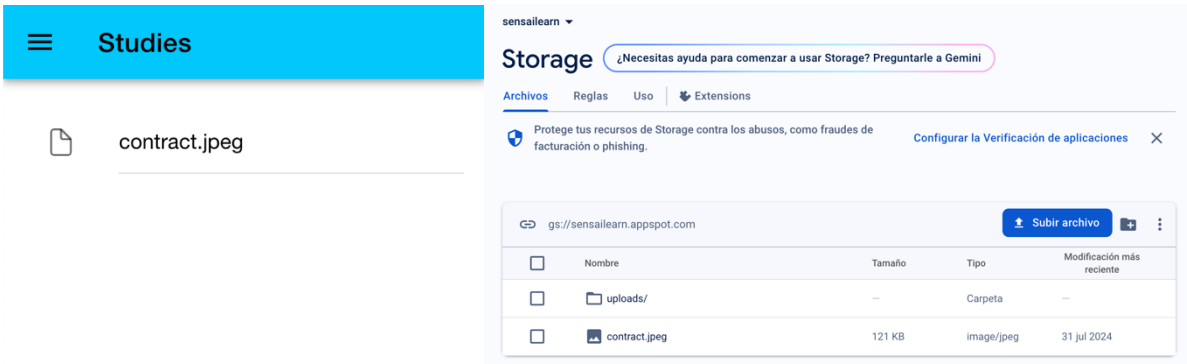


Figura II.B.4.3: Storage de docs

Para la creación de una carpeta, haga clic en el segundo ícono de carpeta dentro del menú desplegable del ícono “+” (Figura II.B.4.2). A continuación, introduzca el nombre de la carpeta en la ventana que aparece tras hacer clic en dicho ícono. De esta manera, la carpeta será creada en el explorador de archivos de la aplicación y almacenada en el almacenamiento de Firebase [38].

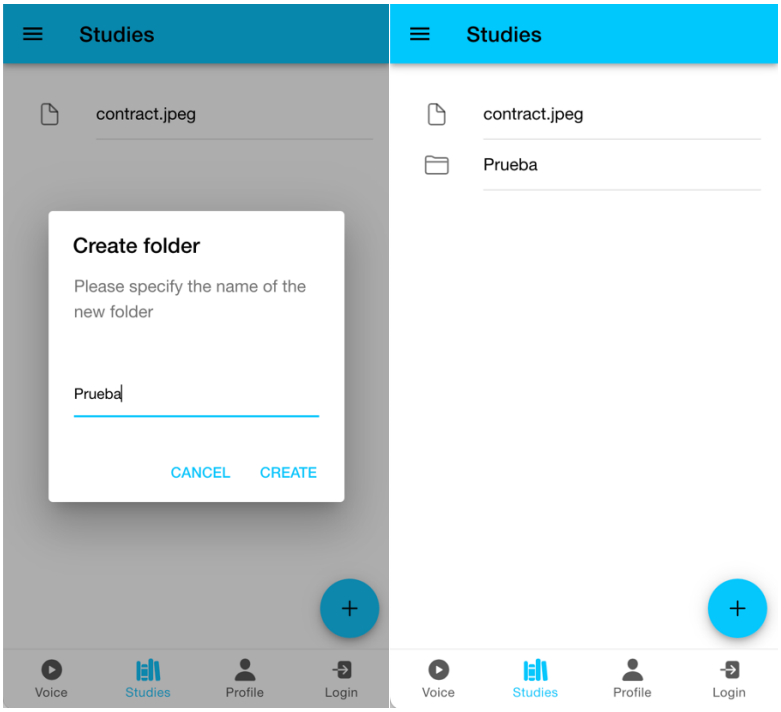


Figura II.B.4.4: Crear carpetas.

Podemos confirmar que la carpeta es accesible dentro de la aplicación mediante una nueva dirección URL. Además, la carpeta se crea correctamente y los documentos dentro de ella se almacenan adecuadamente en Firebase [38], una vez que la carpeta ha sido creada y el documento cargado en la misma desde la aplicación.

Figura II.B.4.5: Storage de carpetas.

En esta página también se ofrecen opciones para eliminar y copiar documentos y carpetas de la siguiente manera: al deslizar el documento o la carpeta hacia la izquierda, aparece un botón de "copiar", mientras que, al deslizar hacia la derecha, aparece un botón de "eliminar". Para eliminar el documento o la carpeta, basta con hacer clic en el ícono de la papelera.

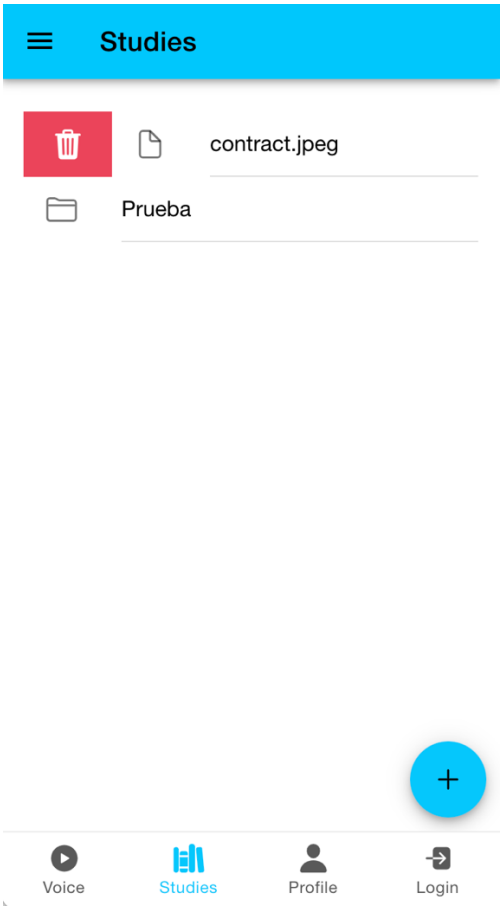


Figura II.B.4.6: Eliminar docs.

Para copiar un documento, primero haga clic en el botón de "copiar". Aparecerá un panel en la parte superior del ícono "+", indicando que el elemento ha sido copiado. Luego, navegue hasta la carpeta de destino en la que desea realizar la copia y haga clic en el panel de copiado. De esta manera, el documento o la carpeta se copiarán a la carpeta seleccionada.

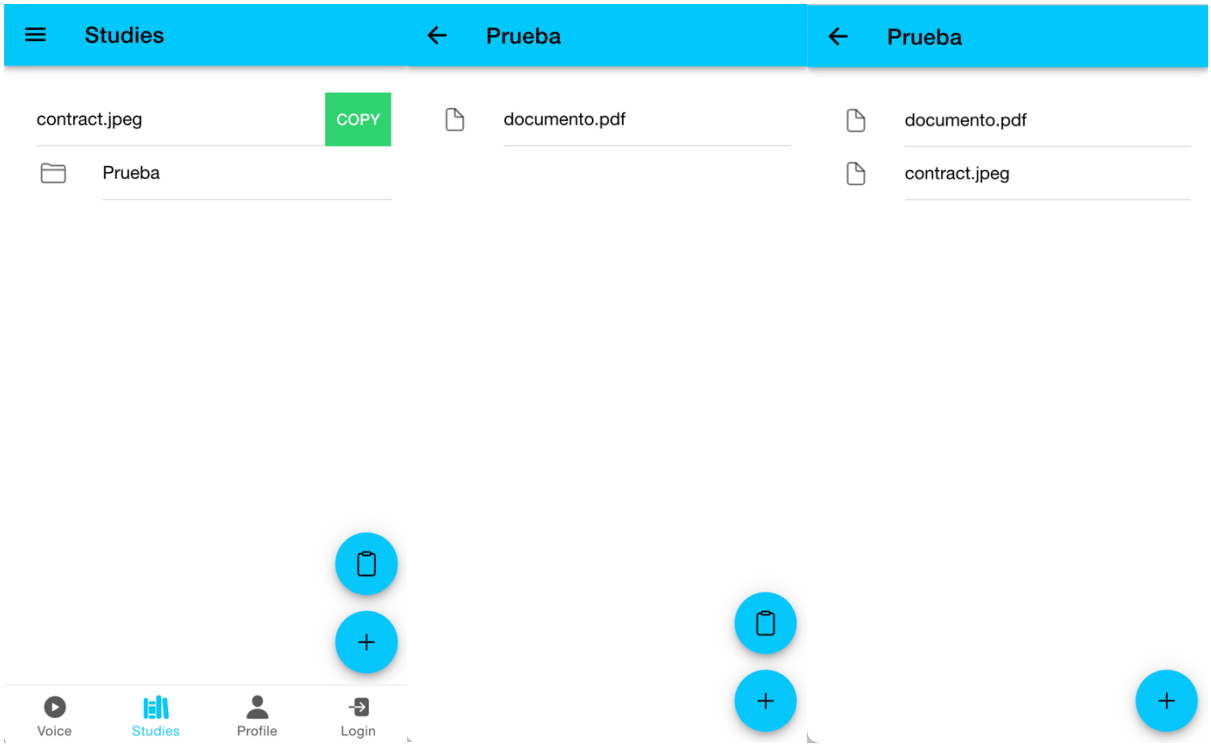


Figura II.B.4.7: Copiar docs

II.B.5 L-Studies

Esta funcionalidad envía un documento al modelo impira/layoutlm-document-qa [17][18] para analizarlo, categorizarlo y proporcionar una respuesta a una pregunta sobre su contenido. El documento puede estar en formato JPG o JPEG u otros, siempre que sea visual.

La página "l-studies" se puede acceder mediante el menú deslizante ubicado en la parte superior izquierda de la página "studies" o a través de la dirección http://localhost:4200/tabs/tab4/l-studies. Está integrada dentro de la página "studies".

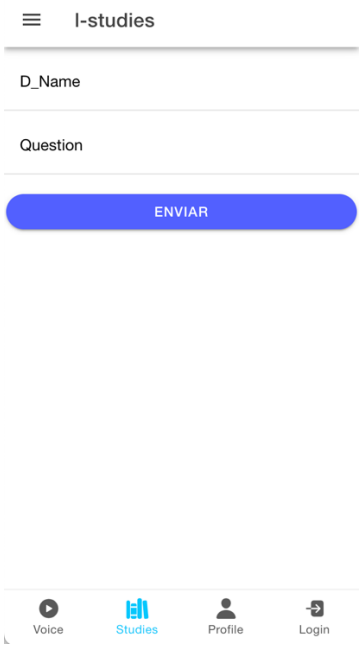
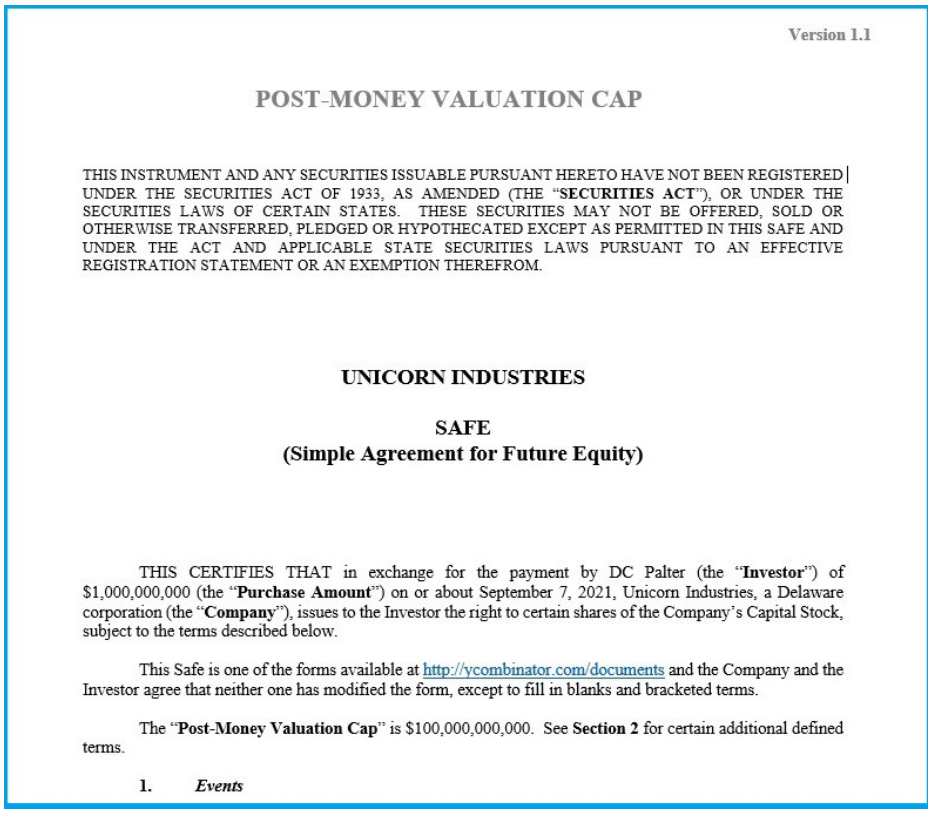


Figura II.B.5.1: Interfaz de Lstudies.

Los pasos para utilizar esta funcionalidad son los siguientes: primero, suba un documento al sistema de explorador de archivos en la página "studies". En este caso, utilizamos el documento de la Figura II.B.5.2 como ejemplo.



Version 1.1

POST-MONEY VALUATION CAP

THIS INSTRUMENT AND ANY SECURITIES ISSUABLE PURSUANT HERETO HAVE NOT BEEN REGISTERED UNDER THE SECURITIES ACT OF 1933, AS AMENDED (THE "**SECURITIES ACT**"), OR UNDER THE SECURITIES LAWS OF CERTAIN STATES. THESE SECURITIES MAY NOT BE OFFERED, SOLD OR OTHERWISE TRANSFERRED, PLEDGED OR HYPOTHECATED EXCEPT AS PERMITTED IN THIS SAFE AND UNDER THE ACT AND APPLICABLE STATE SECURITIES LAWS PURSUANT TO AN EFFECTIVE REGISTRATION STATEMENT OR AN EXEMPTION THEREFROM.

UNICORN INDUSTRIES

SAFE
(Simple Agreement for Future Equity)

THIS CERTIFIES THAT in exchange for the payment by DC Palter (the "**Investor**") of $1,000,000,000 (the "**Purchase Amount**") on or about September 7, 2021, Unicorn Industries, a Delaware corporation (the "**Company**"), issues to the Investor the right to certain shares of the Company's Capital Stock, subject to the terms described below.

This Safe is one of the forms available at http://ycombinator.com/documents and the Company and the Investor agree that neither one has modified the form, except to fill in blanks and bracketed terms.

The "**Post-Money Valuation Cap**" is $100,000,000,000. See **Section 2** for certain additional defined terms.

1. ***Events***

Figura II.B.5.2: doc ejemplo.

En segundo lugar, introducimos el nombre del documento "contract.jpeg" y la pregunta específica sobre el contenido categorizado por el modelo, como "¿Cuál es el monto de la compra?", en los dos campos de la página "l-studies".

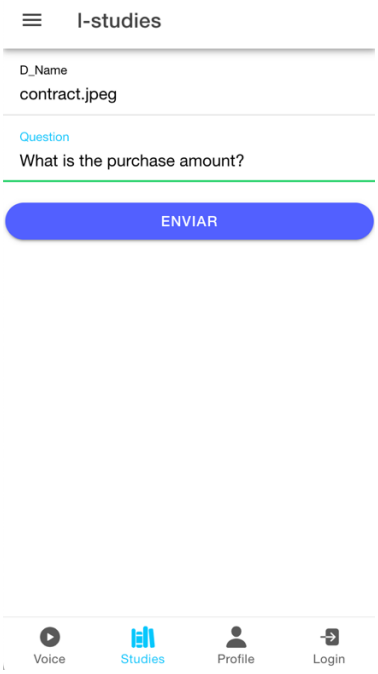


Figura II.B.5.3: Func DQA.

Finalmente, clic en "Enviar" para transmitir los datos al modelo para su procesamiento (es necesario actualizar la página del modelo para que funcione la API, debido al retraso e inestabilidad de la comunidad HuggingFace [29]). El modelo reenvía la respuesta junto con algunos análisis, los cuales se muestran en la parte inferior de la página.

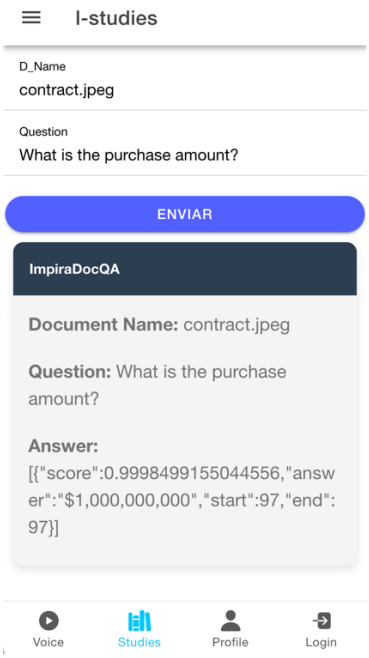


Figura II.B.5.4: Resultado de DQA.

II.B.6 D-Studies

En esta página se presenta un chatbot de análisis de documentos, que nos proporciona información sobre el contenido de los documentos seleccionados a través de un selector de archivos integrado en la página "d-studies".

Podemos acceder a la página utilizando el menú deslizante situado en la parte superior izquierda o a través de la dirección web http://localhost:4200/studies/d-studies.

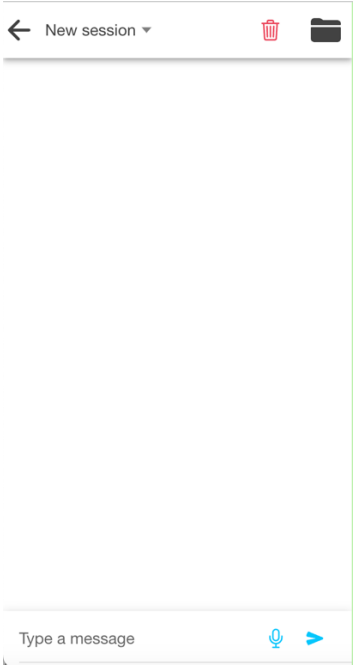


Figura II.B.6.1: Interfaz de Dstudies.

De manera similar, los documentos cuyos contenidos se desean analizar deben estar cargados en el sistema de explorador de archivos en la página "studies". Además, deben ser seleccionados mediante el selector de archivos situado en la parte superior derecha de la página "d-studies".

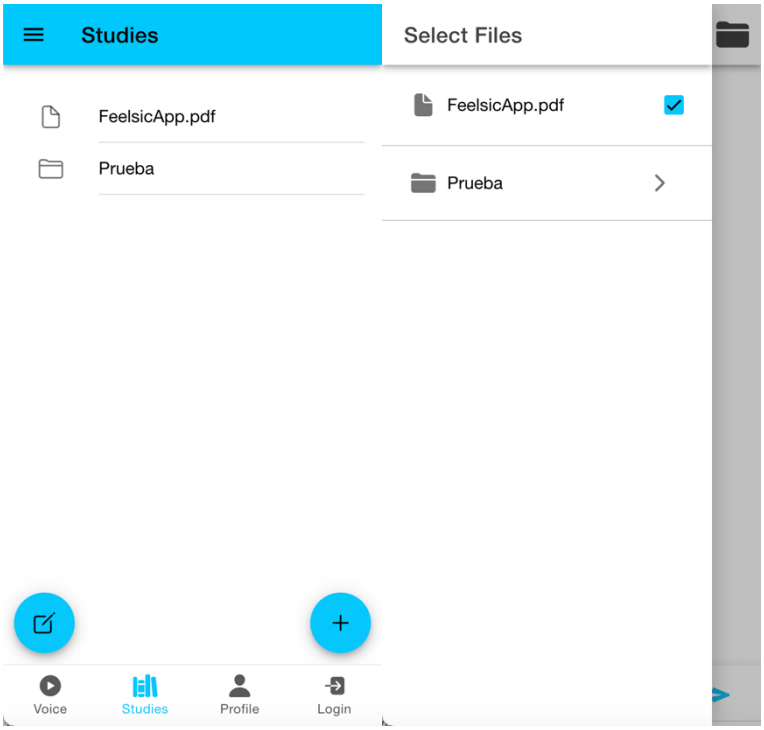


Figura II.B.6.2: Explorador y selector de archivos.

Al hacer clic en los documentos que se desean subir para su análisis, los contenidos y metadatos se transfieren al modelo de OpenAI [22][41]. Y el chatbot genera un mensaje indicando que ya se puede comenzar a hacer preguntas sobre los contenidos de los documentos seleccionados.

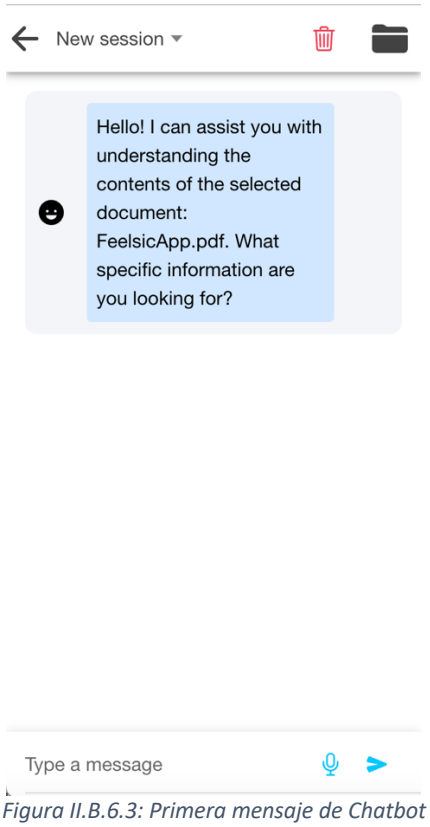


Figura II.B.6.3: Primera mensaje de Chatbot

Al introducir las preguntas en la parte inferior del campo "Type a message" en la página “d-studies” y hacer clic en el botón “Send”, se genera un texto que proporciona información sobre las respuestas a las preguntas formuladas.

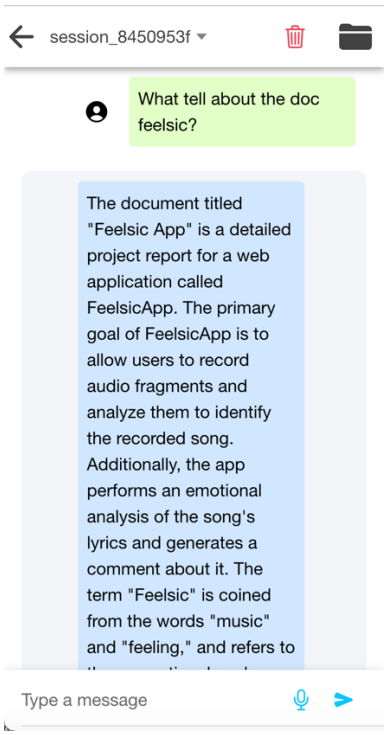


Figura II.B.6.4: Discución con chatbot

El selector de sesiones en la interfaz se utiliza de la siguiente manera: primero, se hace clic en la "New Session" en la parte superior de la pantalla. Luego, se elige la sesión deseada mediante un clic y se confirma la selección haciendo clic en "OK". Este proceso permite recuperar la sesión seleccionada en la interfaz, mostrando el historial de mensajes anteriores de esa sesión de chatbot. Además, se puede consultar los documentos aportados anteriormente en esa sesión.

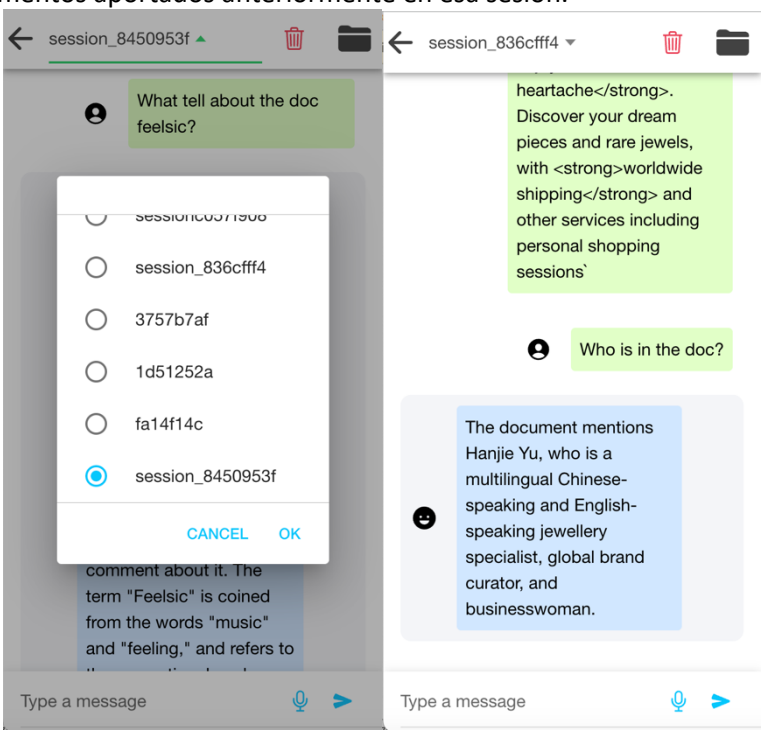


Figura II.B.6.5: selector de session

En la página, el dictado de voz funciona de la manera, mantiene presionado el botón del icono de micrófono para comenzar a hablar. Cuando se suelta el botón, se envía la solicitud con el audio capturado al modelo de reconocimiento de voz, que procesa el contenido y genera el mensaje correspondiente. Este mensaje se coloca automáticamente en el campo de "Type a message".

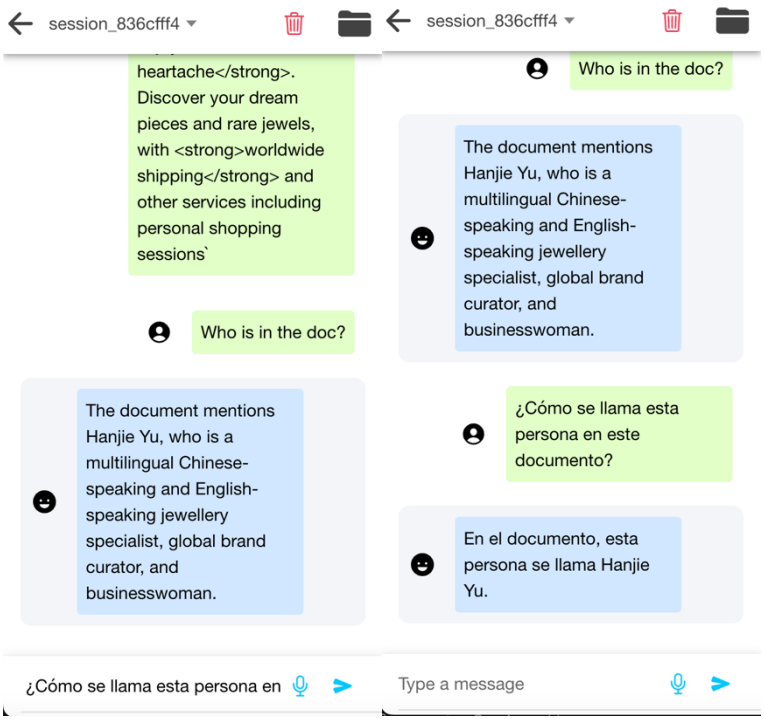


Figura II.B.6.6: El dictador de voz

II.B.7 Text generation

En la página de Studies, se ha integrado un modal accesible a través del botón situado en la esquina inferior izquierda. Este modal permite el uso de la generación de texto basado en un contexto previo proporcionado por el usuario.

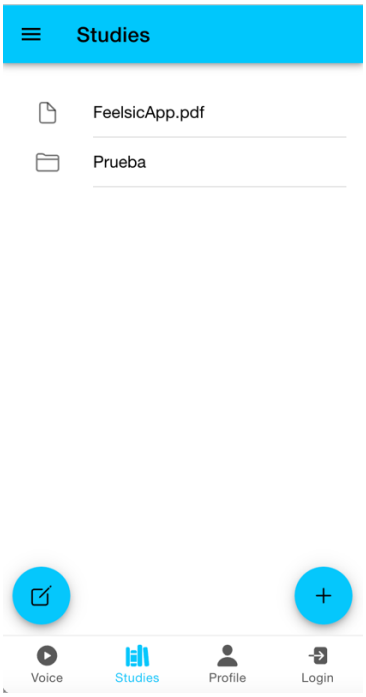


Figura II.B.7.1: Modal de TG

Accedemos al modal mediante un clic en el botón correspondiente. En el modal, se introduce el contexto deseado y se envía al modelo mediante un clic en el botón "Send". Esto permite que el modelo genere textos basados en el contexto proporcionado.

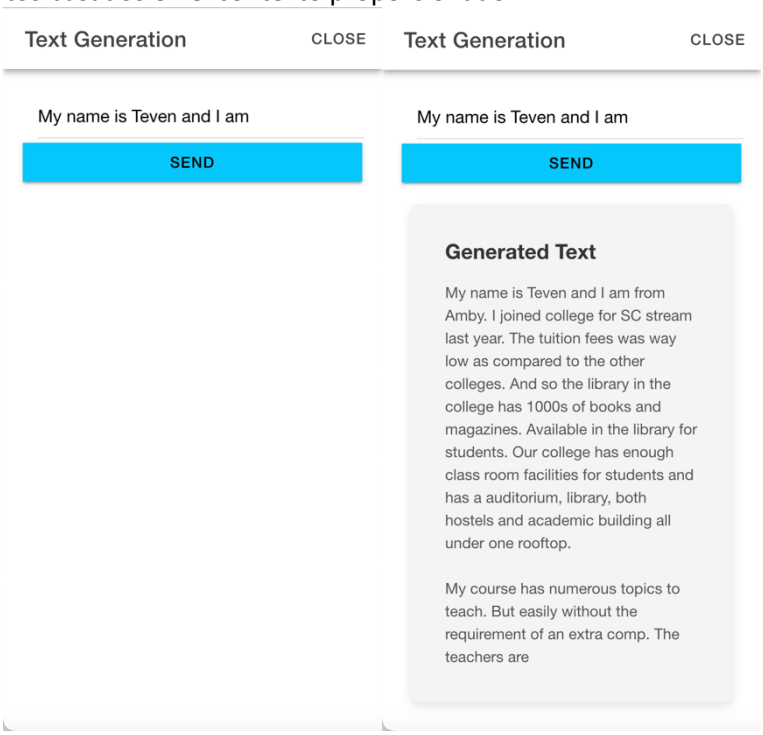


Figura II.B.7.2: Uso de text generation