Universidad de Valladolid

Máster universitario

# Ingeniería Informática



Trabajo Fin de Máster

## TuVozATexto: Servicio web para la conversión de voz a texto

Realizado por **Gino Jesús Quintana Angulo**

✠✠✠

**Universidad de Valladolid**

**19 de septiembre de 2024**

Tutor: Valentín Cardeñoso Payo

# Universidad de Valladolid


## Máster universitario en Ingeniería Informática

D. Valentín Cardeñoso Payo, profesor del departamento de Entornos de Computación Avanzada y Sistemas de Interacción Multimodal (ECA-SIMM), área de Lenguajes y Sistemas Informáticos.

**Expone**:

Que el alumno D. Gino Jesús Quintana Angulo , ha realizado el Trabajo final de Máster en Ingeniería Informática titulado "TuVozATexto: Servicio web para la conversión de voz a texto ".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Valladolid, 19 de septiembre de 2024

Vº. Bº. del Tutor:

Valentín Cardeñoso Payo

Vº. Bº. del Coordinador del Máster:

D. Jesús Vegas Hernández

## Resumen

El objetivo de este proyecto es el desarrollo de un servicio web que permita la transcripción de voz a texto (*speech to text*) de manera precisa y con un rendimiento aceptable. Para lograrlo, se emplearán tecnologías ASR como Whisper y *Wav2Vec*. Además, este servicio establecerá un marco de referencia para la construcción de sistemas similares en este campo tecnológico en constante evolución. El desarrollo del servicio se llevará a cabo siguiendo la metodología ágil *Scrum*, dividiendo el proceso en iteraciones incrementales (*sprints*). Se destacan hitos como la implementación de microservicios usando Docker y Docker Compose, la creación de un prototipo funcional y la mejora continua del servicio, la calidad y velocidad de transcripción. El servicio se diseñará con las siguientes características: escalable, modular y accesible.



## Descriptores

Transcripción, Conversión de voz a texto, *Speech to text*, Tecnologías conversión de voz a texto, Reconocimiento automático del Habla, ASR

## Abstract

The objective of this project is the development of a web service that allows speech to text conversion in an accurate way and with an acceptable performance. To achieve this, ASR technologies such as Whisper and Wav2Vec will be used. In addition, this service will establish a reference framework for the construction of similar systems in this constantly evolving field of technology. The development of the service will be carried out following the agile Scrum methodology, dividing the process into incremental iterations (sprints). Milestones such as the implementation of microservices using Docker and Docker Compose, the creation of a functional prototype and the continuous improvement of the service, the quality and speed of transcription are highlighted. The service will be designed with the following characteristics: scalable, modular and accessible.



## Keywords

Transcription, Speech to text, Speech to text technologies, Automatic Speech Recognition, ASR

# Índice general

# Acrónimos

**AMQP**

Advanced Message Queuing Protocol (Protocolo Avanzado de Colas de Mensajes). 19, 63

**API**

Application programming interface. 1, 2, 12, 13, 19, 21, 30, 31, 35–38, 41–44, 75, IX, XI

**ASR**

Automatic Speech Recognition (Reconocimiento Automático del Habla). 2, 4–6, 8, 12, 22, 24, I–III, IX

**BOE**

Boletín Oficial del Estado. 34

**CAGR**

Compound Annual Growth Rate (Tasa de Crecimiento Anual Compuesta). 1

**CLST**

Centre for Language and Speech Technology. 12

**CNN**

Convolutional Neural Network (Red Neuronal Convolucional). 6

**CRUD**

Create, Read, Update, Delete (Crear, Leer, Actualizar, Eliminar). 16

**CSS**

Cascading Style Sheets (Hojas de estilo en cascada). 62

**DAO**

Data Access Obkect (Objeto de acceso a datos). 58

**DNN**

Deep Neural Network (Red Neuronal Profunda). 7

**GDPR**

General Data Protection Regulation (Reglamento General de Protección de Dato). 34

**GMM**

Gaussian Mixture Model (Modelo de Mezcla Gaussiana). 7

**GPU**

Graphics Processing Unit (Unidad de Procesamiento Gráfico). 63

**HMM**

Hidden Markov Model (Modelo Oculto de Markov). 7

**HTTP**

Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto). 58, 63

**IA**

Inteligencia Artificial (Artificial Intelligence). 5

**IDE**

Integrated Development Environment (Entorno de Desarrollo Integrado). 16

**JAR**

Java ARchive (Archivo Java). 59

**JPA**

Java Persistence API. 19

**JSON**

JavaScript Object Notation. 3, 19, 41, 44, 58, 75

**MB**

Megabyte. 74, 75

**MFCC**

Mel Frequency Cepstral Coefficients (Coeficientes Cepstrales de Frecuencias de Mel). 6

**ORM**

Object-Relational Mapping (Mapeo Objeto-Relacional). 19

**REST**

Representational State Transfer. 2, 3, 19, 75

**RNN**

Recurrent Neural Network (Red Neuronal Recurrente). 9

**SO**

Sistema Operativo (Operating System). 20

**STFT**

Short-time Fourier Transform (Transformada de Fourier de Tiempo Corto). 7

**TPU**

Tensor Processing Unit (Unidad de Procesamiento Tensorial). 63

**UE**

Unión Europea. 34

**UML**

Unified Modeling Language (Lenguaje Unificado de Modelado). 16

**URL**

Uniform Resource Locators (Localizador de recursos uniforme). 68, 70, 71

# Índice de figuras

# Índice de tablas

# Acrónimos

**AMQP**

Advanced Message Queuing Protocol (Protocolo Avanzado de Colas de Mensajes). 19, 63

**API**

Application programming interface. 1, 2, 12, 13, 19, 21, 30, 31, 35–38, 41–44, 75, IX, XI

**ASR**

Automatic Speech Recognition (Reconocimiento Automático del Habla). 2, 4–6, 8, 12, 22, 24, I–III, IX

**BOE**

Boletín Oficial del Estado. 34

**CAGR**

Compound Annual Growth Rate (Tasa de Crecimiento Anual Compuesta). 1

**CLST**

Centre for Language and Speech Technology. 12

**CNN**

Convolutional Neural Network (Red Neuronal Convolucional). 6

**CRUD**

Create, Read, Update, Delete (Crear, Leer, Actualizar, Eliminar). 16

**CSS**

Cascading Style Sheets (Hojas de estilo en cascada). 62

**DAO**

Data Access Obkect (Objeto de acceso a datos). 58

**DNN**

Deep Neural Network (Red Neuronal Profunda). 7

**GDPR**

General Data Protection Regulation (Reglamento General de Protección de Dato). 34

**GMM**

Gaussian Mixture Model (Modelo de Mezcla Gaussiana). 7

**GPU**

Graphics Processing Unit (Unidad de Procesamiento Gráfico). 63

**HMM**

Hidden Markov Model (Modelo Oculto de Markov). 7

**HTTP**

Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto). 58, 63

**IA**

Inteligencia Artificial (Artificial Intelligence). 5

**IDE**

Integrated Development Environment (Entorno de Desarrollo Integrado). 16

**JAR**

Java ARchive (Archivo Java). 59

**JPA**

Java Persistence API. 19

**JSON**

JavaScript Object Notation. 3, 19, 41, 44, 58, 75

**MB**

Megabyte. 74, 75

**MFCC**

Mel Frequency Cepstral Coefficients (Coeficientes Cepstrales de Frecuencias de Mel). 6

**ORM**

Object-Relational Mapping (Mapeo Objeto-Relacional). 19

**REST**

Representational State Transfer. 2, 3, 19, 75

**RNN**

Recurrent Neural Network (Red Neuronal Recurrente). 9

**SO**

Sistema Operativo (Operating System). 20

**STFT**

Short-time Fourier Transform (Transformada de Fourier de Tiempo Corto). 7

**TPU**

Tensor Processing Unit (Unidad de Procesamiento Tensorial). 63

**UE**

Unión Europea. 34

**UML**

Unified Modeling Language (Lenguaje Unificado de Modelado). 16

**URL**

Uniform Resource Locators (Localizador de recursos uniforme). 68, 70, 71

# 1: Introducción

La creciente demanda de soluciones tecnológicas eficientes y accesibles para la interacción humano-máquina ha impulsado un gran interés en la conversión de voz a texto. Esta tecnología no solo se utiliza para transcribir reuniones y conferencias, sino que también mejora la accesibilidad para personas con discapacidades auditivas y abre nuevas vías para la interacción entre humanos y máquinas.

*Este crecimiento se puede observar en los datos del mercado global 1.1, ya que según SNS Insider el tamaño de mercado de las APIs de voz a texto alcanzó los 2.8 mil millones de dólares en 2023 y se espera que alcance los 11.83 mil millones de dolares para 2031 y crezca a una CAGR del 19,2 % para el periodo 2024-2031.* [1]

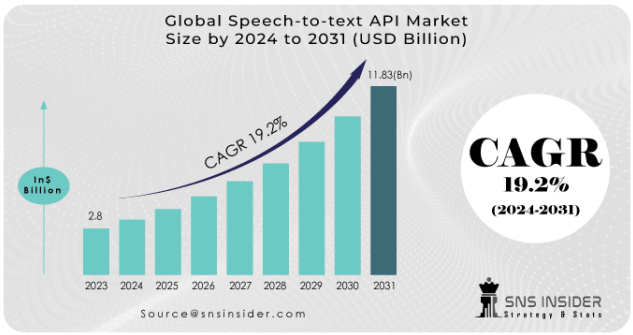


Figura 1.1: Tamaño de Mercado APIs *Speech-to-text.* [1]

El objetivo principal de este proyecto es desarrollar un servicio web que satisfaga la creciente demanda de soluciones de conversión de voz a texto. Al ofrecer un servicio eficiente, preciso y accesible, buscamos mejorar la interacción humano-máquina y facilitar el acceso a la información para todos los usuarios.

El sistema propuesto se basa en tecnologías de reconocimiento automático del habla [8] de vanguardia, como Whisper [9] y Wav2Vec [10] para convertir el audio en texto. Además, se emplearán diversas tecnologías para desarrollar la aplicación web y la API, lo que permitirá realizar solicitudes de transcripción de manera sencilla e intuitiva.

# 2: Objetivos

El objetivo principal del proyecto es desarrollar un servicio web escalable y modular para la conversión de voz a texto, empleando tecnologías de ASR de vanguardia como *Whisper* y *Wav2Vec*. Este servicio no solo proporcionará una herramienta eficiente y precisa para la transcripción de audio, sino que también servirá como referencia para futuros proyectos en este campo en constante evolución.

Para alcanzar este objetivo, se seguirá la metodología ágil *Scrum*, dividiendo el proceso en iteraciones incrementales (*sprints*). Entre los hitos más destacados se encuentran:

- Implementación de microservicios [5] usando *Docker* [11] y *Docker Compose* [12].
- Desarrollo de un *worker* para el proceso de transcripción usando *Python* [13].
- Desarrollo de un API RESTful usando *SpringBoot* [14], para permitir la interacción con el servicio desde diferentes aplicaciones.
- Desarrollo de una aplicación web con Angular [15] para proporcionar una interfaz intuitiva a los usuarios.

El servicio se diseñará con las siguientes características:

- **Escalabilidad**: Capacidad para manejar un aumento en la carga de trabajo.
- **Modularidad**: Facilidad para agregar nuevas funcionalidades o modificar componentes existentes.
- **Accesibilidad**: Interfaz sencilla y clara para usuarios con diferentes niveles de conocimientos técnicos.

En resumen, el objetivo es desarrollar una solución escalable y modular, utilizando tecnologías modernas y metodologías ágiles. El resultado será un prototipo de servicio útil tanto para investigadores como para usuarios finales.

Este problema plantea desafíos técnicos y de diseño, además se establecen las siguientes tareas y restricciones:

- Utilizar *Swagger* [16] para diseñar, construir, documentar y utilizar servicios web RESTful.
- Implementar microservicios para facilitar la escalabilidad y modularidad del sistema, usando *Docker* y *Docker Compose* como herramientas principales.
- Desarrollar un prototipo funcional que acepte archivos de audio como entrada, permita seleccionar el tipo de tecnología a emplear y genere una transcripción en formato JSON, incluyendo en su estructura: el texto generado, la confianza en la transcripción y el tiempo de procesamiento.
- Garantizar que la transcripción es anónima y que solo se guarde información sobre el archivo del que se desea obtener la transcripción y las estadísticas de procesamiento.

# 3: Conceptos teóricos

En la actualidad, el uso de tecnologías capaces de convertir la voz en texto es de gran ayuda para la interacción humano-máquina. Desde asistentes virtuales hasta transcripción de reuniones, estas herramientas se han vuelto indispensables para muchas personas a la hora de interactuar con los diferentes dispositivos.

Para comprender y desarrollar el proyecto de conversión de voz a texto, es necesario abordar varios conceptos teóricos relacionados con las tecnologías de ASR y metodologías de desarrollo ágil.

## 3.1. Panorámica histórica

La historia de la transcripción y el reconocimiento de voz abarca un gran recorrido desde la antigüedad hasta las más avanzadas tecnologías digitales de hoy en día, reflejando la evolución de la comunicación y la tecnología.

En la antigüedad, los escribas egipcios ya documentaban la información manualmente. Con el tiempo, la transcripción se expandió a diversas áreas como la religión, la medicina y la ciencia, siendo esencial para la preservación de conocimiento. En la Edad Media, los monjes copiaban textos religiosos a mano, preservando el conocimiento de la época. Con la invención de la imprenta por Johannes Gutenberg en el siglo XV, la reproducción de textos se volvió más eficiente.

El siglo XX trajo consigo una revolución en la transcripción gracias a las computadoras y los procesadores de texto. Herramientas especializadas permitieron transcripciones más rápidas y precisas. Hoy en día, la combinación de transcripción manual y automatizada es esencial para mantener la calidad y la integridad de la información [17].

El ASR también experimentó un gran avance en el siglo XX. IBM revolucionó este campo tecnológico con "Shoebox."[en] 1961, el primer sistema de reconocimiento de voz que entendía comandos simples. Fue desarrollado por William C. Dersch y marcó un hito en la interacción humano-máquina [18]. Proyectos como el sistema "Harpy"desarrollado en la Universidad Carnegie Mellon, que podía comprender hasta 1,011 palabras y productos

comerciales como "Dragon Dictate"de la empresa Dragon Systems (1990) y "ViaVoice"de IBM (1997) popularizaron esta tecnología.

Con la llegada de la IA y el aprendizaje profundo en el siglo XXI, el reconocimiento automático del habla experimentó mejoras significativas. Sistemas como Siri de Apple, Alexa de Amazon y Google Assistant son capaces de entender y procesar el lenguaje natural con alta precisión. IBM también ha avanzado con su sistema Watson, utilizado en aplicaciones médicas y de servicio al cliente. [18, 19]

Modelos más recientes como Whisper (OpenAI) y Wav2Vec (Facebook) basados en transformadores, han revolucionado el campo al manejar grandes cantidades de datos y aprender representaciones contextuales del habla.

## 3.2. Reconocimiento Automático del Habla (ASR)

El ASR, es una tecnología que permite convertir el habla humana en texto escrito. Los sistemas ASR se basan en modelos matemáticos y algoritmos de aprendizaje automático para interpretar y transcribir el habla humana. [8]

El ASR se inicia cuando se captura la señal de audio que contiene el habla humana en un archivo. Esta señal de voz se compone de sonidos fundamentales denominados fonemas, y la manera en la que se pronuncian las palabras se le denomina léxico de pronunciación.

Cada idioma tiene un conjunto específico de fonemas reconocidos. Por ejemplo:

- El español tiene 24 fonemas. Donde la palabra casa se descompone en 4 fonemas /k/ /a/ /s/ /a/.
- El inglés tiene 44 fonemas. Donde la palabra *house* se descompone en /h/ /a/ /u/ /s/

Un sistema típico de reconocimiento del habla se compone de un extractor de características (*front-end* acústico), que analiza y extrae información relevante. El decodificador usa esta información junto con un modelo acústico, un modelo de lenguaje y un léxico, para determinar la secuencia de palabras más probables que correspondan con la señal de audio. [2, 20] *Además, un sistema ASR tendrá que ser capaz de hacer frente a distintas fuentes de conocimiento, como son la acústica, fonética, léxico, la sintaxis, semántica, etc. Donde inevitablemente se enfrentará a ambigüedades, incertidumbres y errores.* [21]

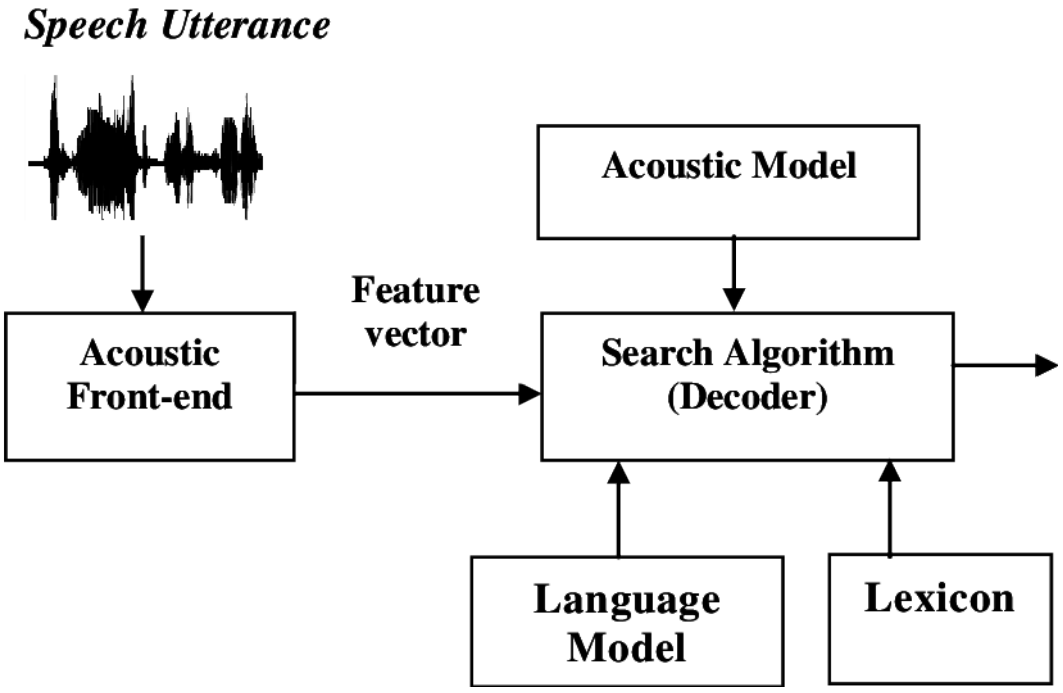


Figura 3.2: Arquitectura de un ASR. [2]

## Extractor de Características

El extractor de características es el encargado de procesar la señal de voz y de obtener las características acústicas. La extracción de características suele tener tres etapas: en la primera se analiza la señal de audio y se generan características que representan el espectro de frecuencia en intervalos de tiempo, en la segunda etapa se compilan las características en un vector con características estáticas y dinámicas y en la última etapa (opcional), se transforma el vector características en un vector más compacto y robusto que se proporciona al decodificador.

Hay muchos métodos de extracción de características, uno de los más utilizados son los MFCC, estos se calculan a través de una serie de pasos que incluyen la transformación de Fourier, el filtrado de Mel y la toma del logaritmo del espectro de potencia.

Hoy en día se utilizan otros métodos de extracción de características, en modelos recientes como Wav2vec, se utilizan CNN para extraer características directamente de la señal de audio.[10]

*Las CNNs permiten reducir la dimensionalidad de las características extraídas de ventanas temporales del audio, representándolas en un espacio de menor dimensión mediante una operación matemática llamada convolución. Esta operación se encarga de transformar los datos de entrada, de tal manera que ciertas características y patrones se vuelven más dominantes en la salida al tener un valor numérico más alto asignado.* [21]

Por otro lado, Whisper utiliza la STFT para obtener un espectrograma, que representan las frecuencias de audio a lo largo del tiempo. Posteriormente se convierte el espectrograma de frecuencia en un espectrograma logarítmicos de Mel, que mapea las frecuencias a una escala perceptualmente más adecuada para el oído humano. [22, 23]

## Modelos acústicos

Los modelos acústicos representan la relación entre las unidades fonéticas (fonemas) del sonido de cada palabra y las características acústicas observadas en la señal de audio. Es decir, modelan cómo se pronuncian los sonidos del lenguaje en términos de las propiedades físicas de la señal sonora.

El HMM, ha sido uno de los modelos estadísticos más utilizados para construir modelos acústicos. El modelo acústico se crea a partir de una gran base de datos denominada corpus del habla y utilizan algoritmos de entrenamiento especiales para crear representaciones estadísticas de cada fonema.

Para mejorar la precisión, se han utilizado modelos mixtos donde se utiliza el modelo DNN y el modelo GMM, los cuales modelan la distribución de las características acústicas en cada estado oculto. Otra aproximación, es la de utilizar una DNN y el modelo HMM que permite obtener un mayor nivel de precisión y el entrenamiento es más simple.

Hoy en día se utilizan otros métodos para construir modelos acústicos, como los transformadores utilizados en Wav2vec y Whisper.

Modelos recientes como Wav2Vec y Whisper utilizan la arquitectura de transformadores. Los transformadores permiten capturar dependencias a largo plazo de las características obtenidas, mediante mecanismos de autoatención que hacen que los modelos sean más robustos y capaces de captar los matices del habla.

## Modelos de lenguaje

Determinan qué secuencias de palabras son gramaticalmente correctas y probables en un idioma dado. Estos modelos se entrenan usando un corpus de texto para estimar probabilidades de n-gramas (secuencias de palabras) y su objetivo es proporcionar contexto a los sistemas de reconocimiento de voz. Aunque reducir la perplejidad (una medida de incertidumbre) en los datos de entrenamiento no siempre mejora el reconocimiento del habla, los modelos lingüísticos ayudan a predecir la siguiente palabra en una secuencia dada. Estos modelos ayudan a mejorar la precisión de la transcripción al considerar el contexto lingüístico.

Los modelos más comunes son los de bigramas y trigramas. Modelos como Wav2Vec y Whisper utilizan arquitecturas de transformadores para construir modelos de lenguaje que son capaces de aprender representaciones contextuales de las palabras a partir de grandes cantidades de datos de texto.

## Decodificador

La fase de decodificación consiste en encontrar la secuencia de palabras más probable $W$ dada una secuencia de observaciones $O$. Este problema se puede resolver con algoritmos de programación dinámica, centrándose en encontrar el mejor camino que coincida con $O$. Es decir, se busca la transcripción textual que mejor explique la señal de audio. [2]

El algoritmo Viterbi se ha utilizado comúnmente para estimar la mejor secuencia de estados para la secuencia de observaciones dada. Para tareas de reconocimiento con vocabulario amplio, se utiliza una búsqueda en haz para considerar solo las palabras con probabilidades de camino por encima de un umbral, acelerando el proceso a costa de la precisión.

Los modelos de lenguaje basados en transformadores, como los utilizados en Wav2Vec y Whisper, incorporan un decodificador que genera secuencialmente la transcripción de texto. El decodificador utiliza un mecanismo de atención para ponderar la importancia de diferentes partes de la secuencia de entrada al predecir la siguiente palabra, lo que permite capturar dependencias de largo alcance y generar transcripciones más coherentes y contextualmente apropiadas.

## Léxico

El léxico proporciona una lista de palabras posibles junto con sus transcripciones fonéticas. En el reconocimiento de voz, un diccionario léxico relaciona cada palabra con su secuencia de fonemas, los cuales son las unidades básicas de sonido en un idioma. Este diccionario es crucial para la conversión de señales de audio en texto escrito, ya que proporciona la base para que el modelo ASR mapee las características del audio a palabras específicas.

## Transformadores

*Los transformadores son una arquitectura de red neuronal utilizada para aprender representaciones útiles de secuencias o conjuntos de datos mediante el modelado de relaciones contextuales entre los elementos. Gracias a su capacidad de capturar dependencias a largo plazo en secuencias, han impulsado avances recientes en áreas como el procesamiento del lenguaje natural, la visión por computadora y el modelado espacio-temporal. A diferencia de arquitecturas anteriores, los transformadores no se basan en recursividad ni convoluciones para procesar datos secuenciales, lo que les permite aprender de manera más eficiente el contexto y el significado de los datos, como las palabras en esta oración.* [24, 25]

*Los Transformers consisten en un Encoder-Decoder (Seq2Seq) [3] con un mecanismo de Auto-Atención [26, 27]. El Encoder recibe una secuencia de entrada que transformará en una representación interna posteriormente usada por el Decoder para transformarla en el tipo de secuencia buscada. A lo largo de toda la arquitectura intervendrá un mecanismo de Auto-Atención. Este se encargará de decidir qué partes de la secuencia son más relevantes*

*con respecto a otras, para así lograr una representación en contexto de la secuencia de entrada.* [22]

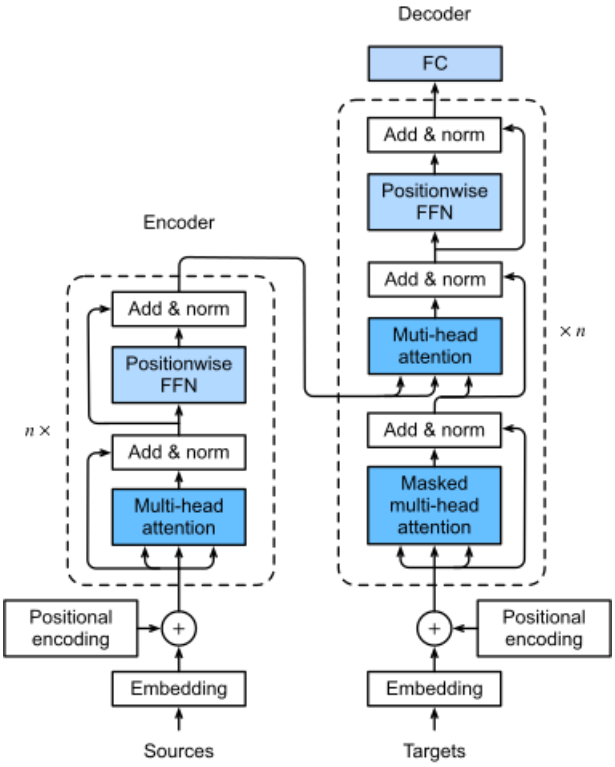


Figura 3.3: Arquitectura *Transformer*. [3]

## Atención

El mecanismo de Atención fue una innovación que originalmente se concibió como una mejora para las RNN de codificador-decodificador aplicadas a sistemas de secuencia a secuencia. [28]

La atención en los transformadores es un mecanismo que permite a un modelo ponderar la importancia de diferentes partes de una secuencia de entrada. Es decir, decide qué palabras o partes de una oración son más relevantes para predecir o generar la siguiente palabra o parte de la salida.

### Atención *multi-head*

La atención *multi-head* es una técnica que consiste en aplicar múltiples mecanismos de atención en paralelo. Cada uno de estos mecanismos, o cabezas, opera de forma independiente, permitiendo al modelo capturar diferentes aspectos de la entrada. Cada cabeza se especializa en un tipo de relación o patrón. Por ejemplo, una cabeza podría

enfocarse en las relaciones sintácticas entre las palabras, mientras que otra podría centrarse en las relaciones semánticas. [29]

#### Atención *self-attention*

La atención *self-attention*, o autoatención, es un mecanismo que permite a un modelo relacionar cada elemento de una secuencia consigo mismo y con todos los demás elementos. Al utilizar la secuencia como consulta, clave y valor, el modelo puede capturar dependencias de largo alcance, es decir, conexiones entre palabras que están distantes entre sí en la secuencia. Esta capacidad es fundamental para tareas de procesamiento del lenguaje natural que requieren comprender el contexto global de un texto. [29]

## 3.3. Metodología Ágil (Scrum)

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar de forma colaborativa, en equipo, y así permitir una entrega incremental de valor en un proyecto. [30]

#### Roles del Equipo

- **Scrum Máster**: Responsable de asegurar que el equipo siga la metodología Scrum.
- **Product Owner**: Define los requisitos del proyecto y prioriza las tareas.
- **Equipo de Desarrollo**: Compuesto por desarrolladores, *testers* y otros roles necesarios para el desarrollo del proyecto.

#### Sprint

Un sprint es un período de tiempo fijo (generalmente de 2 a 4 semanas) durante el cual el equipo trabaja para completar un conjunto específico de tareas o historias de usuario. Al final de cada sprint, se realiza una revisión y retrospectiva para evaluar el progreso y planificar mejoras.

#### Product Backlog

El *Product backlog* es una lista priorizada de objetivos a realizar en el proyecto. Estos objetivos representan la visión y expectativas del proyecto. El *Product Owner* es responsable de mantener y priorizar el *Product Backlog*.

#### Sprint Backlog

Es una lista de tareas seleccionadas del *Product Backlog* que el equipo se compromete a completar durante un sprint. Estas tareas se seleccionan y planifican en la reunión de planificación del sprint.

### Daily Scrum

Es una reunión diaria de corta duración (normalmente de 15 min) en la que el equipo discute el progreso, identifica impedimentos y planifica el trabajo que se realizará durante el día.

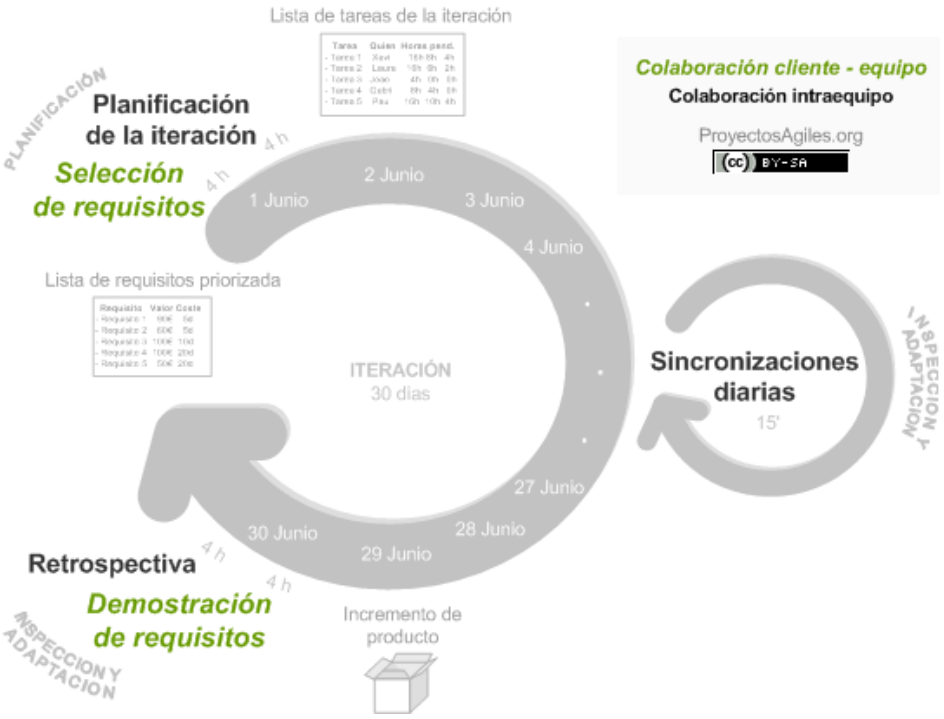


Figura 3.4: Proceso Scrum. [4]

# 4: Trabajos relacionados

Este capítulo está dedicado a revisar los trabajos relacionados con la transcripción de voz a texto. Se describen trabajos académicos que abordan la integración de reconocedores de voz en sistemas móviles y web, así como sistemas comerciales que son tendencia en este campo. Esta revisión de sistemas similares servirá como punto de partida para situar este proyecto dentro del estado del arte.

## 4.1. Trabajos de Fin de Estudios

**Incorporación de un Reconocedor Automáticode Voz local sobre plataformas Android** Este trabajo de fin de grado desarrolla una aplicación Android capaz de reconocer y transcribir palabras y frases dictadas en inglés. Para el desarrollo de este sistema se utilizan herramientas como Android Studio, Kaldi, VOSK y Docker. Kaldi es fundamental a la hora de generar modelos acústicos y lingüísticos necesarios, mientras que VOSK actúa como motor de reconocimiento de voz para realizar transcripciones sin conexión a internet. [31]

**Desarrollo de *frontend* web para grabación de ficheros de audio en tiempo real** Este trabajo de fin de grado se centra en desarrollar un *frontend* que permita la grabación de audio y su envío a un servidor para obtener una transcripción en tiempo real mediante una API de ASR, desarrollada por Cristian Tejedor García en el CLST de Radboud University. La API está disponible desde mayo de 2022 para proyectos de investigación y permite la transcripción en tiempo real de los audios grabados en este proyecto. [32]

## 4.2. Google Speech-to-Text

Google Speech-to-Text es uno de los servicios más avanzados en la conversión de voz a texto. Utiliza modelos de aprendizaje automático entrenados, lo que permite soportar múltiples idiomas y dialectos. Este servicio está integrado con otros sistemas de Google, como

Google Docs o el Asistente de Google, pero también puede usarse de forma independiente a través de la API. Sus principales ventajas son la precisión, especialmente en entornos sin ruidos y con buena calidad de audio. [33]

## 4.3. Azure AI Speech

Azure AI Speech es una solución de vanguardia dentro de la suite de Azure AI, la cual ofrece la capacidad de conversión de voz a texto. Con soporte para varios idiomas y variantes, este servicio ofrece una transcripción precisa y en tiempo real. Con un enfoque en la seguridad y el cumplimiento normativo, Azure AI Speech es ideal para aplicaciones que requieren una transcripción confiable y segura. [34]

## 4.4. IBM Watson Speech-to-Text

IBM Watson Speech to Text es una API basada en la nube que permite la transcripción de voz a texto. Admite transcripción en tiempo real y procesamiento por lotes de archivos de audio, lo que ofrece una alta precisión en varios idiomas. Se puede usar modelos avanzados ya entrenados o permite entrenar modelos de lenguaje para dominios específicos, y puede manejar entornos con ruido o diferentes dialectos. [35]

## 4.5. Amazon Transcribe

Amazon Transcribe es una API en la nube que convierte de forma precisa un audio a texto. Permite realizar transcripciones en tiempo real y a partir de archivos de audio. Además, admite varios formatos de audio y ofrece una alta precisión en múltiples idiomas y dialectos. Se puede usar modelos avanzados ya entrenados o permite entrenar modelos de lenguaje para dominios específicos. [36]

## 4.6. Voice Dictation

Voice Dictation es una aplicación de reconocimiento de voz a texto que permite a los usuarios escribir con su voz en cualquier idioma. Usa el motor de reconocimiento de voz de Google para transcribir tu voz en texto, y los resultados son extremadamente rápidos. Tiene una interfaz sencilla e intuitiva, lo que lo hace fácil de usar para cualquier persona. [37]

# 5: Técnicas y herramientas

El proyecto de conversión de voz a texto se basa en una arquitectura de microservicios y utiliza varias tecnologías y herramientas para su desarrollo, despliegue y mantenimiento.

La elección de la mayoría de las herramientas viene condicionada por su uso constante durante mi desarrollo profesional y otras vienen condicionadas por los requisitos técnicos del proyecto.

## 5.1. Microservicios

La arquitectura de microservicios es un enfoque donde una aplicación se compone de pequeños servicios independientes que se comunican entre sí a través de interfaces bien definidas. Cada microservicio se centra en una funcionalidad específica y puede ser desarrollado, desplegado y escalado de manera independiente. Cada microservicio se puede implementar usando la mejor herramienta para su desarrollo.

*Con los microservicios se mejora la escalabilidad y la disponibilidad del sistema, ya que cada servicio puede ser replicado según la demanda.* [38]

Para la escalabilidad se hace uso de un orquestador como Docker Compose, que permite alojar los servicios en un solo *host* mediante de contenedores, lo que permite un uso más eficiente de los recursos.

Uno de los principales desafíos en la arquitectura de microservicios es la administración y orquestación de los mismos, lo que hace indispensable contar con una cultura DevOps consolidada para facilitar la integración continua y el despliegue automatizado. Además, la configuración y coordinación entre microservicios puede ser compleja, ya que a menudo una sola operación de usuario requiere realizar múltiples llamadas a diferentes servicios, lo que introduce retos en la comunicación y la consistencia de datos distribuidos. [39].

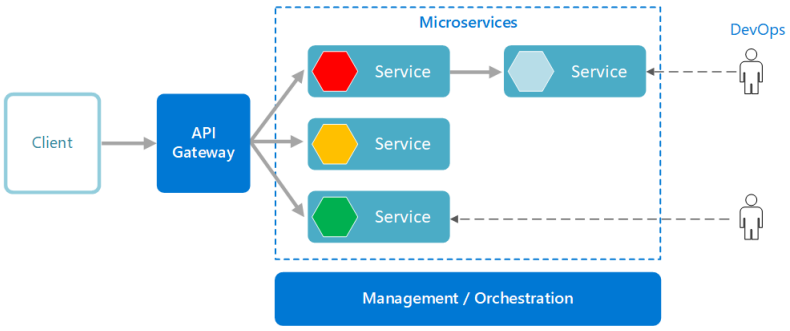


Figura 5.5: Arquitectura de microservicios. [5]

## 5.2. Herramientas de planificación y gestión

En este grupo de herramientas se encontrarán las que sirven para la planificación del proyecto así como su seguimiento, las encargados del control de versiones y la utilizada como repositorio remoto.

### Pivotal Tracker

Pivotal Tracker [40] es una herramienta simple para la gestión de proyectos ágiles que permite el seguimiento de tareas. La elección de esta herramienta viene determinada por el uso durante el proyecto de fin de grado, además de por su facilidad de uso y la gran cantidad de funcionalidades que ofrece.

Es de gran utilidad a la hora de administrar múltiples proyectos. Incluye espacios de trabajo que permiten ver proyectos en paralelo, además de otras funcionalidades.

### Git

Git [41] es un sistema de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es permitir el seguimiento el código de la aplicación.

### GitHub

GitHub [42] es un servicio web de control de versiones y gestor de repositorios basado en Git. Similar a su principal competidor, GitLab.

## 5.3. Herramientas de Diseño y Desarrollo

En este grupo de herramientas encontraremos las utilizadas para la realización de diagramas, los entornos de desarrollo con el que se realiza la aplicación, el entorno para realizar la documentación y las utilizadas para el desarrollo del sistema.

### Astah

Astah [43] una herramienta de diseño de sistemas que soporta UML, diagrama de relación de entidades, diagramas de flujo, CRUD, Diagrama de flujo de datos, Tabla de Requisiciones y Mapas Mentales. En este proyecto se utilizará la versión Astah UML, que es gratuita para estudiantes.

### Eclipse

Eclipse [44] es un entorno de desarrollo, diseñado principalmente para Java (siendo uno de los mejores IDE para su desarrollo) pero que gracias a los *plug-ins*, se puede extender a otros lenguajes y herramientas de desarrollo.

Es un proyecto de código abierto y además cuenta con la ventaja de que se puede utilizar en diversos sistemas operativos como son Windows, cualquier distribución del sistema Linux o Solaris.

Es un proyecto desarrollado por la comunidad *open source* llamada Eclipse Foundation, por lo que posee una licencia de utilización pública.

### Visual Studio Code

Visual Studio Code [45] es un editor de código fuente desarrollado por Microsoft, ampliamente utilizado por desarrolladores debido a su versatilidad y rendimiento. Ofrece soporte para una gran variedad de lenguajes de programación y cuenta con una extensa colección de extensiones que permiten personalizar y ampliar sus funcionalidades. Además incluye soporte para la depuración y control de versiones mediante Git, refactorización de código, compilación de código y muchas más funciones para mejorar el proceso de desarrollo de software.

### Overleaf

Overleaf [46] es un servicio de *LateX* colaborativo en línea con el cual puedes realizar documento utilizando *LateX* en la nube. La documentación del proyecto ha sido elaborada con este

### Balsamiq Mockups

Balsamiq Mockups [47] es una herramienta para la creación de bocetos de interfaces de usuario. Con esta herramienta podemos tener un punto de partida de la visualización de cara al cliente al cliente.

### Figma

Figma [48] es una plataforma de edición gráfica y diseño de interfaces. Además, es una plataforma *online* y colaborativa. Con Figma se puede hacer un poco de todo a nivel de diseño gráfico, desde diseñar páginas web e interfaces gráficas de aplicaciones, o crear publicaciones para redes sociales, hasta la posibilidad de poder crear presentaciones.

### Maven

Maven [49] es una herramienta de código abierto (*open source*) creada en 2001 para la gestión y construcción de proyectos Java, con el objetivo de simplificar los procesos de *build* (compilar y generar ejecutables a partir del código fuente).

Permite la gestión de un proyecto completo, desde la primera etapa en la que se comprueba que el código es correcto, hasta que se despliega la aplicación, pasando por la etapa de pruebas y generación de informes y documentación.

### Python

Python [13] es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un enfoque simple pero efectivo para la programación orientada a objetos. La sintaxis elegante y la tipificación dinámica de Python, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para la creación de *scripts* y el desarrollo rápido de aplicaciones en muchas áreas en la mayoría de las plataformas.

La elección de esta herramienta viene determinada por su compatibilidad con librería de *Transformers* de Hugging Face.

## 5.4. Herramientas de transcripcióni voz a texto (ASR)

En este grupo de herramientas, encontraremos las utilizadas para realizar las transcripciones de audio a texto.

### Transformers

*Transformers* [50] es una biblioteca integral de Hugging Face que proporciona herramientas y modelos preentrenados para una amplia gama de tareas de procesamiento de

lenguaje natural. La biblioteca incluye modelos para tareas como traducción automática, generación de texto, análisis de sentimientos y muchas otras aplicaciones de NLP. Hugging Face ofrece documentación detallada y ejemplos prácticos para ayudar a los desarrolladores e investigadores a integrar estos modelos en sus proyectos.

La biblioteca de HuggingFace implementa modelos de transformadores [29]. Un transformador es una arquitectura de aprendizaje profundo basada en el mecanismo de atención de múltiples cabezas, propuesto en un artículo de 2017 .ªttention Is All You Need", desarrollada por investigadores de Google.[29] El texto se convierte en representaciones numéricas llamadas tokens, y cada token se convierte en un vector al buscar en una tabla de incrustación de palabras. En cada capa, cada token se contextualiza dentro del alcance de la ventana de contexto con otros tokens (sin máscara) a través de un mecanismo de atención de múltiples cabezas paralelo que permite que la señal de los tokens clave se amplifique y los tokens menos importantes se reduzcan.

### Whisper-large-v3

Openai/whisper-large-v3 [51] es un modelo de procesamiento de lenguaje natural desarrollado OpenAI. Es ampliamente utilizado para tareas de transcripción y reconocimiento de voz. Este modelo ha sido optimizado para proporcionar resultados precisos y eficientes en una variedad de contextos de audio y texto. Se puede acceder y utilizar fácilmente a través de la plataforma Hugging Face, que ofrece documentación completa y ejemplos de uso.

### Facebook/wav2vec2-large-960h-lv60-self

Facebook/wav2vec2-large-960h-lv60-self [52] es otro modelo destacado en el campo del procesamiento de lenguaje natural, también desarrollado por Facebook. Este modelo se especializa en el reconocimiento automático del habla, utilizando técnicas avanzadas de aprendizaje automático para convertir audio en texto. Es especialmente útil para aplicaciones que requieren transcripción precisa de grabaciones de audio en diversos idiomas y entornos. Se puede acceder y utilizar fácilmente a través de la plataforma Hugging Face, que ofrece documentación completa y ejemplos de uso.

## 5.5. Herramientas Backend

En esta sección encontraremos las herramientas utilizadas para el desarrollo en el lado del servidor.

### SpringBoot

SpringBoot [14] es una herramienta que acelera y simplifica el desarrollo de microservicios y aplicaciones web con Spring Framework.

Actualmente es el *framework* más popular para Java empresarial, ya que nos ofrece la posibilidad de crear código de alto rendimiento, poco pesado y además reutilizable.

Su principal objetivo es estandarizar, agilizar, manejar y resolver los problemas que puedan ir surgiendo en el transcurso de la programación.

### JPA (Java Persistence API)

JPA es una API que nos ofrece Java para implementar un *framework* ORM, de manera que se pueda interactuar con la base de datos por medio de objetos. Podemos deducir que JPA se encarga de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB). Tendrá como implementación Hibernate [53] y lo integraremos junto a SpringBoot.

### PostgreSQL

PostgresSQL [54] es un sistema de gestión de base de datos relacionales, está orientado a objetos, es multiplataforma y open source. En este caso se ha optado por usar una base de datos PostgreSQL por su uso durante mi formación profesional.

### RabbitMQ

RabbitMQ [55] es un sistema de mensajería basado en el protocolo AMQP. Se utiliza para gestionar y facilitar la comunicación entre diferentes sistemas o aplicaciones a través del envío y recepción de mensajes de manera eficiente y escalable.

### Swagger

Swagger [16] es conjunto de reglas, herramientas y especificaciones para el diseño, construcción y documentación de APIs RESTful. Su objetivo principal es facilitar la creación de documentación interactiva y detallada para las APIs. Swagger usa archivos YAML o JSON para describir todas las operaciones de una API, sus parámetros, respuestas y seguridad.

## 5.6. Herramientas *Frontend*

En esta sección encontraremos las herramientas utilizadas para el desarrollo en el lado del cliente.

### Angular

Angular [15] es un *framework* de JavaScript de código abierto escrito en TypeScript. Su objetivo principal es desarrollar aplicaciones de una sola página.

La arquitectura de una aplicación en Angular se basa en ciertos conceptos fundamentales. Cómo son los módulos y los componentes.

### PrimeNG

PrimeNG [56] es una biblioteca de interfaz de usuario para Angular. Todos los componentes son de código abierto y de uso gratuito bajo la licencia MIT. PrimeNG es desarrollado y mantenida por PrimeTek, una empresa reconocida por su conjunto integral de componentes de interfaz de usuario para varios marcos.

## 5.7. Herramientas de Despliegue

Este grupo incluye herramientas utilizadas para el despliegue del sistema.

### Docker

Docker [11] es una plataforma de código abierto que permite a los desarrolladores crear, desplegar, ejecutar y gestionar contenedores, que son componentes estandarizados y ejecutables que combinan el código fuente de aplicación con las dependencias y las bibliotecas del SO necesarias para ejecutar dicho código en cualquier entorno.

### Docker Compose

Docker Compose [11] es una herramienta para definir y ejecutar aplicaciones multicontenedor. Con Docker Compose, los desarrolladores pueden definir en un solo archivo (generalmente un archivo YAML) todos los servicios que componen una aplicación, simplificando la orquestación de contenedores. Es la clave para lograr una experiencia de desarrollo e implementación optimizada y eficiente.

# 6: Desarrollo del proyecto

En este capítulo, se reseñan exclusivamente los aspectos relevantes del desarrollo del proyecto, dejando los detalles para el apéndice correspondiente. El desarrollo del servicio web de conversión de voz a texto ha sido un proceso lleno de aprendizaje y adaptación, en el que se han enfrentado y superado diversos desafíos técnicos y de diseño. Algunos puntos clave del desarrollo son los siguientes:

## 6.1. Selección de tecnologías

La elección de SpringBoot para el *backend* y Angular para el *frontend* es acertada, ya que ambas tecnologías son ampliamente utilizadas en la industria y por lo tanto tienen soporte a largo plazo, lo que garantiza la sostenibilidad del proyecto en el futuro.

El uso de RabbitMQ para la gestión de la cola de mensajes es un punto clave en el sistema, ya que permite balancear las peticiones de transcripción, asegurando que estas se procesen de manera ordenada y sin perder datos.

La transferencia de los archivos de audio entre servicios se realiza a través de la descarga de archivos. Por ello en la API hay un *endpoint* dedicado a la descarga de archivos para que el *worker* pueda obtener el archivo de audio.

## 6.2. Precisión y rendimiento de los modelos ASR

La precisión de las transcripciones varió dependiendo del modelo utilizado. Whisper y Wav2vec mostraron buenos resultados, aunque el modelo de Wav2vec presentó menos precisión en la transcripción.

El tiempo de procesamiento se ha mantenido menos de 60 segundos para las pruebas realizadas, por lo que se considera una medida que puede reducirse con optimizaciones.

## Cálculo de Confianza en modelos ASR

En este apartado se explica como se realiza el cálculo de la confianza para los modelos utilizados. [57]

En ambos modelos se usa *softmax* para convertir los *logits* en probabilidades normalizadas. Esto permite medir la certeza del modelo sobre cada token del vocabulario. [58]

### Whisper

Whisper genera secuencias de *logits* durante el proceso de generación de texto, además de puntuaciones para cada token. Los *logits* representan las puntuaciones no normalizadas para cada token del vocabulario.

```
outputs = model.generate(**input_features, output_scores=True,
↪  return_dict_in_generate=True)
```

Se aplica *softmax* a los *logits* para obtener las probabilidades normalizadas. Después, para cada token en cada paso de generación, se determina la probabilidad máxima. Finalmente, se calcula la confianza promediando las probabilidades máximas obtenidas a lo largo de todos los pasos de generación.

```
scores = outputs.scores
confidences = []
for score in scores:
    probabilities = torch.softmax(score, dim=-1)
    max_probabilities, _ = torch.max(probabilities, dim=-1)
    confidences.append(max_probabilities.mean().item())
confianza_media = round(sum(confidences) / len(confidences), 3)
```

### Wav2Vec2

*Wav2Vec2* produce *logits* para la secuencia de entrada completa, donde cada *logit* representa la puntuación para cada token en el vocabulario.

```
logits = model(input_values).logits
```

Se aplica *softmax* a los *logits* para obtener las probabilidades normalizadas. Luego, se determina la probabilidad máxima para cada token en la secuencia. Finalmente, se calcula la confianza realizando promediando las probabilidades máximas obtenidas a lo largo de todos los tokens generados.

```
probabilities = torch.softmax(logits, dim=-1)
max_probabilities, _ = torch.max(probabilities, dim=-1)
confianza_media = round(max_probabilities.mean().item(), 3)
```

Las principales diferencias son:

- **Whisper:** Realiza la generación de texto en pasos secuenciales y calcula la confianza para cada paso de generación. Esto implica un proceso iterativo donde cada paso puede influir en la confianza calculada.
- **Wav2Vec2:** Calcula la confianza directamente a partir de la secuencia completa de *logits*. No hay pasos intermedios en el cálculo, ya que el modelo produce una secuencia completa en una sola pasada.

### Cálculo del tiempo de procesamiento

Para el cálculo del tiempo de procesamiento solo se tiene en cuenta el tiempo que tarda cada modelo en obtener el resultado.

```
startTime = time.time()
transcription, confianza_media = process_audio(filePath, technology)
endTime = time.time()
elapsedTime = round(endTime - startTime, 3)
```

## 6.3. Escalabilidad y Arquitectura de Microservicios

La adopción de una arquitectura de microservicios permite una mayor flexibilidad y escalabilidad. Sin embargo, introduce complejidades en la configuración, gestión de dependencias y despliegue. La configuración inicial y el mantenimiento de la infraestructura requieren un conocimiento avanzado de las tecnologías utilizadas.

Se usa Docker Compose para la orquestación de los microservicios, lo que permite un ecosistema estable para el correcto funcionamiento del sistema.

## 6.4. Experiencia del Usuario

La interfaz de usuario desarrollada con Angular recibió comentarios positivos por su diseño simple y efectivo, lo que facilita la interacción de los usuarios con el servicio.

Ha surgido la necesidad de implementar transcripciones en tiempo real, lo cual requerirá una arquitectura más sofisticada y posiblemente el uso de tecnologías de *streaming*.

## 6.5. Pruebas del sistema

En las pruebas manuales se han utilizado los mismos archivos de audio para ambos modelos, observando los resultados obtenidos y utilizando únicamente el procesador.

En ambos casos, se ha logrado una transcripción bastante buena. Sin embargo, se observa que Whisper presenta un grado de precisión mayor que Wav2vec, Además, se ha visto que la confianza varía dependiendo del modelo utilizado para distintos archivos.

Por último, se ha observado que el tiempo de procesamiento es menor en Wav2vec. Esta diferencia radica en la naturaleza y la arquitectura de cada modelo, ya que Whisper esta destinado a tareas más complejas, como la traducción, transcripción multi-idioma y generación de texto en diferentes contextos, lo que implica una mayor complejidad computacional. Por otro lado, Wav2vec es un modelo que se enfoca exclusivamente en tareas de ASR y no requiere la misma complejidad de generación iterativa, por lo que está optimizado para una tarea más directa y eficiente en términos de procesamiento.

Los archivos de audio utilizados en las pruebas se encuentran en la carpeta compartida: Link carpeta compartida.

Los archivos específicos utilizados son: Audio 1 (‘conference.wav‘) y Audio 2 (‘speech.wav‘).

| **Archivo** | **Resultado** | **Método** | **Tiempo requerido** | **Confianza** |
|---|---|---|---|---|
| Audio 1 | *This is Peter. This is Johnny. Kenny. And Josh. We just wanted to take a minute to thank you.* | Whisper | 13.703 s | 90.3 % |
| Audio 1 | *MISSUS PETER MISSUS JOHNNY KENNY AND JO WE JUST WANTED TO TAKE A MINUTE TO THANK* | Wav2vec | 2.498 s | 94.9 % |
| Audio 2 | *the nearest, said the district doctor, is a good Italian abbe who lives next door to you. Shall I call on him as I pass?* | Whisper | 15.071 s | 98.1 % |
| Audio 2 | *THE NEAREST SAID THE DISTRICT DOCTOR IS A GOOD ITALIAN ABBE WHO LIVES NEXT DOOR TO YOU SHALL I CALL ON HIM AS I PASS* | Wav2vec | 2.836 s | 99.8 % |

Tabla 6.1: Resultados de transcripción para los archivos de audio.

# 7: Conclusiones y Líneas de trabajo futuras

## 7.1. Conclusiones

El trabajo ha cumplido con los objetivos planteados, desarrollando una solución escalable, modular y eficiente para la conversión de voz a texto.

La orquestación de los microservicios con *Docker Compose* permite una alta escalabilidad y flexibilidad para el despliegue del sistema.

La implementación de una interfaz de usuario accesible y fácil de usar ha logrado que el servicio sea adecuado para una amplia variedad de usuarios, incluyendo aquellos con discapacidades auditivas.

La integración de modelos avanzados como Whisper y Wav2Vec ha sido fundamental para alcanzar los niveles de precisión y rendimiento de la transcripción de voz.

El sistema es una base sólida para futuras mejoras y nuevas implementaciones, como la introducción de la conversión de voz a texto transcripción en tiempo real, nuevos modelos de conversión de voz a texto, entre otros.

La adopción de una metodología ágil como Scrum ha permitido gestionar de manera eficiente el proyecto, garantizando la entrega continua y adaptándose a las necesidades de los usuarios.

En general, la combinación de tecnologías de conversión de voz a texto, una arquitectura escalable y la gestión de proyecto mediante Scrum ha dado como resultado un servicio que cumple con los requisitos abordados en los distintos *sprints*. Este proyecto sirve como base para futuras implementaciones y mejoras, asegurando que el servicio pueda seguir evolucionando para satisfacer las necesidades de los usuarios. La experiencia adquirida y las lecciones aprendidas en el desarrollo me han resultado valiosas para abordar futuros proyectos de conversión de voz a texto o realizar otras aplicaciones con modelos de inteligencia artificial.

## 7.2. Líneas de trabajo futuro

Debido al tiempo limitado para realizar el trabajo de fin de máster, quedan pendientes historias de usuario por cubrir. Estas historias quedan reflejadas en la pila de historias de usuario definida como *Icebox* en Pivotal Tracker, que se compone de todas aquellas historias del *Backlog* que no se implementaron en el último sprint, como podemos ver en la siguiente figura 7.6

La funcionalidad restante son aquellas historias destinadas a la consulta del estado o resultado de un proceso que se ha enviado a la cola de mensajes y la posibilidad de hacer transcripciones en tiempo real.

Una posible mejora sería la mejorar la calidad y la velocidad de las transcripciones, ya sea ajustando los microservicios, mejorando el servidor dedicado a las transcripciones o realizando ajustes en los modelos, como la optimización de hiperparámetros mediante *fine-tuning*, la mejora del procesamiento de los datos de entrada, el uso de hardware optimizado que mejoren el rendimiento y en su caso, el tiempo de entrenamiento, entre otros.

Otra mejora sería optimizar la transferencia de archivos de audio utilizando una carpeta compartida entre servicios, aprovechando los volúmenes proporcionados por el entorno Docker.

Las funcionalidades pendientes de implementar se podrán desarrollarse en un futuro, ya que este proyecto puede ser modificado por cualquier interesado en continuar con el proyecto.

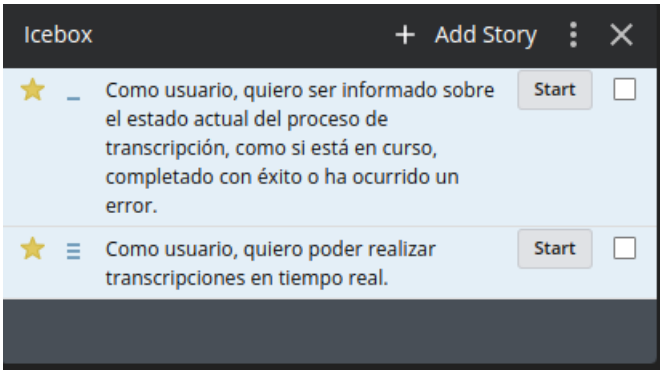


Figura 7.6: *Icebox* de Pivotal Tracker

# Apéndices

*Apéndice* $A$

# Plan de Proyecto

## A.1. Introducción

La realización del proyecto está determinada por una metodología ágil basada en SCRUM [4] con ciertas modificaciones para adaptarlas a las limitaciones del equipo y la propia asignatura. Por lo tanto en este marco de trabajo, los requisitos de usuario pasan a ser historias de usuario. Dichas historias se repartirán a lo largo del proyecto, en intervalos de tiempo denominados *sprints*.

## A.2. Planificación temporal

El comienzo del proyecto se ha fijado para el 08 de abril de 2024, teniendo en cuenta el calendario laboral. Se establecen 5 iteraciones de 2 semanas de duración por cada iteración y la última iteración se considera opcional. La fecha prevista de finalización del proyecto será el 21 de junio de 2024.

El proyecto se desarrollará a lo largo de 10 semanas, con un número de horas estimadas para la realización de cada *sprint* como se muestra en la A.1.

| Sprints 1 | Horas requeridas |
|---|---|
| Sprint 1 | 68h |
| Sprint 2 | 68h |
| Sprint 3 | 68h |
| Sprint 4 | 68h |
| Sprint 5 | 68h |
| **Total** | 340h |

Tabla A.1: Estimación en horas por iteración.

Dado que se trabajará en un marco de trabajo iterativo e incremental, donde los requisitos evolucionan en función de las necesidades del proyecto, se trabaja colaborativamente en equipo y se mantiene una relación con el cliente. Se realizarán entregas parciales y regulares del producto final, donde al comienzo y al final de cada *sprint* se realizarán reuniones para mejorar la organización y aumentar la comunicación entre el equipo.

Por lo tanto, las reuniones denominadas *dailies scrum meeting* no se realizarán diariamente si no que se realizarán a pedido de una tutoría. Por otro lado, el *sprint review* y el *sprint retrospective* se hacen al finalizar cada *sprint*, tras presentarle la funcionalidad realizada en el *sprint* al tutor del proyecto.

### Calendarización

Para el desarrollo completo del proyecto, se han estimado 5 iteraciones (*sprints*), cada uno con un número de horas estimadas de duración. Se establecen los plazos estimados de entrega de cada iteración, como se muestra a continuación.

| Sprints | Fecha finalización |
|---|---|
| Sprint 1 | 22 de abril de 2024 |
| Sprint 2 | 7 de mayo de 2024 |
| Sprint 3 | 22 de mayo de 2024 |
| Sprint 4 | 6 de junio de 2024 |
| Sprint 5 | 21 de junio de 2024 |

Tabla A.2: Estimación de entrega por iteración.

En la siguiente tabla se muestra el desglose semanal estimado desde el inicio del proyecto el 22 de abril de 2024 con su *sprint* o iteración correspondiente, así como el final previsto del proyecto.

| Semana | Fechas | Sprint |
|---|---|---|
| 1 | 8 de abril - 15 de abril | 1 |
| 2 | 15 de abril - 22 de abril | |
| 3 | 23 de abril - 30 de abril | 2 |
| 4 | 30 de abril - 7 de mayo | |
| 5 | 8 de mayo - 15 de mayo | 3 |
| 6 | 15 de mayo - 22 de mayo | |
| 7 | 23 de mayo - 30 de mayo | 4 |
| 8 | 30 de mayo - 6 de junio | |
| 9 | 7 de junio - 14 de junio | 5 |
| 10 | 14 de junio - 22 de junio | |
| - | 22 de Junio | Entrega |

Tabla A.3: Calendarización.

### Planificación sprints

#### Sprint 1 (8 de abril - 22 de abril)

- Definición de requisitos y alcance del proyecto.
- Investigación y selección de tecnologías de conversión de voz a texto.
- Configuración del entorno de desarrollo y repositorios de código con GitHub.
- Diseño de la arquitectura física y lógica del sistema.
- Diseño conceptual y lógico de la aplicación.
- Utilización de Swagger para la definición y utilización del API.
- Crear una primera versión de los microservicios con Docker y Docker Compose, API de Springboot con implementación de API de Whisper y uso modelo de prueba de Wav2vec.

#### Sprint 2 (23 de abril - 7 de mayo)

- Modificar diseño lógico para permitir guardar nombre de fichero.
- US-001:
  - Añadir imagen de RabbitMQ a Docker Compose.
  - Implementar un consumidor de colas de mensajes (RabbitMQ) usando Python.
  - Integración de tecnologías de conversión de voz a texto (whisper-large-v3 y wav2vec2-large-960h-lv60-self) con *Transformers* de HuggingFace.
  - Prueba de transcripción utilizando un archivo de audio.
  - Permitir la recepción y guardado de archivos de audio.
  - Permitir generar una respuesta con la transcripción.
  - Configuración de Docker e instalación de dependencias.
  - Añadir configuración de servicio de transcripción en Docker Compose.
  - Pruebas con el consumidor de mensajes, con guardado y generación de transcripción.
- US-002:
  - Modificar el API usando Swagger a través de SpringBoot.
  - Crear cola de mensajes.
  - Implementar el recurso de transcripción.
  - Implementar el recurso de descarga de archivos.

- Configuración de Docker e instalación de dependencias.
- Añadir configuración de servicio de transcripción en Docker Compose.
- Pruebas del API con servicio de transcripción.

**Sprint 3 (8 de mayo - 22 de mayo)**

- US-003:
  - Implementar servicio con Angular y permitir la conexión con el API de SpringBoot.
  - Implementar un *dashboard* que permita subir el archivo de audio y ver la transcripción generada.
  - Implementar la conexión con el API de SpringBoot.
  - Configuración de Docker e instalación de dependencias.
  - Añadir configuración de servicio de transcripción en Docker Compose.
  - Prueba del servicio con API y servicio de transcripción.
- US-004:
  - Añadir editor de texto.
  - Añadir páginas adicionales de documentación, parte legal y contacto.
  - Prueba para ver si el editor muestra la transcripción del archivo.
- US-005:
  - Añadir reproducción de archivo subido.
  - Añadir nombre y peso del archivo.
  - Prueba de reproducción y datos del archivo.

**Sprint 4 (23 de mayo – 6 de junio)**

- Modificar diseño lógico para permitir guardar la confianza y el tiempo de procesamiento.
- Modificar servicios de Angular y Python para que permitan crear la cola y que no se borre al reiniciar los servicios.
- Preparación de documentación técnica y guías de usuario para el servicio.
- US-006:
  - Modificar *worker* de Python para obtener tiempo y confianza de la transcripción.

  - Modificar servicio SpringBoot para obtener tiempo y confianza de la transcripción.
  - Modificar *front-end* para obtener tiempo y confianza de la transcripción.
- US-007:
  - Añadir *spinner* de progreso al subir un archivo.
  - Añadir barra de progreso al procesar un archivo.
  - Preparación para la fase de pruebas beta y la recopilación de retroalimentación del usuario.

**Sprint 5 (7 de junio - 21 de junio): OPCIONALES**

- Añadir la posibilidad de consultar peticiones a través del identificador dado por el sistema.
- Mejora en la calidad y la velocidad de las transcripciones mediante ajustes y optimizaciones en los modelos de conversión, en el cliente o el servidor.
- Pruebas de rendimiento y ajustes en la infraestructura para garantizar la escalabilidad del sistema.
- Incorporación de la capacidad de conversión de voz a texto en tiempo real (*streaming*), una opción sería hacerlo en el cliente utilizando los cortes en el habla del usuario para ir convirtiendo la voz en el servidor siguiendo un orden.
- Despliegue del servicio en un entorno de producción y configuración de monitoreo.

En principio, no se implementará la funcionalidad de registro de usuarios con AppWriter, pero se considerará su incorporación en futuras iteraciones, dependiendo de los recursos necesarios para el servicio y las demandas de los usuarios. Además, en cada sprint se han realizado labores de aprendizaje de las diferentes tecnologías usadas para la corrección de errores o la mejora de los microservicios.

## A.3. Estudio de viabilidad

### Viabilidad económica

En la viabilidad económica del proyecto se evaluará tanto el hardware, el software utilizado, así como el coste de horas trabajadas en el proyecto. A continuación se detallan los principales componentes económicos:

### Costos iniciales

Para el desarrollo del proyecto se ha utilizado el siguiente hardware:

- Ordenador portátil con un coste de 0 €, ya que se ha usado uno propio.
- Máquina virtual para el despliegue del sistema de conversión de voz a texto proporcionada por la escuela con un coste de 0 €.

No se tienen gastos por parte del hardware. El software utilizado es gratuito, no requiere un coste a la hora de utilizarlo, por lo tanto los gastos originados por el software también son nulos.

### Coste recursos humanos

En cuanto a los recursos humanos, ha sido necesario buscar información sobre cuál es la estimación de salario de un Ingeniero de Software en España y basándonos en ellos hacemos una estimación inicial de cuánto va a ser el coste, tomando como unidad de medida horas/hombre.

Se hace una estimación del salario basado según los sueldos proporcionados por los portales de empleo de la A.4.

| | glassdoor | | indeed | | talent.com | |
|---|---|---|---|---|---|---|
| Puesto | Año | Día | Año | Día | Año | Día |
| Ingeniero de software | 34.950 € | 117.78 € | 44.744 € | 264 € | 35.000 € | 135 € |

Tabla A.4: Comparación de salarios en diferentes plataformas.

Teniendo en cuenta que la media de salario nos da como resultado 38.231,33 € y 199,5 € respectivamente. Hacemos una estimación del coste del trabajador teniendo en cuenta todos los gastos requeridos para su contratación según la calculadora de coste de un trabajador [59] se obtienen los costes estimados A.5.

| Gasto | Anual | Mensual |
|---|---|---|
| Salario bruto | 38.231,33 € | 3.185,94 € |
| Complementos | 0,00 € | 0,00 € |
| Contingencias comunes | 9.022,59 € | 751,88 € |
| AT y EP | 860,20 € | 71,68 € |
| Desempleo | 2.102,72 € | 175,23 € |
| Formación Profesional | 229,39 € | 19,12 € |
| FOGASA | 76,46 € | 6,37 € |
| **Coste del empleado** | **50.522,70 €** | **4.210,23 €** |

Tabla A.5: Desglose de salarios y cotizaciones de Ingeniero de Software

Teniendo en cuenta el coste del empleado y las horas del convenio según el BOE que son de 1792 horas [60].

Obtenemos que a la hora un Ingeniero de Software cobra 28,19 €. Dado que la estimación inicial tiene como objetivo hacer 340 horas, en total de recursos humanos tenemos un coste de 340 horas * 28,19 € = 9584,60 €.

## Viabilidad legal

La viabilidad legal asegura que el proyecto cumple con todas las leyes y regulaciones aplicables, minimizando riesgos legales y protegiendo los derechos tanto de los usuarios como de los desarrolladores. Los principales aspectos legales a considerar son:

### Protección de datos y privacidad

Reglamento General de Protección de Datos (GDPR): Cumplimiento con las normativas de la UE sobre protección de datos personales. [61]

En este caso el proyecto no guarda información de los usuarios finales. Sin embargo, si se ampliase esta posibilidad será necesario profundizar en esta área.

### Derechos de autor y propiedad intelectual

Licencias de software: Se usa software y bibliotecas con las licencias permisivas que permiten el uso comercial. [62]

Gestión del contenido de los usuarios: Asegurar que los archivos de audio subidos por los usuarios no infringen derechos de autor y se manejan de acuerdo con las leyes aplicables. [63]

En este caso, no se ha podido implementar una validación de derechos de autor. Ya que se considera complejo, porque implica aspectos legales cómo técnicos que escapan del objetivo del proyecto.

#### Términos de servicio y condiciones de uso

Definición de responsabilidades: Clarificación de los derechos y responsabilidades tanto del proveedor del servicio como de los usuarios. [64]

Se ha introducido los términos y condiciones de uso de la aplicación en la interfaz web para que los usuarios tengan conocimientos sobre ellos.

## A.4. Seguimiento

### Seguimiento sprint 1

#### Objetivo:

Establecer las bases del proyecto y crear una primera versión funcional del *backend* con SpringBoot junto con la configuración inicial del entorno de desarrollo.

#### Milestones:

- Se analizan y establecen los requisitos y objetivos principales del proyecto.
- Se realiza un análisis de las herramientas disponibles y se decide utilizar Whisper y Wav2Vec para la conversión de voz a texto.
- Se configura el entorno de desarrollo y se establece un repositorio de código para el proyecto.
- Se define la arquitectura general del sistema, incluyendo la estructura de los microservicios y su interacción.
- Se utiliza Swagger para definir la documentación de la API, incluyendo los *endpoints* para la conversión de voz a texto.
- Se implementan los servicios básicos utilizando Docker, incluyendo la configuración de imágenes para SpringBoot y PostgreSQL.

#### Todo:

- Refinamiento de la arquitectura, la definición de la API y las historias de usuario.
- Definición y estimación de tareas para las implementaciones.

- Implementación de *backend* para la conversión de voz a texto usando modelos de Whisper y Wav2Vec con Python, donde reciba el archivo y haga la transcripción.
- Realizar pruebas para garantizar el correcto funcionamiento del sistema.
- Mejorar documentación del proyecto añadiendo referencias e investigación de la conversión de voz a texto.

## Seguimiento sprint 2

### Objetivo:

Avanzar en la integración de las tecnologías necesarias para la transcripción de voz a texto, completando las funcionalidades de *backend* y estableciendo la comunicación entre los microservicios mediante colas de mensajes, asegurando que el sistema pueda recibir, procesar y almacenar archivos de audio.

### Milestones:

- Se añade con éxito la imagen de RabbitMQ en Docker Compose y se configura el entorno para que el *backend* pueda comunicarse a través de colas de mensajes.
- Se implementa un consumidor de colas de mensajes utilizando Python, lo que permite que los servicios de transcripción procesen los mensajes recibidos.
- Se integran los modelos Whisper y Wav2Vec de HuggingFace en el sistema, permitiendo la conversión de voz a texto.
- Se realizan pruebas exitosas de transcripción utilizando un archivo de audio, confirmando que los modelos funcionan correctamente dentro del entorno configurado.
- Se habilita la recepción y almacenamiento de archivos de audio en el sistema, permitiendo que estos sean procesados para la transcripción.
- Se implementa la funcionalidad de generación de una respuesta con la transcripción, completando el ciclo de procesamiento del archivo de audio.
- Se configura Docker para incluir todos los servicios necesarios, incluyendo la transcripción y el almacenamiento de archivos.
- Se realizan pruebas con el consumidor de mensajes, verificando que la transcripción se genera y se guarda correctamente en el sistema.
- Se modifica el API utilizando Swagger a través de SpringBoot, permitiendo la creación de *endpoints* para la transcripción y la descarga de archivos.
- Se implementa el recurso de transcripción en el API, que permite enviar archivos de audio para su procesamiento.

- Se añade el recurso de descarga de archivos, permitiendo la transferencia de los archivos de audio.
- Se realizan pruebas del API con el servicio de transcripción, asegurando que los servicios del sistema trabajen en conjunto y sin errores.

**Todo:**

- Refinamiento de la arquitectura, la definición de la API y las historias de usuario.
- Definición y estimación de tareas para las implementaciones.
- Implementación de la funcionalidad de subida de archivos en el *frontend* utilizando Angular.
- Correcciones y mejoras de *backend* de conversión de voz a texto.
- Realizar pruebas para garantizar el correcto funcionamiento del sistema.
- Mejorar documentación del proyecto añadiendo referencias e investigación de las herramientas utilizadas.

## Seguimiento sprint 3

**Objetivo:**

Completar la integración del *frontend* con el *backend*, permitiendo que los usuarios interactúen con el sistema a través de una interfaz web. Se busca implementar las funcionalidades necesarias para la carga de archivos, visualización de transcripciones, y gestión básica del contenido, asegurando una experiencia de usuario fluida y funcional.

**Milestones:**

- Se implementa con éxito el servicio en Angular, estableciendo la conexión con el API de SpringBoot para la transcripción de audio.
- Se desarrolla un *dashboard* donde los usuarios pueden subir archivos de audio y ver las transcripciones generadas.
- Se actualiza la configuración de Docker para incluir el servicio con angular y todas las dependencias necesarias para la integración completa del sistema.
- Se añade el servicio de transcripción en Docker Compose, garantizando la correcta orquestación de los servicios.
- Se realizan pruebas para asegurar que el servicio con angular funciona correctamente con el API y el *worker* en Python. Además, se asegura que se pueden cargar archivos y ver sus transcripciones.

- Se añade un editor de texto en la interfaz web para permitir la edición de las transcripciones generadas.
- Se incluyen páginas adicionales en la aplicación para la documentación del proyecto, información legal y detalles de contacto.
- Se verifica que el editor de texto muestra correctamente la transcripción del archivo de audio y permite su edición.
- Se prueban las nuevas funcionalidades del *dashboard*, como la reproducción de archivos de audio y la visualización de su nombre y tamaño, asegurando que los datos del archivo se muestran correctamente y que la reproducción es funcional.

**Todo:**

- Refinamiento de la arquitectura, la definición de la API y las historias de usuario.
- Definición y estimación de tareas para las implementaciones.
- Implementación de la funcionalidad para obtener el tiempo y la confianza de la transcripción.
- Implementación de la funcionalidad para mostrar los progresos de subida de archivos y el procesamiento de la transcripción.

## Seguimiento sprint 4

**Objetivo:**

Mejorar el sistema de transcripción añadiendo funcionalidades que permitan capturar y mostrar métricas de rendimiento, como el tiempo de procesamiento y la confianza en las transcripciones. Además, optimizar la experiencia de usuario mediante la implementación de indicadores de progreso y preparar el sistema para la fase de pruebas.

**Milestones:**

- Se adapta el diseño lógico para permitir el almacenamiento de la confianza y el tiempo de procesamiento de las transcripciones. Esto facilita la evaluación y el análisis del rendimiento de los modelos de transcripción.
- Se modifican los servicios de Angular y Python para que puedan crear colas de mensajes persistentes, asegurando que estas no se borren al reiniciar los servicios.
- El *worker* de Python es modificado para capturar y registrar tanto el tiempo de procesamiento como la confianza en cada transcripción generada.

- El servicio *Spring Boot* es actualizado para manejar y transmitir estos datos adicionales (tiempo y confianza) al *frontend*.
- El *frontend* en Angular es modificado para mostrar el tiempo de procesamiento y la confianza de cada transcripción al usuario.
- Se añade un *spinner* de progreso al subir archivos, mejorando la experiencia de usuario al proporcionar una indicación visual de la carga en curso.
- Se implementa una barra de progreso que muestra el avance del procesamiento del archivo de audio, permitiendo a los usuarios visualizar que su petición esta en proceso.
- Se prepara la aplicación para las pruebas en el servidor de la escuela y se establecen los mecanismos necesarios para recopilar retroalimentación de los usuarios.
- Se añade la documentación para los programadores y las guías de usuario del servicio.

**Todo:**

Se deja el último sprint para mejoras futuras, estas mejoras son esenciales para que el sistema mejore la calidad en el procesamiento de las transcripciones y además, dote de nueva la funcionalidad del sistema permitiendo la capacidad de transcripciones en tiempo real. Ya que se pretende que el sistema esté preparado para enfrentar nuevos desafíos y demandas adicionales en el futuro.

Una vez desarrolladas las historias de usuario, las que se han logrado implementar quedan en el Done de *Pivotal Tracker* como se ve en la figura .

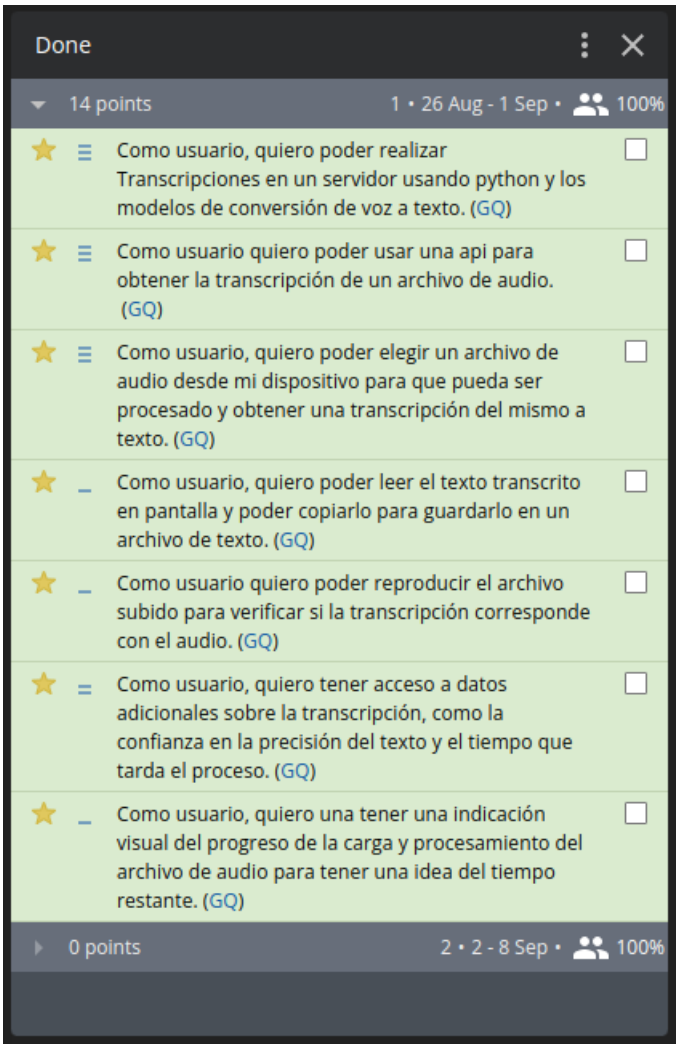


Figura A.1: Done de Pivotal Tracker.

Apéndice $B$

# Especificación de Requisitos

## B.1. Introducción

La especificación de requisitos servirá como guía para el desarrollo y la implementación del software, asegurando que todas las necesidades del usuario y los objetivos del proyecto sean alcanzados.

Dado que el proceso de desarrollo está basado en *Scrum*, los requisitos del proyecto quedarán determinados en forma de historias de usuario. Estas historias representan las necesidades del cliente y estarán distribuidas en intervalos de tiempo denominados *sprints*.

## B.2. Objetivos generales

Los aspectos clave del desarrollo son los siguientes:

- Las entradas de la aplicación web serán archivos de audio a través de un *frontend* en Angular o directamente a través del API implementada con *SpringBoot*.
- Las salidas de la aplicación web serán transcripciones de voz a texto en formato JSON, para ser usados por el *frontend* en Angular o en otros clientes de la API.
- Proceso: Utilización de modelos de conversión de voz a texto para generar transcripciones en formato JSON, con indicadores de precisión y tiempo de procesamiento.

## B.3. Catálogo de requisitos

Todas las historias de usuario que se prevé que serán implementadas para dotar de funcionalidad completa a la aplicación, se van añadiendo en el *Icebox*. Por lo tanto, en este apartado se van añadiendo historias de usuario que se quieren implementar en un

futuro y pueden ir incrementándose debido a las necesidades que tenga el cliente durante el desarrollo del proyecto.

| Historia de usuario |
|---|
| Como usuario, quiero poder realizar Transcripciones en un servidor usando Python y los modelos de conversión de voz a texto. |
| Como usuario quiero poder usar una API, para obtener la transcripción de un archivo de audio. |
| Como usuario, quiero poder elegir un archivo de audio desde mi dispositivo para que pueda ser procesado y obtener una transcripción del mismo a texto. |
| Como usuario, quiero poder leer el texto transcrito en pantalla y poder copiarlo para guardarlo en un archivo de texto. |
| Como usuario, quiero poder reproducir el archivo subido para verificar si la transcripción corresponde con el audio. |
| Como usuario, quiero tener acceso a datos adicionales sobre la transcripción, como la confianza en la precisión del texto y el tiempo que tarda el proceso. |
| Como usuario, quiero una tener una indicación visual del progreso de la carga y procesamiento del archivo de audio para tener una idea del tiempo restante. |
| Como usuario, quiero ser informado sobre el estado actual del proceso de transcripción, como si está en curso, completado con éxito o ha ocurrido un error. |
| Como usuario, quiero poder realizar transcripciones en tiempo real. |

Tabla B.1: Icebox

Una vez analizados los requisitos a través de historias de usuario seleccionamos las que desarrollaremos, definiremos el *Product Backlog* donde las historias de usuario quedarán priorizadas y ordenadas para su implementación en los distintos *sprints*.

| Historia de usuario | Descripción | Prioridad | Estimación |
|---|---|---|---|
| US-001 | Como usuario, quiero poder realizar transcripciones en un servidor usando Python y los modelos de conversión de voz a texto. | 1 | 3 |
| US-002 | Como usuario, quiero poder usar una API para obtener la transcripción de un archivo de audio. | 2 | 3 |
| US-003 | Como usuario, quiero poder elegir un archivo de audio desde mi dispositivo para que pueda ser procesado y obtener una transcripción del mismo a texto. | 3 | 3 |
| US-004 | Como usuario, quiero poder leer el texto transcrito en pantalla y poder copiarlo para guardarlo en un archivo de texto. | 4 | 1 |
| US-005 | Como usuario, quiero poder reproducir el archivo subido para verificar si la transcripción corresponde con el audio. | 5 | 1 |
| US-006 | Como usuario, quiero tener acceso a datos adicionales sobre la transcripción, como la confianza en la precisión del texto y el tiempo que tarda el proceso. | 6 | 2 |
| US-007 | Como usuario, quiero tener una indicación visual del progreso de la carga y procesamiento del archivo de audio para tener una idea del tiempo restante. | 7 | 1 |
| US-008 | Como usuario, quiero ser informado sobre el estado actual del proceso de transcripción, como si está en curso, completado con éxito o ha ocurrido un error. | 8 | 1 |
| US-009 | Como usuario, quiero poder realizar transcripciones en tiempo real. | 9 | 3 |

Tabla B.2: Product Backlog

Estas historias de usuario se estimarán con una puntuación de puntos-historia que determinan el esfuerzo necesario para su realización y llevarán asociada una prioridad con la que serán implementadas. Esta estimación sigue una tendencia lineal (0,1,2,3), dando '0' a las historias de usuario que requieran poco esfuerzo para su implementación y '3' las que requieran más esfuerzo para implementarlas.

## B.4. Especificación de requisitos

En esta sección se hará un estudio detallado de las historias de usuario a implementar ya que como hemos visto durante el seguimiento del proyecto. Para la descripción de las historias tendremos una definición y unos escenarios de aceptación en los que utilizaremos el lenguaje Gherkin [65], que define la estructura y una sintaxis básica para la descripción del comportamiento deseado.

| **US-001** | **Prioridad** | **Estimación** |
|---|---|---|
| | 1 | 3 |
| **Definición** | | |
| Como usuario, quiero poder realizar transcripciones en un servidor usando Python y los modelos de conversión de voz a texto. | | |
| **Escenario 1** | | |
| El archivo de audio se transcribe correctamente | | |
| Dado que tengo un archivo de audio válido<br>Cuando lo subo al servidor<br>Entonces debería recibir una transcripción más precisa del contenido de audio. | | |
| **Escenario 2** | | |
| El archivo de audio no se ha podido transcribir | | |
| Cuando lo subo al servidor<br>Entonces debería recibir un mensaje de error indicando que el archivo no se ha podido transcribir. | | |

Tabla B.3: El usuario podrá realizar transcripciones en un servidor usando Python y los modelos de conversión de voz a texto.

| **US-002** | **Prioridad** | **Estimación** |
|---|---|---|
| | 2 | 3 |
| **Definición** | | |
| Como usuario, quiero poder usar una API para obtener la transcripción de un archivo de audio. | | |
| **Escenario 1** | | |
| El archivo de audio se transcribe correctamente a través de la API | | |
| Dado que tengo un archivo de audio válido<br>Cuando envío una solicitud a la API con el archivo<br>Entonces debería recibir una respuesta con la transcripción más precisa del contenido de audio en formato JSON. | | |
| **Escenario 2** | | |
| El archivo de audio no se puede transcribir a través de la API | | |
| Dado que tengo un archivo de audio inválido<br>Cuando envío una solicitud a la API con el archivo<br>Entonces debería recibir un mensaje de error en la respuesta indicando que el archivo no se ha podido transcribir. | | |

Tabla B.4: El usuario podrá usar un API para obtener la transcripción de un archivo de audio.

| **US-003** | **Prioridad** | **Estimación** |
|---|---|---|
| | 3 | 3 |
| **Definición** | | |
| Como usuario, quiero poder elegir un archivo de audio desde mi dispositivo<br>para que pueda ser procesado y obtener una transcripción del mismo a texto. | | |
| **Escenario 1** | | |
| El archivo de audio se transcribe correctamente | | |
| Dado que tengo un archivo de audio válido<br>Cuando lo selecciono desde mi dispositivo y lo subo<br>Entonces debería recibir una transcripción más precisa del contenido de audio. | | |
| **Escenario 2** | | |
| El archivo no se puede transcribir | | |
| Dado que tengo un archivo de audio inválido<br>Cuando lo selecciono desde mi dispositivo y lo subo<br>Entonces debería recibir un mensaje de error<br>indicando que el archivo no se puede transcribir. | | |

Tabla B.5: El usuario podrá elegir un archivo de audio desde su dispositivo para que pueda ser procesado y obtener una transcripción del mismo a texto.

| **US-004** | **Prioridad** | **Estimación** |
|---|---|---|
| | 4 | 1 |
| **Definición** | | |
| Como usuario, quiero poder leer el texto transcrito en pantalla<br>y poder copiarlo para guardarlo en un archivo de texto. | | |
| **Escenario 1** | | |
| La transcripción se muestra correctamente | | |
| Dado que tengo un transcripción exitosa<br>Entonces se debería mostrar en pantalla y permite copiar el contenido. | | |
| **Escenario 2** | | |
| Se muestra un error de la transcripción | | |
| Dado que la transcripción ha fallado<br>Entonces se debería mostrar en pantalla el error ocurrido y las soluciones posibles | | |

Tabla B.6: El usuario podrá visualizar y copiar la transcripción del archivo de audio procesado.

| **US-005** | **Prioridad** | **Estimación** |
|---|---|---|
| | 5 | 1 |
| **Definición** | | |
| Como usuario quiero poder reproducir el archivo subido<br>para verificar si la transcripción corresponde con el audio. | | |
| **Escenario 1** | | |
| El archivo subido se puede reproducir correctamente | | |
| Dado que la subida del archivo ha sido exitosa<br>Entonces se debería poder escuchar el contenido del audio. | | |
| **Escenario 2** | | |
| El archivo de audio no se puede reproducir | | |
| Dado que la subida del archivo no ha sido exitosa<br>Entonces no se debería mostrar el reproductor de audio. | | |

Tabla B.7: El usuario podrá reproducir el audio subido al servidor.

| **US-006** | **Prioridad** | **Estimación** |
|---|---|---|
| | 6 | 2 |
| **Definición** | | |
| Como usuario, quiero tener acceso a datos adicionales sobre la transcripción,<br>como la confianza en la precisión del texto y el tiempo que tarda el proceso. | | |
| **Escenario 1** | | |
| El archivo de audio se transcribe correctamente | | |
| Dado que el procesamiento del archivo ha sido exitoso<br>Entonces se debería obtener la confianza y el tiempo de procesamiento. | | |
| **Escenario 2** | | |
| El archivo de audio no se puede transcribir | | |
| Dado que el procesamiento del archivo no ha sido exitoso<br>Entonces no se debería mostrar el error y no obtener ningún dato<br>sobre la confianza y el tiempo de procesamiento. | | |

Tabla B.8: El usuario podrá obtener la confianza y el tiempo de procesamiento de la transcripción.

| **US-007** | **Prioridad** | **Estimación** |
|---|---|---|
| | 7 | 1 |
| **Definición** | | |
| Como usuario, quiero una tener una indicación visual del progreso de la carga y procesamiento del archivo de audio para tener una idea del tiempo restante. | | |
| **Escenario 1** | | |
| Se muestra el progreso al subir y procesar el archivo de audio | | |
| Al subir el archivo o procesarlo<br>Entonces se debería mostrar el progreso de las operaciones. | | |
| **Escenario 2** | | |
| No se muestra el progreso al subir o procesar el archivo de audio | | |
| Dado que hay problemas con la subida o el procesamiento del archivo<br>Entonces no se debería mostrar ningún progreso de las operaciones. | | |

Tabla B.9: El usuario tendrá un indicar del progreso de subida y procesamiento del archivo de audio.

| **US-008** | **Prioridad** | **Estimación** |
|---|---|---|
| | 8 | 1 |
| **Definición** | | |
| Como usuario, quiero ser informado sobre el estado actual del proceso de transcripción, como si está en curso, completado con éxito o ha ocurrido un error. | | |
| **Escenario 1** | | |
| Se obtiene el estado actual de la petición de transcripción | | |
| Al subir el archivo o procesarlo<br>Entonces se debería mostrar el progreso de las operaciones. | | |
| **Escenario 2** | | |
| No se puede obtener el estado el archivo de audio | | |
| Dado que hay problemas con la consulta del estado de la petición<br>Entonces se debería mostrar un mensaje de error. | | |

Tabla B.10: El usuario podrá consultar el estado de la petición de transcripción.

| **US-009** | **Prioridad** | **Estimación** |
|---|---|---|
| | 9 | 3 |
| **Definición** | | |
| Como usuario, quiero poder realizar transcripciones en tiempo real. | | |
| **Escenario 1** | | |
| El archivo de audio se transcribe correctamente | | |
| Dado que tengo un archivo de audio válido<br>Cuando lo subo al servidor<br>Entonces debería recibir la transcripción más precisa del contenido de audio mientras se va procesando hasta completar la transcripción. | | |
| **Escenario 2** | | |
| El archivo de audio no se ha podido transcribir | | |
| Cuando lo subo al servidor<br>Entonces debería recibir un mensaje de error indicando que el archivo no se ha podio transcribir. | | |

Tabla B.11: El usuario podrá realizar transcripciones en tiempo real.

*Apéndice* $C$

# Documento de Diseño

## C.1. Introducción

En este apartado se mostrará el modelo conceptual. Se proporciona el modelo lógico de la base de datos para guardar las peticiones, junto con una explicación de qué contiene la tabla y los tipos de datos que almacena. También se mostrará una visión general de la arquitectura de la aplicación, los distintos *frameworks* utilizados y los diagramas de secuencia principales del sistema. Además se indica cómo se desplegará la aplicación en un nodo computacional.

## C.2. Diseño de datos

### Modelo conceptual

Se ha añadido una parte para guardar las peticiones realizadas por los usuarios de aplicación como vemos en la figura C.1. El desarrollo del sistema, parte de este modelo ya que se creará una entidad correspondiente al dominio.

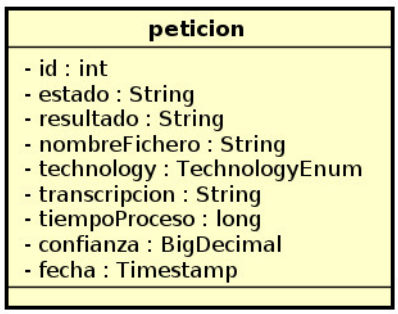


Figura C.1: Modelo conceptual para peticiones.

### Modelo lógico

Una vez aplicadas las técnicas de modelado para la obtención del modelo conceptual, se crea un modelo lógico de datos basado en este modelo para poder almacenar toda la información relacionada con las peticiones recibidas. Esto lo podemos observar en la figura C.2 .

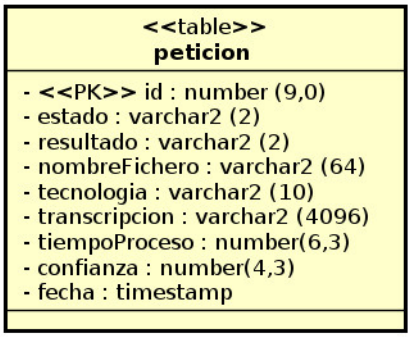


Figura C.2: Modelo lógico para peticiones.

Este modelo lógico almacena información sobre las peticiones que se realizan en la aplicación web. Los campos *id*, *estado*, *resultado*, *nombreFichero*, *tecnologia*, *transcripcion*, *tiempoProceso*, *confianza* y *fecha* permiten almacenar información relevante de la petición.

Una vez obtenido el diseño de la base de datos relacional, se procede a realizar un diseño físico de la base de datos con PostgreSQL, que permita guardar el historial de peticiones.

Se destaca la elección de PostgreSQL como sistema de gestión de bases de datos relacional, debido a su potencia y robustez, además de ofrecer soporte eficiente para la concurrencia de múltiples usuarios. Esta elección garantiza una escalabilidad adecuada a medida que la aplicación crezca. [66, 54]

## C.3. Diseño procedimental

### Patrón Arquitectónico

El patrón arquitectónico que se aplica a nivel lógico es el de tres capas, donde cada capa tiene dependencias con la capa inmediatamente inferior como se puede ver en la figura C.3.

Gracias a este patrón arquitectónico en capas, reducimos el acoplamiento entre componentes de la aplicación, ya que cada capa se encarga de operaciones específicas mejorando así la flexibilidad, el mantenimiento y la escalabilidad.

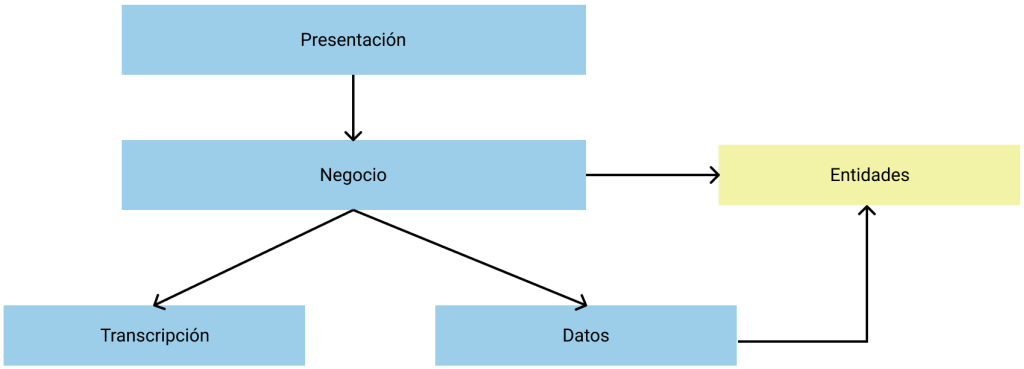


Figura C.3: Arquitectura en 3 capas de la aplicación.

## Diagrama de despliegue

El siguiente diagrama muestra el modelo de despliegue de la aplicación web en dos nodos computacionales: el cliente y el servidor de transcripción.

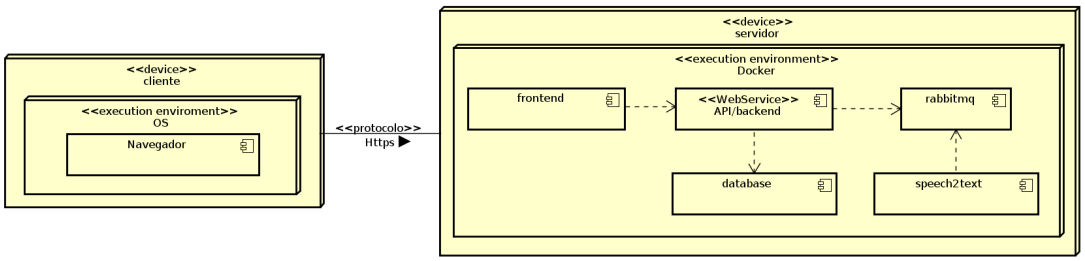


Figura C.4: Modelo de despliegue.

La decisión de tener dos dispositivos para la aplicación, viene determinada por la necesidad de tener un servidor dedicado a la transcripción de audio a texto. Este diagrama de despliegue concuerda con el patrón arquitectónico propuesto, ya que tendremos los microservicios desplegados en un servidor con separación responsabilidades.

## Modelos dinámicos

En esta sección se incluyen los diagramas de secuencia, que reflejan el comportamiento del sistema en cuanto al flujo de operaciones. Para ello nos centraremos en el caso de uso principal, que es la conversión de un archivo de audio a texto.

El diagrama de secuencia que se muestra en las figura C.5, ilustra el escenario donde un archivo de audio va siendo procesado por cada microservicio. En primer lugar, el

*frontend* guarda el fichero de audio y crea una *request* que se envía al *backend*, que guarda una petición en la base de datos y crea una *request* para que el servicio de transcripción *speech2text* genere la transcripción. Luego, el *backend* actualiza la petición previamente creada y responde al *frontend* con la transcripción, tiempo de procesamiento y confianza.

Todo este proceso en síncrono, por lo que el tiempo de respuesta dependerá del tiempo de procesamiento de cada microservicio. Además, el diagrama muestra un resumen del proceso general.

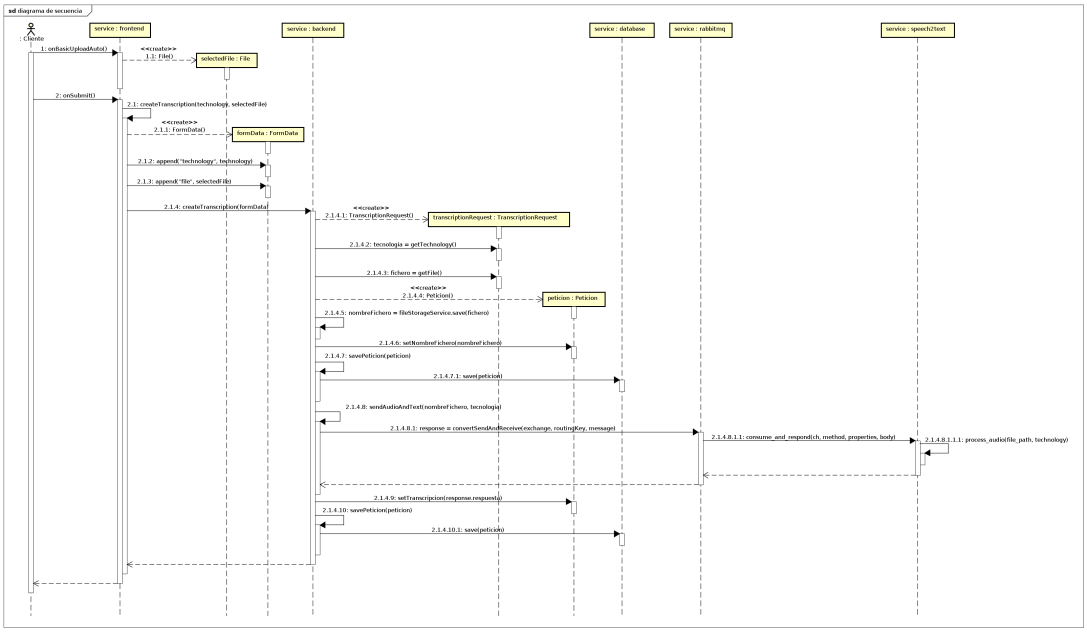


Figura C.5: Diagrama de secuencia para transcripción.

## C.4. Diseño interfaz Gráfica

En esta sección se mostrará el boceto que se ha utilizado para guiar el diseño de la interfaz de usuario.

Para la creación del boceto se ha utilizado la herramienta Balsamiq Mockups. Este boceto permitió obtener una visión general del diseño gráfico de la aplicación y fue elaborado en el *sprint* correspondiente en Angular.

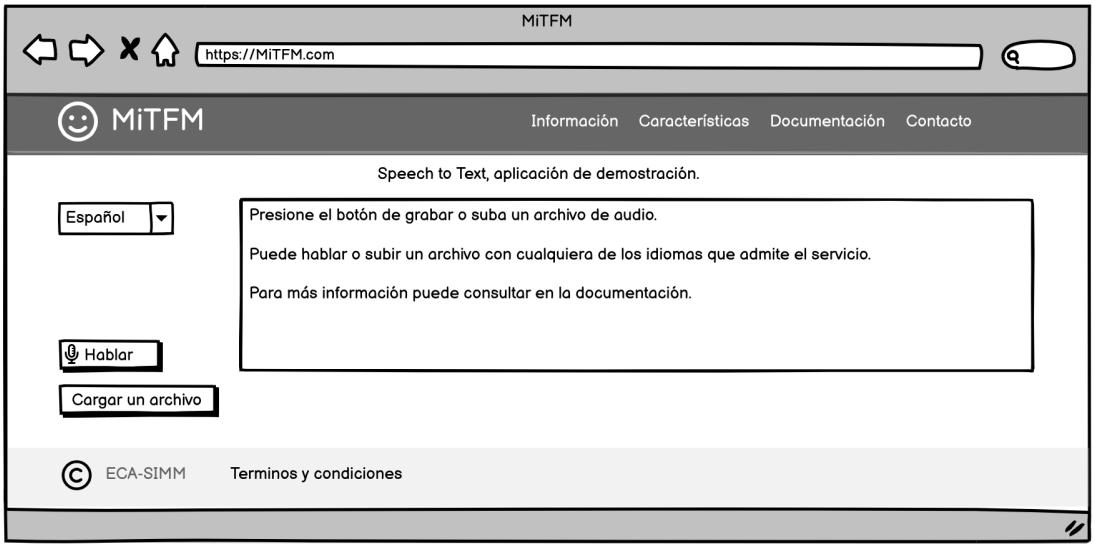


Figura C.6: Boceto de página principal.

El resultado final de la interfaz gráfica se puede ver en la figura E.2.

La interfaz gráfica se diseñó de modo que el usuario pueda interactuar de forma intuitiva y eficiente con la aplicación. Además, se ha tenido en cuenta la adaptación de la interfaz para diferentes dispositivos (escritorio, tabletas y móviles).

## C.5. Diseño arquitectónico

En este apartado se presentan los dos modelos arquitectónicos del sistema: uno físico, que muestra la distribución física del sistema y uno lógico, que muestra la comunicación entre microservicios.

Dado que se trata de una arquitectura de microservicios y está basada en contenedores Docker, se facilita la orquestación y administración de servicios mediante Docker Compose, lo que simplifica la gestión y el mantenimiento de la aplicación. En un entorno de mayor escala, se podría usar Kubernetes para aprovechar el escalado automático y ajustar dinámicamente los recursos según la carga de trabajo.

### Arquitectura física

El modelo arquitectónico físico de la aplicación web es de tipo cliente-servidor y se basa en un patrón arquitectónico de dos niveles, que corresponden con la forma en que las capas lógicas (lógica de presentación y lógica de negocio) se encuentran distribuidas de forma física, como se muestra en la figura C.7.

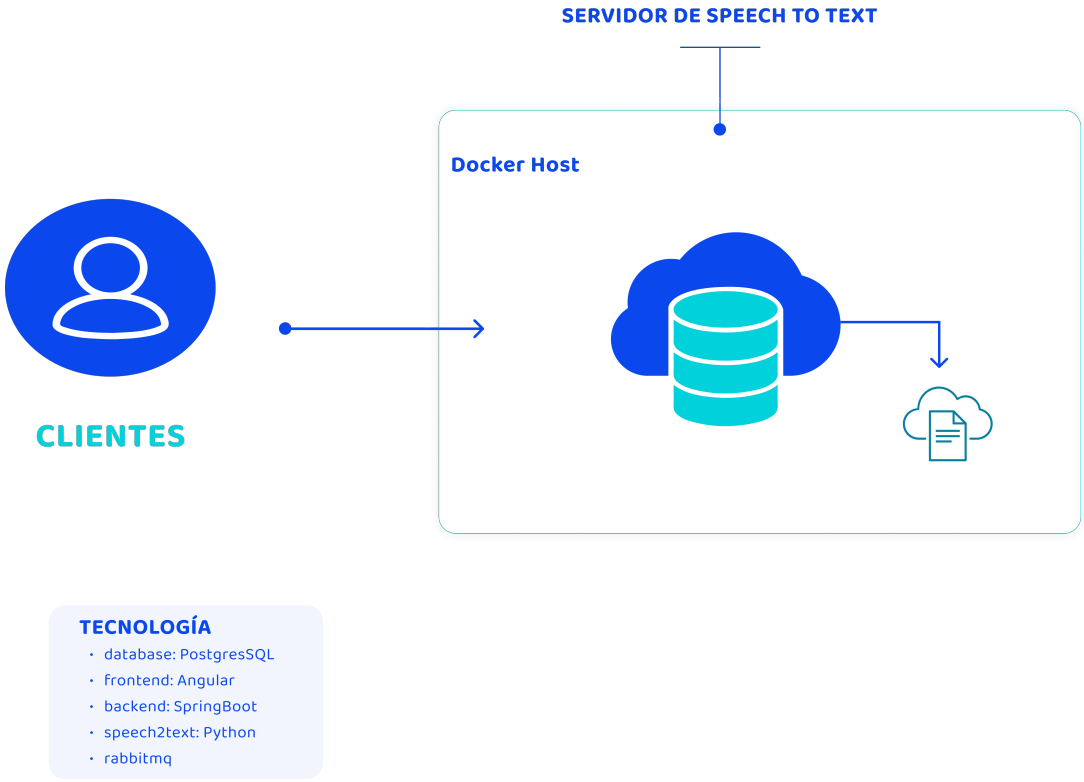


Figura C.7: Arquitectura física. Basado en diagrama Figma [6]

Es diseño arquitectónico representa un servidor que contiene los microservicios desplegados en contenedores de Docker, que pueden ser replicados según la demanda de los clientes, como se puede ver en el diagrama de despliegue C.4. Además, se indican los distintos *frameworks* utilizados para cada microservicio.

## Arquitectura lógica

La arquitectura propuesta a nivel lógico sigue una arquitectura de microservicios como se puede ver en la C.8.

En este diagrama se puede apreciar la comunicación entre microservicios y como se van guardando los ficheros de audio en el servidor, las peticiones en la base de datos y el uso de la cola de mensajes.

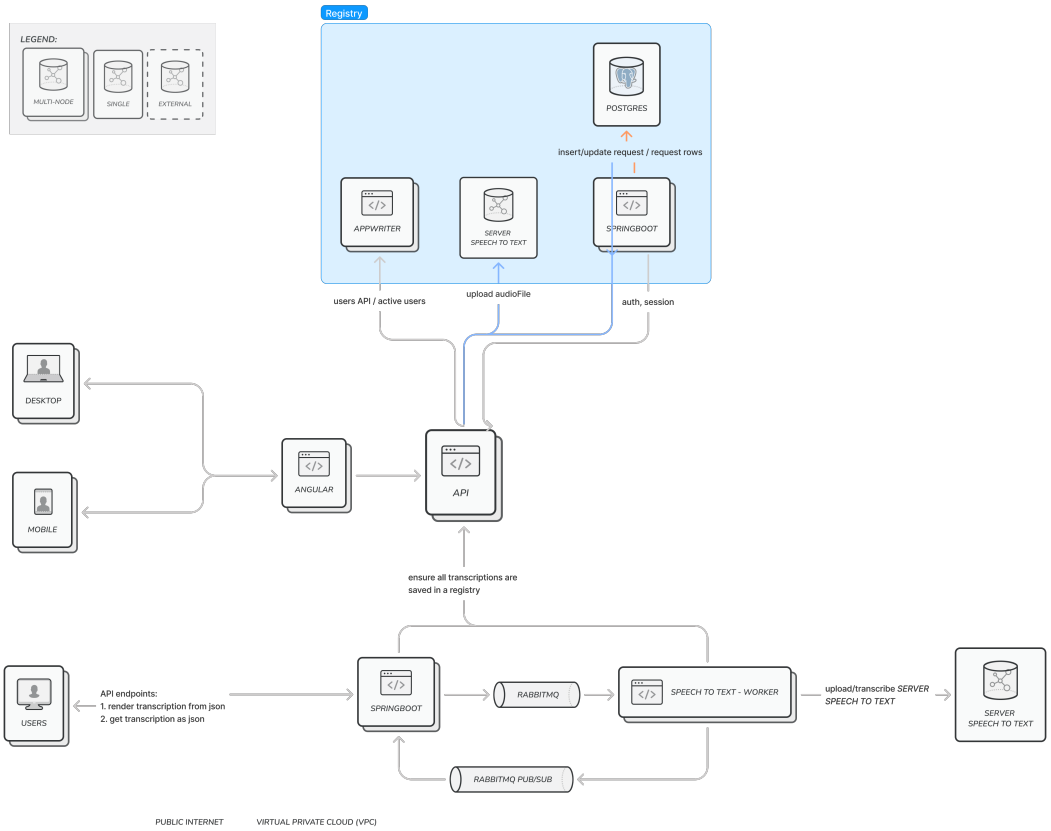


Figura C.8: Arquitectura lógica. Basado en diagrama Figma [7]

Apéndice $D$

# Documentación del Programador

En este apéndice se incluye la documentación necesaria para que el programador de aplicaciones pueda comprender la estructura de la solución software aportada y para poder modificarla. El código está disponible en el repositorio GitHub [1].

## D.1. Introducción

Este apéndice proporciona una guía para los programadores que deseen comprender, modificar y mantener el proyecto de transcripción de voz a texto. Se aborda la estructura de directorios del proyecto, la configuración del entorno, las instrucciones para la compilación, instalación y ejecución del proyecto, así como las pruebas del sistema.

## D.2. Estructura de directorios

El proyecto está dividido en tres componentes principales: *frontend* con Angular, *backend* con SpringBoot y *speech2text* con Python para los servicios de procesamiento de audio. A continuación se detalla la estructura de directorios para cada componente.

### Estructura de directorios de Angular

En la estructura de directorios se muestran las carpetas que contienen los módulos y componentes de la aplicación. Además de la carpeta de recursos y entornos.

[1] https://github.com/st2app/s2tapp


Figura D.1: Estructura del directorio de *front-end* en Angular.

- **AppModule (‘/app‘)**: Es el punto de entrada de la aplicación y se encarga de arrancar la aplicación, importar las dependencias globales y configurar el enrutamiento.
- **CoreModule (‘/core‘)**: Contiene las funcionalidades globales para el funcionamiento de la aplicación y sólo debe importarse una vez.
- **SharedModule (‘/shared‘)**: Reúne funcionalidades utilizadas por otros módulos.
- **LegalModule (‘/legal‘)**: Incluye componentes relacionados con la parte legal de la aplicación. *privacy-policy* y *terms-conditions* son componentes de la aplicación, donde se describen las políticas de privacidad y los términos y condiciones
- **Contact, Dashboard y Documentation** : Contiene componentes de la aplicación. *Dashboard* contiene la página principal.
- **Recursos (‘/assets‘)**: Contiene imágenes y otros archivos estáticos.
- **Entornos (‘/environments‘)**: Archivos de configuración para diferentes entornos (desarrollo, producción).

## Estructura de directorios de Spring Boot

La estructura de directorios sigue el arquetipo de Maven proporcionado por SpringBoot. contiene los archivos de configuración y las clases organizadas paquetes.

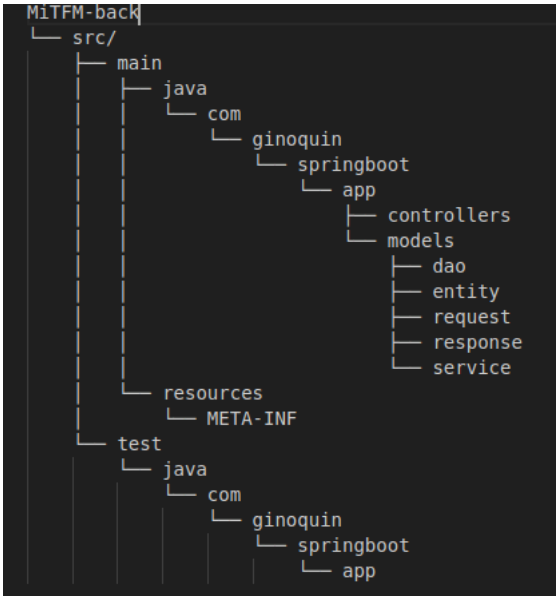


Figura D.2: Estructura del directorio de *back-end* en SpringBoot.

- **Fuentes (‘/src‘)**: Contiene el código fuente principal de la aplicación.
- **Controladores (‘/controllers‘)**: Gestiona las solicitudes HTTP y responde con datos en formato JSON.
- **Modelos (‘/models‘)**: Representan las entidades del dominio y mapea los datos desde y hacia la base de datos. Contiene subcarpetas con los DAO, las entidades, las clases para las solicitudes y respuestas y los servicios.
- **Recursos (‘/resources‘)**: Archivos de configuración y otros recursos estáticos.
- **Pruebas (‘/test‘)**: En esta carpeta se guardan las pruebas unitarias y de integración.

## Estructura de directorios de Python

El componente Python cuenta con un único fichero que contiene el el *worker* encargado de las transcripciones.

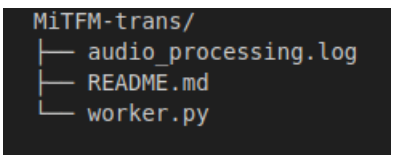


Figura D.3: Estructura del directorio de *back-end* en Python.

## Estructura de directorios de infraestructura de microservicios

La infraestructura de microservicios tiene dos directorios denominados 'st2app' y 'volumes'.

### Estructura del directorio 'st2app'

En la carpeta 'st2app' contiene una carpeta por cada servicio que requiere el sistema:

- **rabbitmq**: Contiene el Dockerfile necesario para construir una imagen Docker personalizada de RabbitMQ. Esta imagen incluye configuraciones específicas para nuestra aplicación, como la zona horaria, la configuración del usuario y grupo de rabbitMQ y los permisos.
- **database**: Contiene el Dockerfile para construir una imagen Docker personalizada de PostgreSQL. Esta imagen incluye configuraciones específicas como la zona horaria y la configuración del usuario y grupo de PostgreSQL.
- **backend**: Almacena el ejecutable JAR de la aplicación SpringBoot y el Dockerfile necesario para construir una imagen Docker personalizada. Esta imagen incluye configuraciones específicas para ejecutar la aplicación, como la definición de variables de entorno y la configuración del usuario y grupo para la ejecución del JAR.
- **frontend**: Esta carpeta contiene la estructura completa para desplegar nuestra aplicación Angular como un contenedor Docker. En esta carpeta se guarda la aplicación construida en Angular, una fichero para la configuración del servidor en Nginx, un fichero con los MIME Types y el DockerFile para crear la imagen. Esta imagen incluye la instalación de la aplicación en Nginx, la configuración del servidor y la copia de los archivos de la aplicación.
- **speech2text**: Esta carpeta contiene la estructura completa para desplegar nuestra aplicación de reconocimiento de voz como un contenedor Docker. Incluye el código para cargar modelos preentrenados, procesar el audio y generar la transcripción. Además, contiene un DockerFile para crear la imagen, donde se especifican todas las dependencias necesarias y la configuración del entorno para la ejecución.

También incluye un archivo .env que contiene la instrucción COMPOSE_FILE para indicar a Docker Compose que use el archivo 's2tapp-services.yml' como plantilla para configurar y ejecutar los servicios de la aplicación. Este archivo YAML detalla cómo se construyen, conectan y configuran los componentes de la aplicación.

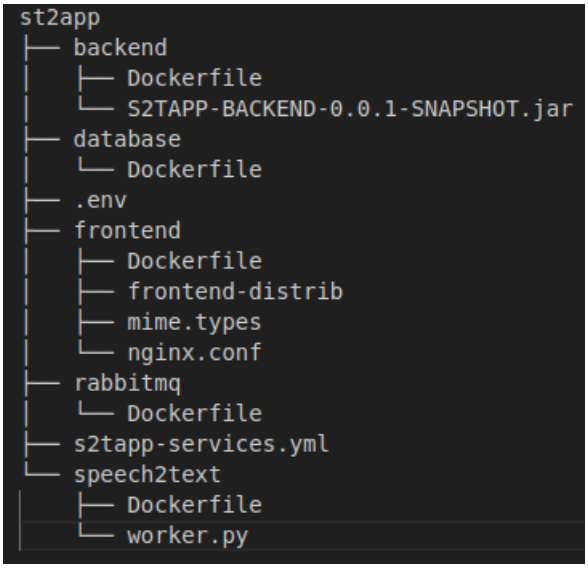


Figura D.4: Estructura del directorio 'st2app'.

### Estructura del directorio 'volumes'

En la carpeta 'volumes' encontraremos una carpeta por cada volumen necesario por los servicios del sistema:

- **postgres**: Esta carpeta es el esencial para el funcionamiento de PostgreSQL. La subcarpeta 'init' alberga el script init-user-db.sh, crucial para configurar el usuario y la base de datos iniciales. En la carpeta 'data' se almacenan de manera persistente todos los datos de la base, lo que permite recuperar la información en caso de fallo.
- **rabbitmq**: Almacena de manera persistente los datos críticos de RabbitMQ, como la configuración, los metadatos y los *logs*. Esto permite recuperar el sistema en caso de fallo y facilita la gestión y el mantenimiento.
- **huggingface**: : Almacena los modelos de inteligencia artificial utilizados por el servicio de conversión de voz a texto. Específicamente, se encuentran los modelos Whisper Large v3 de OpenAI y Wav2Vec2 Large 960h LV60 Self de Facebook. Estos modelos, previamente entrenados en grandes conjuntos de datos, son descargados desde la plataforma Hugging Face y utilizados por la librería *Transformers* para realizar la tarea de transcripción de audio a texto.

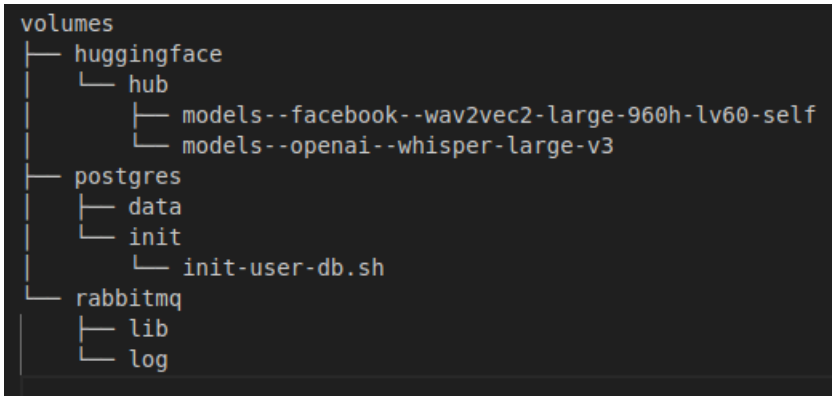


Figura D.5: Estructura del directorio 'volumes'.

## D.3. Configuración del Entorno

En esta sección se explica cómo montar el entorno de desarrollo y obtener el código fuente del proyecto.

### Infraestructura microservicios y otros

Requisitos para la gestión de la infraestructura de microservicios, control de versiones, sistema operativo y navegador web:

- *Git* versión 2.34.1
- Docker versión 27.1.0, build 6312585
- Docker Compose version v2.29.0
- Sistema operativo *Linux* por ejemplo Ubuntu, Xubuntu, entre otros.
- Navegador web como Google Chrome, Edge o Firefox en su versión más reciente.

### Front-end Angular

Requisitos para la parte de *front-end* en Angular:

- *Node.js* versión 22.2.0.
- *Angular CLI* versión 18.0.1
- Visual Studio Code versión 1.91.1.

Se deben instalar las dependencias necesarias para el proyecto, para ello se utilizará el comando 'npm' de *Node.js*:

```
~$  npm install
```

**Nota:** La lista completa de dependencias está en el archivo 'package.json'.

Estas son algunas dependencias que se utilizan en el proyecto:

- *Primeflex* 3.3.1: Un conjunto de clases de utilidad CSS para el diseño y alineación del contenido.
- *PrimeNG* 17.16.1: La biblioteca de componentes de Angular utilizada para construir la interfaz de usuario.
- *Primeicons* 7.0.0: Iconos de *PrimeNG* utilizados en la interfaz de usuario.
- *Quill* 1.3.7: Un editor de texto rico utilizado en la aplicación.

Para obtener el código fuente del proyecto en Angular hay que ejecutar los siguientes comandos:

```
~$ mkdir repositorio
~$ cd repositorio
~$ git clone https://github.com/s2tapp/MiTFM-front.git
```

## Back-end SpringBoot

Requisitos para la parte de *back-end* en SpringBoot:

- OpenJdk versión 17
- Eclipse con Spring Tools 4
- Apache Maven versión 3.6.3

Para obtener el código fuente del proyecto de SpringBoot, se deben ejecutar los siguientes comandos:

```
~$ mkdir repositorio
~$ cd repositorio
~$ git clone https://github.com/s2tapp/MiTFM-back.git
```

## Back-end Python

Requisitos para la parte de *back-end* en Python:

- Python versión 3.x
- Visual Studio Code versión 1.91.1.

Se deben instalar los paquetes necesarios para el proyecto, para ello se utilizará el comando 'pip':

```
~$ pip install
```

Estas son los paquetes que se utilizan en el proyecto:

- Torch versión 2.3.0: PyTorch es una librería de *machine learning* que proporciona herramientas para realizar cálculos numéricos y construir redes neuronales.
- Pika versión 1.3.2: Pika es una librería para interactuar con RabbitMQ, un sistema de mensajería basado en el protocolo AMQP.
- Requests versión 2.31.0: Requests es una librería para realizar solicitudes HTTP de manera simple y eficiente en Python. utilizada para construir la interfaz de usuario.
- Ffmpeg versión 1.4: FFmpeg es una herramienta para manejar archivos multimedia, que permite convertir, grabar y transmitir audio y vídeo.
- Librosa versión 0.10.2: Librosa es una librería de Python para análisis de música y audio, que proporciona herramientas para tareas como el análisis de señales y extracción de características.
- Transformers versión 4.41.0.dev0: Transformers es una librería desarrollada por Hugging Face que proporciona modelos preentrenados para la conversión de voz a texto, como Whisper, WAv2Vec, entre otros.
- Accelerate versión 0.29.3: Accelerate es una librería que facilita la distribución de modelos y tareas de *machine learning* en múltiples dispositivos, como GPUs y TPUs.

Para obtener el código fuente del proyecto en Python, se deben ejecutar los siguientes comandos:

```
~$ mkdir repositorio
~$ cd repositorio
~$ git clone https://github.com/s2tapp/MiTFM-trans.git
```

## D.4. Compilación, instalación y ejecución del proyecto

Es recomendable usar un sistema operativo Linux para garantizar la compatibilidad con las herramientas de desarrollo.

### Front-end Angular

Para el proyecto en Angular será necesario instalar los requisitos indicados previamente.

Tras clonar el repositorio pasaremos a abrir el proyecto con Visual Studio Code, para ello en iremos a 'Archivo' →'Abrir carpeta' →'repositorio' →'MiTFM-front'.

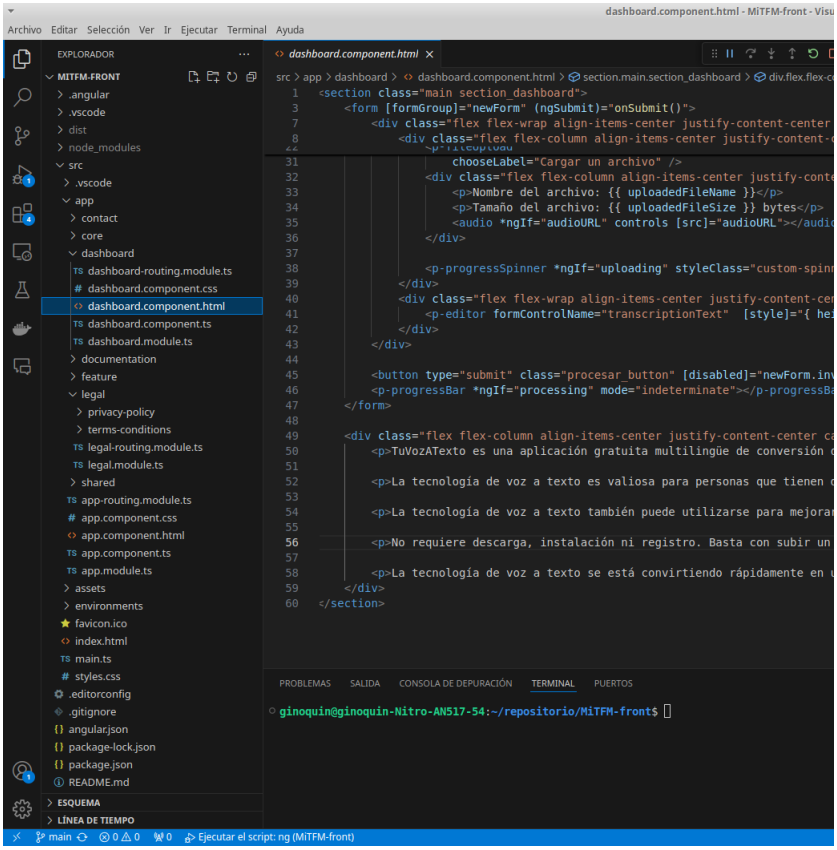


Figura D.6: Importación de proyecto Angular

Una vez importado para ejecutar el proyecto debemos situarnos en el directorio y ejecutar el siguiente comando en el terminal que proporciona Visual Studio Code o en un terminal del sistema operativo:

```
~$ ng serve
```

Se puede iniciar en modo debugger con Visual Studio Code, para ello iremos a 'Ejecución y depuración' →'EJECUCIÓN...' →'Node js...' →'Ejecutar el script: start'.

Esto arrancará la aplicación en el siguiente enlace http://localhost:4200/.

Para compilar el proyecto debemos situarnos en el directorio y ejecutar el siguiente comando en el terminal que proporciona Visual Studio Code o en un terminal del sistema operativo:

```
~$ ng build
```

Este comando genera una carpeta 'dist' que contendrá los archivos para desplegar la aplicación. Esta carpeta se debe copiar en la carpeta *frontend* de la estructura de microservicios para su despliegue.

## Back-end SpringBoot

Para el proyecto en *SrpingBoot* será necesario instalar los requisitos indicados previamente.

Tras clonar el proyecto, pasaremos a importarlo en *Eclipse*. Para ello elegiremos la opción de importar proyectos existentes de Maven. Será necesario indicar el repositorio local en el que tenemos la aplicación, lo seleccionamos y damos finalizar.

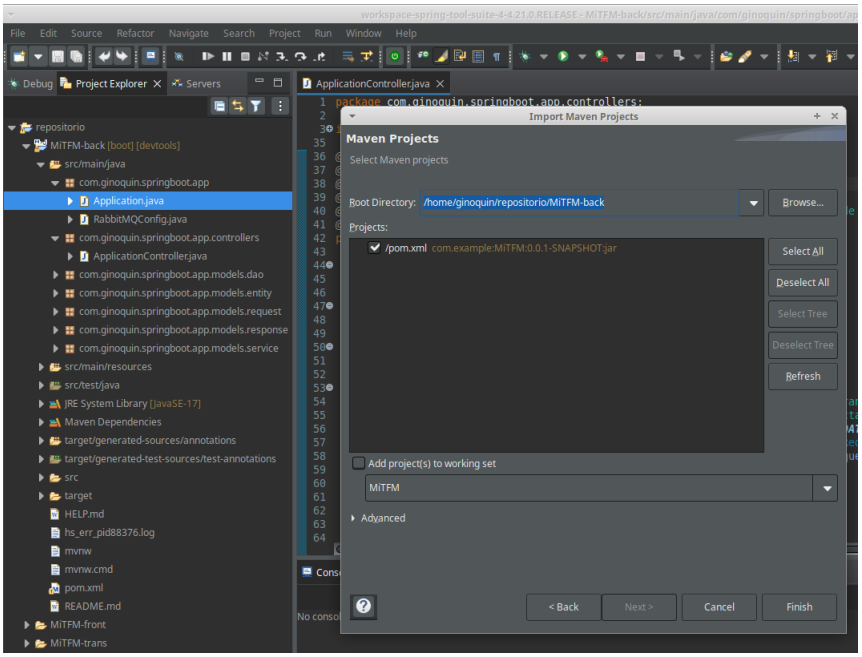


Figura D.7: Importación de proyecto SpringBoot

Una vez importado será necesario configurar dos variables de entorno antes de ejecutar el proyecto, para ello iremos a 'MiTFM-back' →'Debug As' →'Debug Configurations...' →'Spring Boot App' →'New Configuration'

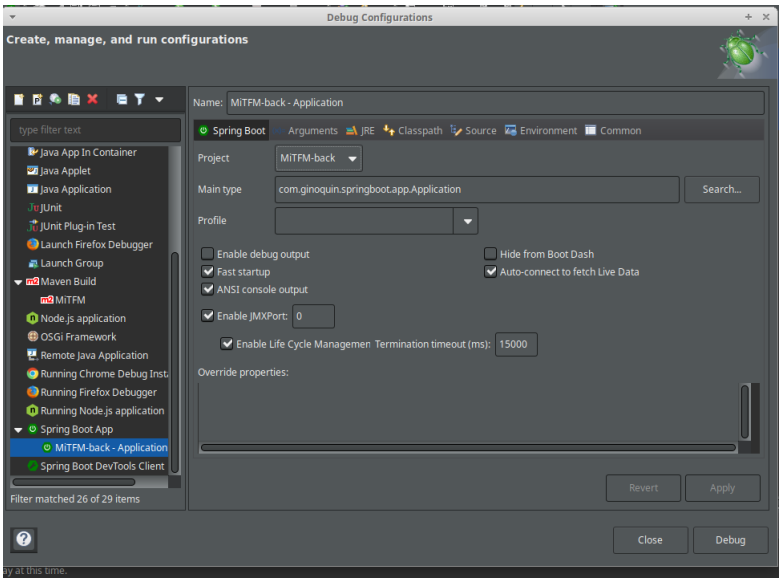


Figura D.8: Configuración proyecto SpringBoot

Después hay que introducir las siguientes variables de entorno:

- DB_PASSWORD: SConver2023! (Contraseña para la base de datos).
- DB_URL: jdbc:postgresql://localhost:5432/db_mitfm?serverTimezone=Europe/Madrid (URL de conexión a la base de datos PostgreSQL).
- DB_USER: sconver (Usuario de la base de datos).
- RABBITMQ_HOST: localhost (*Host* de RabbitMQ).

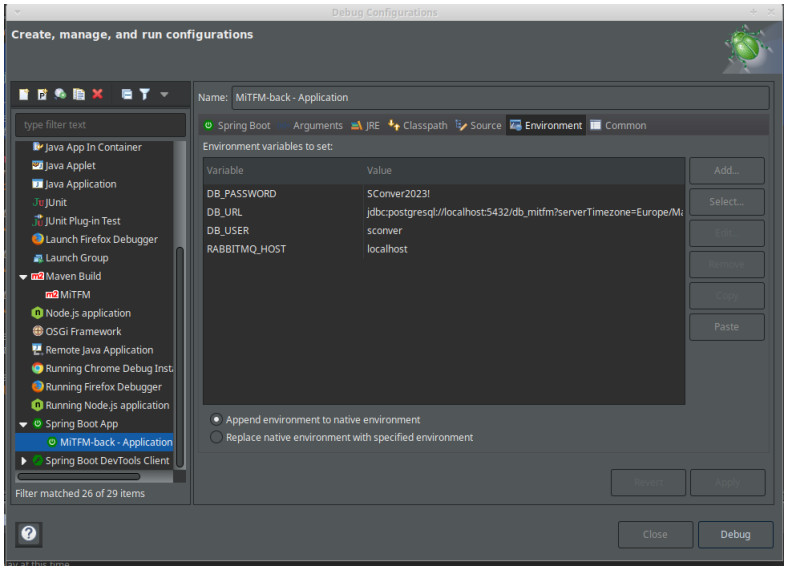


Figura D.9: Variables de entorno SpringBoot

Para compilar el proyecto será necesario configurar la compilación a través de Maven por ello antes de compilar el proyecto, iremos a 'MiTFM-back' →'Debug As' →'Debug Configurations...' →'Maven Build' →'New Configuration'

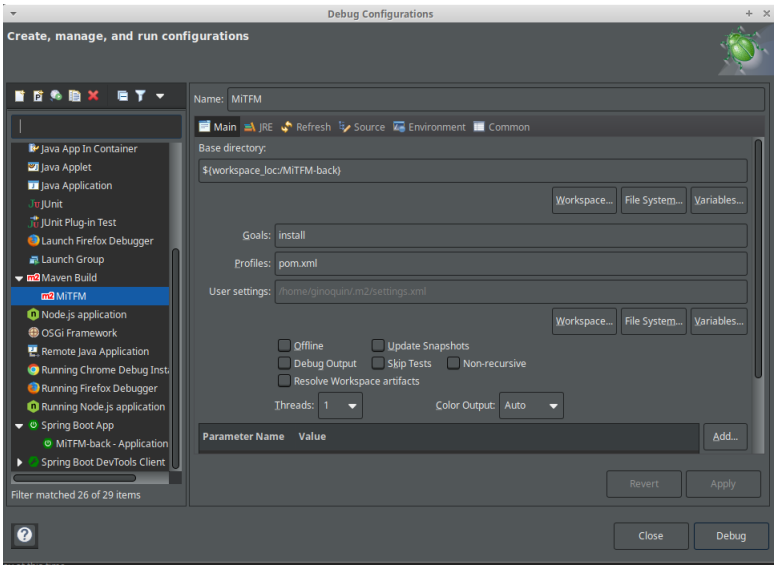


Figura D.10: Configuración Maven en SpringBoot

Tras ejecutar el comando con Eclipse genera una carpeta 'target' que contendrá el ejecutable 'MiTFM-0.0.1-SNAPSHOT.jar' y que posteriormente copiaremos a la carpeta *backend* de la estructura de microservicios para su despliegue.

## Back-end Python

Para el proyecto en Python será necesario instalar los requisitos indicados previamente.

Tras clonar el repositorio pasaremos a abrir el proyecto con Visual Studio Code, para ello iremos a 'Archivo' →'Abrir carpeta' →'repositorio' →'MiTFM-trans'.

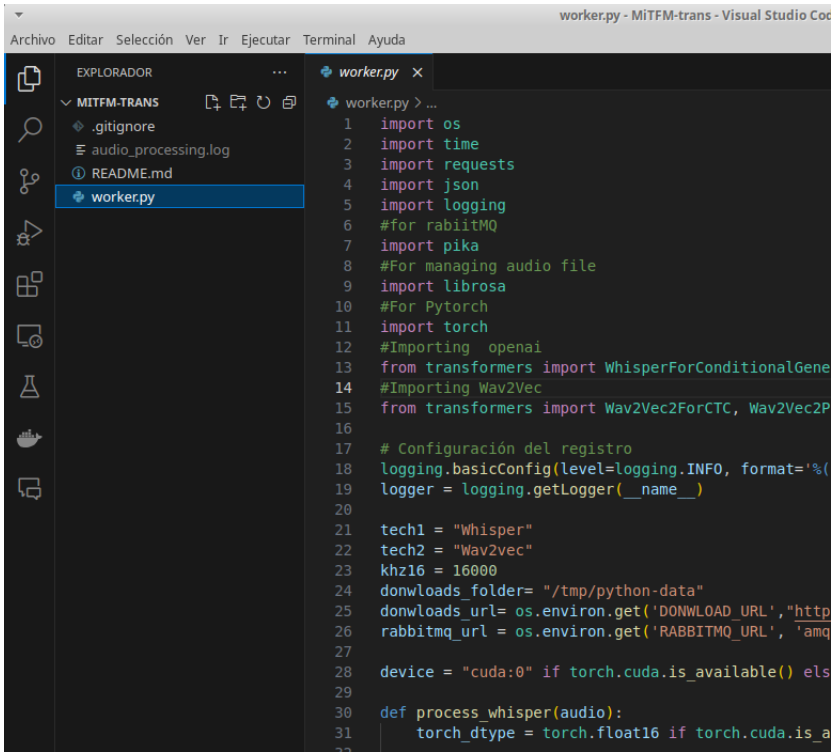


Figura D.11: Importación de Python

Se puede iniciar en modo *debugger* en Visual Studio Code para ello iremos a 'Ejecución y depuración' →'Python Debugger...'. Esto arrancará la aplicación que recibe mensajes para realizar la transcripción.

Después, es necesario definir las variables de entorno que permiten la comunicación entre servicios a través de Docker Compose, en este caso se han definido las siguientes:

- DONWLOAD_URL: URL base para descargar archivos de audio. En este caso se ha definido la siguiente URL: `http://mitfmapp-back:8080/api/files/`
- RABBITMQ_URL: URL de RabbitMQ para la comunicación con la cola de mensajes.En este caso se ha definido la siguiente URL:`amqp://rabbitmq?connection_attempts=10&retry_delay=10`

El archivo 'worker.py'se debe copiar en la carpeta *speech2text* de la estructura de microservicios para su despliegue.

## Infraestructura microservicios y otros

Para la ejecución del proyecto utilizando el ecosistema de Docker, será necesario instalar los requisitos indicados previamente.

Lo primero que se debe hacer es replicar las carpetas necesarias siguiendo la estructura indicada anteriormente.

En primer lugar, se debe crear la estructura de directorios que usan los servicios. Esta estructura de directorios, así como los archivos necesarios, se encuentran en el repositorio de GitHub [2].

Dentro de esta estructura de carpetas, se encuentra un archivo *Compose* llamado 's2tapp-services.yml' que permite la administración de servicios, redes y volúmenes en un único archivo de configuración YAML.

Lo siguiente que se debe hacer es replicar las carpetas dedicadas a los volúmenes. Esta estructura de directorios, así como los archivos necesarios, se encuentran en el repositorio de GitHub [3]. La estructura no está completa, ya que algunas carpetas se recrean cuando se ejecutan los servicios con Docker.

El archivo *Compose* simplifica el control de las aplicaciones y permite, con un solo comando, crear e iniciar todos los servicios.

Para iniciar todos los servicios del sistema, se puede ejecutar el siguiente comando:

```
~$ docker compose up -d
```

Para parar todos los servicios se puede ejecutar el siguiente comando:

```
~$ docker compose stop
```

Para detener y eliminar los contenedores, redes y volúmenes definidos se usa el siguiente comando que permite limpiar todos los recursos creados por Docker Compose y restaurar el entorno a su estado original.

```
~$ docker compose down
```

[2] https://github.com/st2app/st2app
[3] https://github.com/st2app/volumes

Otra opción interesante es construir las imágenes de los servicios del sistema para después volver a iniciarlos. Ya que, al realizar cualquier modificación en el código de las aplicaciones o cualquier otra modificación en la configuración de los contenedores es necesario reconstruir las imágenes para que detecte los cambios. Para ello se puede ejecutar el siguiente comando:

```
~$ docker compose build
```

Para el desarrollo del sistema, se recomienda iniciar solo los servicios de base de datos y cola de mensajes. Para ello, se deben ejecutar los siguientes comandos:

```
~$ docker compose up -d database adminer rabbitmq
```

Esto permite ejecutar cada aplicación en el entorno de desarrollo e ir probando las nuevas funcionalidades o correcciones de errores que se lleven a cabo.

El archivo *Compose* contiene la configuración para todos los servicios del sistema. Pero, hay dos servicios que ofrecen una interfaz web para la administración de la base de datos y la cola de mensajes.

### Servicio de administración de base de datos

Se usa Adminer para proporcionar una interfaz web intuitiva para administrar la base de datos de forma gráfica. Este servicio facilita la administración del contenido de la base de datos. Este servicio es opcional y se puede acceder a la interfaz web a través de la URL: `http://localhost:9090`.

Para acceder a la base de datos del sistema hay que indicar los siguientes datos:

- Motor de base de datos: PostgreSQL.
- Servidor: database
- Usuario: sconver
- Contraseña: SConver2023!
- Base de datos: db_mitfm

### Servicio de mensajería

Se usa RabbitMQ como *broker* de mensajes para facilitar la comunicación entre los diferentes microservicios. Se ha configurado un servicio dedicado para gestionar la cola de mensajes y garantizar una entrega confiable.

Este servicio debe estar siempre iniciado a la hora del desarrollo de los servicios principales y se puede acceder a la interfaz web a través de la URL: `http://localhost:15672/`.

Para acceder a la gestión de colas de mensajes hay que indicar los siguientes datos:

- Username: guest
- Password: guest

## D.5. Pruebas del sistema

Dado que se trata de un sistema de transcripción donde se depende de un modelo encargado de este proceso, las pruebas se han realizado de forma manual. Siempre garantizando que se cumple la funcionalidad implementada.

*Apéndice* $E$

# Documentación de usuario

En este apéndice se incluye la documentación necesaria para que el usuario entienda cómo utilizar el sistema desarrollado.

## E.1. Introducción

Este documento está diseñado para guiar a los usuarios a utilizar el sistema de transcripción de voz a texto. A lo largo de este apéndice, los pasos necesarios para acceder al sistema a través de un navegador web, así como una guía detallada sobre cómo utilizar las funcionalidades del sistema.

## E.2. Requisitos de usuarios

Para poder utilizar el sistema de manera efectiva, los usuarios deben cumplir con los siguientes requisitos:

- Conocimientos básicos de informática: Los usuarios deben tener una comprensión básica del uso de computadoras y navegación web.
- Sistema operativo compatible: El sistema ha sido probado y es compatible con Windows, macOS y Linux.
- Navegador web: Un navegador web moderno como Google Chrome, Mozilla Firefox, Microsoft Edge, o Safari.
- Conexión a Internet: Se requiere una conexión estable a Internet usar el sistema y utilizar todas sus funcionalidades.

## E.3. Instalación

El sistema de transcripción de voz a texto no requiere instalación. Todo lo que necesita es un navegador web compatible y acceso a Internet.

Para acceder al sistema:

1. Abra su navegador web preferido.
2. Ingrese la dirección web `http://localhost/dashboard`.
3. Esto lo llevará a la página principal del sistema donde se podrá subir archivos para obtener la transcripción.

## E.4. Manual del usuario

### Tecnologías de transcripción disponibles

El sistema de transcripción de voz a texto permite al usuario elegir entre diferentes tecnologías de reconocimiento de voz, cada una con características particulares en cuanto a los idiomas y formatos de audio compatibles.

- Whisper es un potente modelo de reconocimiento de voz desarrollado por OpenAI. Es compatible con una amplia gama de idiomas y formatos de audio.
  - Idiomas soportados: Inglés, Español, Francés, Alemán, Chino, Japonés, Coreano, Ruso, Portugués, Italiano.
  - Formatos de audio soportados: `wav`, `mp3`, `flac`, `ogg`.
- Wav2Vec es una tecnología de reconocimiento de voz desarrollada por Meta AI. Actualmente, está optimizada para el idioma inglés.
  - Idiomas soportados: Inglés.
  - Formatos de audio soportados: `wav`.

### Interfaz principal y funcionalidad básica

La pagína principal de la aplicación es un formulario donde se puede elegir la tecnología de transcripción, un cuadro de texto donde se mostrará la transcripción y un botón para iniciar el proceso de transcripción, como se muestra en la figura E.1.

Además, se muestra opciones para ver las características de la aplicación, documentación para su uso y contacto por si surge algún problema.

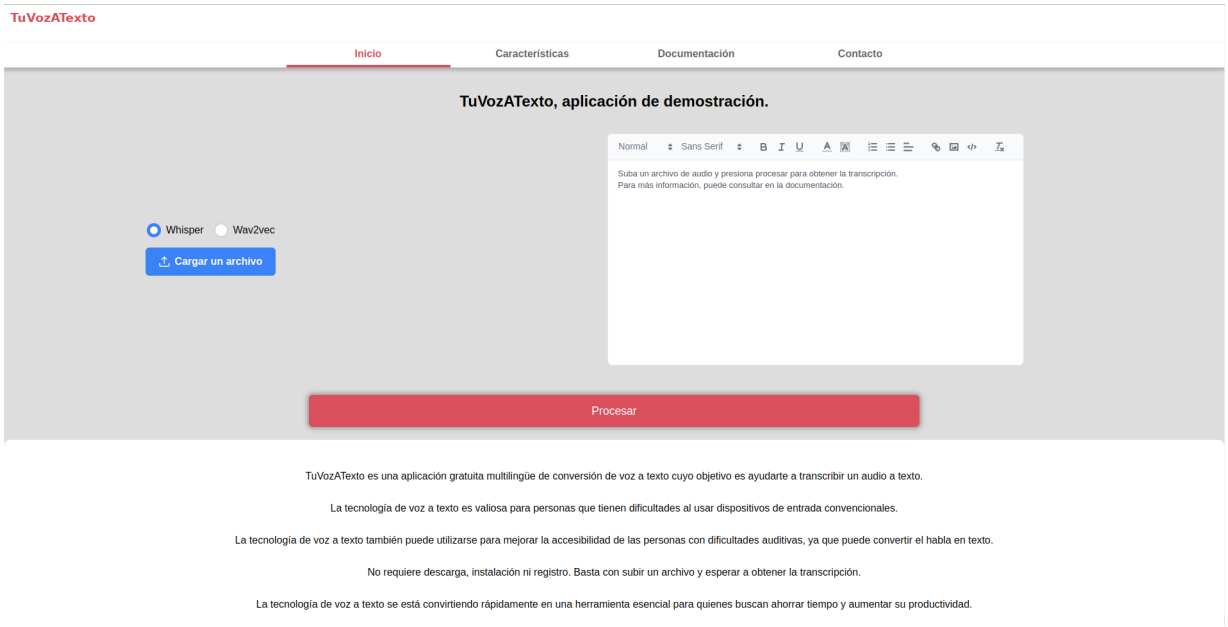


Figura E.1: Página principal.

## Subir un archivo de audio para la transcripción

1. Después de ingresar al sistema, seleccione la opción 'Cargar un Archivo' en la página principal.
2. Elija el archivo de audio que desea transcribir. El peso máximo que admite el sistema es de 2 MB.
3. Haga clic en 'Procesar' para iniciar el proceso.
4. Una vez completada la transcripción, el resultado se mostrará en el cuadro de texto mostrado en la pantalla.

Siguiendo los pasos descritos, se mostrará un reproductor de audio que permitirá escuchar el archivo cargado, junto con su nombre y el tamaño en bytes. Una vez procesado el archivo, la transcripción obtenida se visualizará en el cuadro de texto.

El resultado de este proceso se ilustra en la figura E.2.

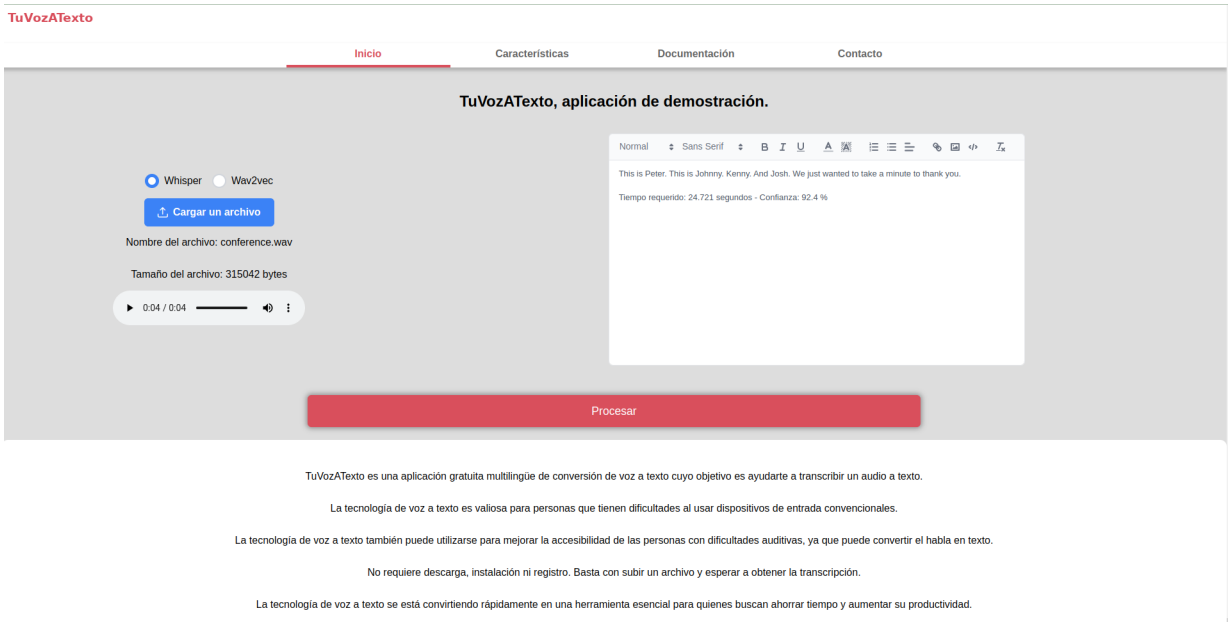


Figura E.2: Resultado procesamiento.

## Uso de la API para transcripción

Además de la interfaz de usuario, la aplicación ofrece una API para realizar transcripciones sin utilizar la interfaz web. A continuación se describen los pasos básicos para usar la API:

1. Abra su navegador web preferido.
2. Ingrese la dirección web: `http://localhost:8080/swagger-ui/index.html#/`.
3. Envíe una solicitud *POST* al *endpoint* de la API destinado a la transcripción de archivos de audio. La solicitud debe incluir el archivo de audio y la tecnología deseada para la transcripción. El peso máximo de archivo que admite el sistema es de 2 MB.
4. Ejecute la solicitud para obtener el resultado de la transcripción.
5. El resultado se devolverá en formato JSON.

Siguiendo los pasos anteriores, se mostrará la respuesta con la transcripción obtenida en formato JSON como se muestra en la figura E.3.

Si se desea descargar el documento de especificación de OpenAPI 3 para la API REST se debe ingresar a la dirección: `http://localhost:8080/v3/api-docs.yaml`. Este documento describe los *endpoints*, los parámetros, las respuestas y otros metadatos de la API. Los desarrolladores lo suelen utilizar para interactuar con la API de forma programática.

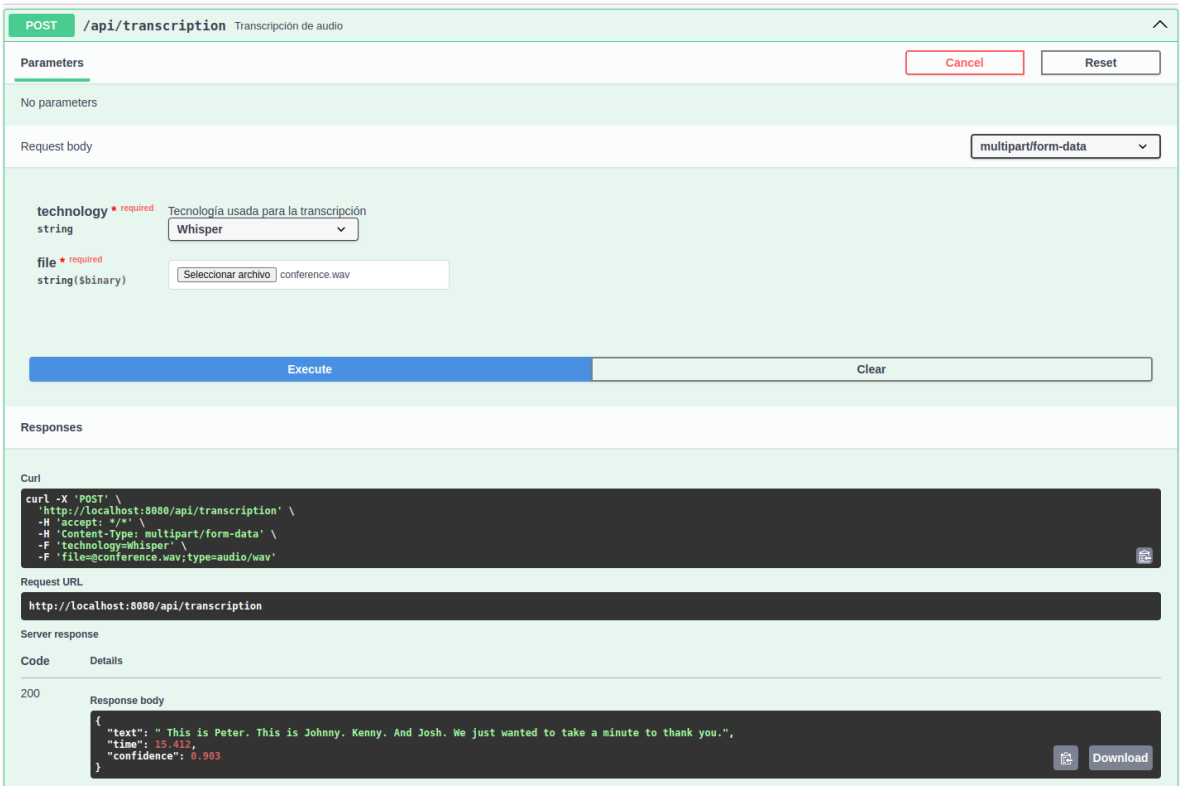


Figura E.3: Resultado procesamiento.

# Bibliografía


[1] S. Insider, "Speech-to-text api market." https://www.snsinsider.com/reports/speech-to-text-api-market-1576. [Online; accedido 09-septiembre-2024].

[2] K. S and E. Chandra, "A review on automatic speech recognition architecture and approaches," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, pp. 393–404, 04 2016.

[3] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *ArXiv.org*, 2023. https://arxiv.org/abs/2106.11342.

[4] proyectosagiles.org, "Qué es scrum." https://proyectosagiles.org/que-es-scrum. [Online; accedido 26-abril-2024].

[5] Y. G. Ibáñez, "Patrones de arquitectura de microservicios, ¿qué son y qué ventajas nos ofrecen?." https://www.paradigmadigital.com/dev/patrones-arquitectura-microservicios-que-son-ventajas/. [Online; accedido 17-julio-2024].

[6] darren@figma, "Architecture diagram example- multiplayer." https://www.figma.com/community/file/989634471195357925. [Online; accedido 11-junio-2024].

[7] Jimena, "Diagrama de arquitectura." https://www.figma.com/community/file/1230226464794538709. [Online; accedido 11-junio-2024].

[8] IBM, "¿qué es el reconocimiento del habla?." https://www.ibm.com//es-es/topics/speech-recognition. [Online; accedido 11-junio-2024].

[9] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision." https://cdn.openai.com/papers/whisper.pdf, 2023. [Online; accedido 23-junio-2024].

[10] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020. https://arxiv.org/abs/2006.11477.

[11] D. Inc., "Docker overview." https://docs.docker.com/get-started/overview/. [Online; accedido 20-abril-2024].

[12] D. Inc., "Docker compose overview.." https://docs.docker.com/compose/. [Online; accedido 17-julio-2024].

[13] P. S. Foundation, "The python tutorial." https://docs.python.org/3/tutorial/index.html. [Online; accedido 11-junio-2024].

[14] Broadcom, "Springboot." https://spring.io/projects/spring-boot. [Online; accedido 20-abril-2024].

[15] Google, "Angular." https://angular.io/. [Online; accedido 20-abril-2024].

[16] S. Software, "What is swagger?." https://swagger.io/docs/specification/2-0/what-is-swagger/. [Online; accedido 11-junio-2024].

[17] Transcripciones.net, "La historia de la transcripción." https://transcripciones.net/blog/la-historia-de-la-transcripcion/. [Online; accedido 30-junio-2024].

[18] IBM, "Speech recognition." https://www.ibm.com/history/voice-recognition. [Online; accedido 30-junio-2024].

[19] Wikipedia, "Speech recognition." https://en.wikipedia.org/wiki/Speech_recognition. [Online; accedido 30-junio-2024].

[20] L. Rabiner and B.-H. Juang, *Fundamentals of speech recognition*. USA: Prentice-Hall, Inc., 1993.

[21] Álvaro Alonso, "Reconocimiento del habla con hugging face." https://blog.kairosds.com/reconocimiento-del-habla-con-hugging-face/, 2022. [Online; accedido 23-junio-2024].

[22] H. Face, "Preprocesamiento de un conjunto de datos de audio." https://huggingface.co/learn/audio-course/es/chapter1/preprocessing, 2023. [Online; accedido 11-julio-2024].

[23] neobundy, "Menny the sonic whisperer - a deep dive into audio processing and the whisper model part i." https://github.com/neobundy/Deep-Dive-Into-AI-With-MLX-PyTorch/blob/master/deep-dives/003-whisper/README.md, 2024. [Online; accedido 11-julio-2024].

[24] R. E. Turner, "An introduction to transformers," 2024. https://arxiv.org/abs/2304.10557.

[25] R. Merritt, ""¿qué es un modelo transformer?"." https://la.blogs.nvidia.com/blog/que-es-un-modelo-transformer/, 2022. [Online; accedido 12-julio-2024].

[26] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.

[27] Wikipedia, "Attention (machine learning)." https://en.wikipedia.org/wiki/Attention_(machine_learning). [Online; accedido 12-julio-2024].

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv.org*, 2016.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[30] F. Turley and N. Rad, *Los Fundamentos de Agile Scrum*. Van Haren Publishing, 2019. https://books.google.es/books?id=yX-3DwAAQBAJ.

[31] S. Blasco Arnaiz, "Incorporación de un reconocedor automático de voz local sobre plataformas android," *UVaDOC*, 2020. https://uvadoc.uva.es/handle/10324/44146.

[32] D. Figueroa Fernández *et al.*, "Desarrollo de frontend web para grabación de ficheros de audio en tiempo real," *UVaDOC*, 2023. https://uvadoc.uva.es/handle/10324/62930.

[33] Google, "Speech-to-text." https://cloud.google.com/speech-to-text. [Online; accedido 16-septiembre-2024].

[34] M. Corporation, "Azure ai speech documentation." https://azure.microsoft.com/en-us/products/ai-services/ai-speech. [Online; accedido 16-septiembre-2024].

[35] IBM, "Ibm watson speech to text." https://www.ibm.com/products/speech-to-text. [Online; accedido 16-septiembre-2024].

[36] A. Inc., "Amazon transcribe." https://aws.amazon.com/es/transcribe/. [Online; accedido 16-septiembre-2024].

[37] D. Inspiration., "Dictation." https://dictation.io/speech. [Online; accedido 16-septiembre-2024].

[38] M. Corporation, "Estilo de arquitectura de microservicios." https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices, 2023. [Online; accedido 17-julio-2024].

[39] F. Vera-Rivera, H. Astudillo, and M. Gaona, "Desarrollo de aplicaciones basadas en microservicios: tendencias y desafíos de investigación," *RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao*, vol. E23, pp. 107 – 120, 07 2019.

[40] I. Pivotal Software, "Pivotal tracker." https://www.pivotaltracker.com/. [Online; accedido 20-abril-2024].

[41] S. F. Conservancy, "Git." https://www.git-scm.com/. [Online; accedido 20-abril-2024].

[42] I. GitHub, "Github." https://github.com/. [Online; accedido 20-abril-2024].

[43] I. ChangeVision, "Astah." https://astah.net/. [Online; accedido 20-abril-2024].

[44] I. Eclipse Foundation, "Eclipse." https://www.eclipse.org/. [Online; accedido 20-abril-2024].

[45] M. Corporation, "Visual studio code." https://code.visualstudio.com/. [Online; accedido 15-septiembre-2024].

[46] Overleaf, "Overleaf." https://overleaf.com. [Online; accedido 20-abril-2024].

[47] L. Balsamiq Studios, "Quick and easy wireframing tool." https://balsamiq.com/wireframes/. [Online; accedido 20-abril-2024].

[48] I. Figma, "Figma." https://www.figma.com/. [Online; accedido 20-abril-2024].

[49] T. A. S. Foundation, "Maven." https://maven.apache.org/. [Online; accedido 20-abril-2024].

[50] HuggingFace, "Transformers." https://huggingface.co/docs/transformers/es/index. [Online; accedido 11-junio-2024].

[51] OpenAI, "openai/whisper-large-v3." https://huggingface.co/docs/transformers/es/index. [Online; accedido 11-junio-2024].

[52] Facebook, "facebook/wav2vec2-large-960h-lv60-self." https://huggingface.co/facebook/wav2vec2-large-960h-lv60-self. [Online; accedido 11-junio-2024].

[53] Hibernate, "Hibernate." https://hibernate.org/. [Online; accedido 20-abril-2024].

[54] T. P. G. D. Group, "Postgresql: The world's most advanced open source relational database." https://www.postgresql.org/. [Online; accedido 20-abril-2024].

[55] Broadcom, "Rabbitmq." https://www.rabbitmq.com/. [Online; accedido 09-septiembre-2024].

[56] PrimeTek, "Primeng." https://primeng.org/. [Online; accedido 15-septiembre-2024].

[57] patrickvonplaten, "Confidence scores / self-training for wav2vec2 / ctc models." https://discuss.huggingface.co/t/confidence-scores-self-training-for-wav2vec2-ctc-models/17050. [Online; accedido 30-junio-2024].

[58] T. Pearce, A. Brintrup, and J. Zhu, "Understanding softmax confidence and uncertainty," 2021. https://arxiv.org/abs/2106.04972.

[59] factorial, "Calculadora de coste de un trabajador." https://proyectosagiles.org/que-es-scrum. [Online; accedido 28-junio-2024].

[60] M. de Trabajo y Economía Social, "Resolución de 27 de febrero de 2023, de la dirección general de trabajo, por la que se registra y publica el xx convenio colectivo nacional de empresas de ingeniería; oficinas de estudios técnicos; inspección, supervisión y control técnico y de calidad.." https://www.boe.es/diario_boe/txt.php?id=BOE-A-2024-4469#:~:text=Art%C3%ADculo%2015.&text=15.1%20Jornada%20anual%3A%20La%20jornada,la%20vigencia%20del%20presente%20convenio. [Online; accedido 28-junio-2024].

[61] C. of the EU and the European Council, "The general data protection regulation." https://www.consilium.europa.eu/en/policies/data-protection/data-protection-regulation/. [Online; accedido 28-junio-2024].

[62] Wikipedia, "Licencia de software libre." https://es.wikipedia.org/wiki/Licencia_de_software_libre. [Online; accedido 28-junio-2024].

[63] M. de cultura, "El derecho de autor." https://www.cultura.gob.es/dam/jcr:e35818ab-b5d2-42b8-a297-ce9637bd8e16/derechos-autor-c.pdf. [Online; accedido 28-junio-2024].

[64] D. V., "Guía sobre el aviso legal para tu página web." https://www.hostinger.es/tutoriales/aviso-legal-pagina-web?utm_campaign=Generic-Tutorials-DSA|NT:Se|LO:ES-t4&utm_medium=ppc&gad_source=1&gclid=CjwKCAjw4f6zBhBVEiwATEHFVmSiPV7kV3SSdze6Bf7S4NBfBrYAoHUwC_nr1xqaGQsPCTbyCXFVuxoCb5wQAvD_BwE, 2024. [Online; accedido 29-junio-2024].

[65] JOSEPABLOSARCO, "Lenguaje gherkin." https://josepablosarco.wordpress.com/2015/03/11/lenguaje-gherkin/. [Online; accedido 30-junio-2024].

[66] J. Segovia, "Ventajas y desventajas de postgresql." https://www.todopostgresql.com/ventajas-y-desventajas-de-postgresql/. [Online; accedido 18-septiembre-2024].