

**UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA**  
**DEPARTAMENTO DE ELECTRÓNICA**  
**VALPARAÍSO – CHILE**



**Diseño y Desarrollo de**  
**Plataforma de Apreciación Artística para**  
**Personas con Discapacidad Visual**

**Alexey Nikolay Mitjaew Hupat**

**Memoria de Titulación para optar al título de Ingeniero Civil Telemático**

**Profesor Guía:**

Patricio Olivares Roncagliolo

**Profesor Correferente:**

Nicolás Torres Rudloff

**Diciembre 2025**

## **Agradecimientos**

A mi familia,  
mi perro Elvis  
y el buen café.

## **Resumen**

Este documento expone el desarrollo de una plataforma accesible de apreciación artística para público con discapacidad visual. La propuesta consiste en transformar obras en representaciones sonoras multifacéticas, que integran descripciones narrativas detalladas, contexto histórico-artístico y atmósferas auditivas generadas mediante modelos multimodales de lenguaje, sistemas de síntesis de voz de alta y modelos de generación sonora basada en texto.

La solución expuesta propone facilitar el acceso al arte mediante una experiencia adaptada, con el objetivo de ofrecer alternativas de apreciación sensorial para personas con discapacidad visual.

**Palabras clave:** *Accesibilidad / Arte inclusivo / Inteligencia artificial / Sonificación*

## **Lista de Figuras**

Figura 1	Multimodal LLMs .....	10
Figura 2	Tacotron 2 Arquitecture .....	11
Figura 3	AudioLDM .....	12
Figura 4	Arquitectura General de Sistema .....	13
Figura 5	Ejemplo de segmentación de imagen en cuadrantes Obra: El Macho Cabrío, Francisco de Goya .....	15
Figura 6	Modelo de Datos de la Plataforma .....	16
Figura 7	Flujo de Navegación Frontend .....	17
Figura 8	Documentación Swagger de la API .....	18
Figura 9	Frontend de la Plataforma .....	19
Figura 10	Funcionalidad de Búsqueda en Frontend .....	19
Figura 11	Obra «El Aquejarre» de Francisco de Goya .....	20

## **Lista de Tablas**

Tabla 1	Muestra de Elementos Detectados en Obra «El Aquejarre» .....	23
---------	--	----

# Contenidos

Introducción .....	6
Objetivo General .....	7
Objetivos Particulares .....	7
Estructura .....	7
Marco Teórico .....	8
Modelos de Lenguaje Multimodales .....	8
Modelos Generativos de Audio .....	8
Estándares de Accesibilidad .....	9
Estado del Arte .....	10
Modelos de Lenguaje Multimodales .....	10
Modelos de Texto a Voz .....	11
Modelos Generativos de Ambiente Sonoro .....	12
Desarrollo de la Plataforma .....	13
API Gateway .....	14
Generador de Audio Descriptivo .....	14
Generador de Narraciones de Contexto .....	14
Generador de Sonidos Ambientales .....	15
Catálogo de Obras/Imagenes .....	16
Frontend .....	17
Limitaciones .....	17
Resultados .....	18
API Gateway .....	18
Frontend .....	19
Módulos Generativos .....	20
Generación de Texto de Audio Descriptivo .....	21
Generación de Texto de Narraciones de Contexto .....	22
Generador de Sonidos Ambientales .....	23

Audio Generado con TTS .....	24
Conclusiones .....	25
Bibliografía .....	26
Anexo .....	27
Utilidades Generales .....	27
Pipeline de Extracción de Fuentes .....	28
Generación de Contenido General .....	29
Utilidades Generales de Procesamiento de Imágenes .....	33
Narrador de Contexto .....	34
Generación de Audio Descriptivo .....	35
Generación de Sonidos Ambientales .....	36
Text to Speech .....	39

## **Introducción**

El acceso equitativo al arte constituye un componente esencial del desarrollo cultural y social, sin embargo, las personas con discapacidad visual continúan enfrentando barreras significativas para disfrutar de obras visuales en condiciones comparables al resto del público. Aunque las tecnologías de apoyo han avanzado, aún existe una brecha entre la experiencia estética que ofrecen los museos y plataformas digitales tradicionales y las necesidades sensoriales de quienes no pueden percibir elementos visuales directamente.

En este contexto, la inteligencia artificial emerge como una oportunidad para reimaginar la forma en que se transmite el contenido artístico, permitiendo crear experiencias inmersivas que integren narrativa, sonido y contextualización cultural. Este proyecto desarrolla una plataforma que transforma obras de arte en representaciones sonoras enriquecidas, combinando modelos multimodales capaces de interpretar imágenes, sistemas de síntesis de voz de alta naturalidad y tecnologías generativas orientadas a la creación de paisajes sonoros coherentes con la obra original.

La presente investigación se enmarca en los principios de accesibilidad universal y diseño inclusivo, procurando que la experiencia resultante no solo sea funcional, sino también significativa desde una perspectiva estética. La plataforma busca responder a la necesidad de ofrecer alternativas sensoriales que amplíen el acceso al patrimonio artístico y cultural, fortaleciendo la inclusión mediante herramientas tecnológicas avanzadas.

## **Objetivo General**

Mejorar la experiencia estética y promover la accesibilidad universal a las obras de arte para personas con discapacidad visual mediante el diseño e implementación de una plataforma accesible que utilice inteligencia artificial para transformar automáticamente imágenes de obras artísticas en paisajes sonoros narrativos, contextuales y ambientales.

## **Objetivos Particulares**

- **Proveer un catálogo curado de obras de dominio público**, basado en fuentes autoritativas y verificadas, que permita a los usuarios acceder a contenido confiable, culturalmente riguroso y legalmente seguro.
- **Generar descripciones auditivas de alta calidad** utilizando modelos de lenguaje multimodales capaces de interpretar intuitivamente los elementos visuales presentes en cada obra, garantizando una representación objetiva y comprensible de su contenido.
- **Crear narraciones históricas y biográficas** fundamentadas en fuentes reconocidas, con el fin de contextualizar la obra, su autor(a) y su relevancia artística dentro de un marco cultural accesible.
- **Producir ambientes sonoros inmersivos** que recreen atmósferas coherentes con la escena o época sugerida por la obra, mediante modelos generativos capaces de transformar imágenes en audio evocativo y sensorialmente Enriquecido.
- **Diseñar una interfaz centrada en accesibilidad**, que permita la navegación autónoma de personas en situación de discapacidad visual, incorporando criterios y normativas vigentes en materia de accesibilidad universal y diseño inclusivo.

## **Estructura**

El presente análisis incluye:

- **Revisión del estado del arte sobre modelos generativos y estándares de accesibilidad.**
- **Evaluación del marco teórico utilizado para el desarrollo de la plataforma.**
- **Descripción de la plataforma desarrollada a nivel de componentes.**
- **Revisión de resultados obtenidos y conclusiones al respecto.**

## **Marco Teórico**

### **Modelos de Lenguaje Multimodales**

Los modelos de lenguaje (LLMs, por sus siglas en inglés) representan sistemas computacionales diseñados para procesar, comprender y generar texto en lenguaje natural mediante arquitecturas basadas en aprendizaje profundo. Estos modelos, entrenados en vastos corpus textuales, capturan patrones lingüísticos, semánticos y contextuales, permitiendo tareas como la traducción automática, la generación de respuestas coherentes o el análisis de sentimientos. Su evolución ha dado paso a los modelos de lenguaje multimodales, los cuales extienden estas capacidades al integrar múltiples formas de información, como imágenes y audio.

Estos modelos multimodales no solo interpretan contenido visual o auditivo, sino que también generan descripciones detalladas de escenas, objetos o composiciones artísticas, traduciendo elementos visuales en narrativas estructuradas. Además, su capacidad multimodal les permite procesar entradas de voz y participar en flujos conversacionales voz a voz, facilitando interacciones más naturales y accesibles. Esta convergencia de modalidades (texto, imagen y sonido) facilita la creación de sistemas más ricos y adaptables, capaces de ofrecer interpretaciones contextualizadas y personalizadas según las necesidades de percepción del usuario.

### **Modelos Generativos de Audio**

Los modelos generativos de audio representan sistemas capaces de crear sonidos, música y voces artificiales a partir de descripciones semánticas o patrones aprendidos. Estos modelos, basados en arquitecturas de aprendizaje profundo, pueden sintetizar desde elementos literales —como el canto de pájaros, el sonido de una tormenta o pasos sobre grava— hasta composiciones musicales complejas, ajustándose a estilos, géneros o emociones específicas. Su funcionamiento se sustenta en la interpretación de instrucciones textuales o parámetros acústicos, permitiendo generar audio coherente y contextualizado

sin necesidad de muestras preexistentes. Esta capacidad no solo amplía las posibilidades creativas en producción musical y diseño sonoro, sino que también facilita la creación de entornos auditivos personalizados, como paisajes sonoros para aplicaciones de accesibilidad o asistentes de voz con tonos y matices más naturales.

## **Estándares de Accesibilidad**

Los estándares de accesibilidad proporcionan el marco normativo que guía el diseño de interfaces inclusivas, asegurando que los contenidos digitales sean utilizables por personas con diversas capacidades sensoriales, motoras o cognitivas. Entre los más relevantes se encuentran las Pautas de Accesibilidad para el Contenido Web (WCAG), que establecen criterios relacionados con la perceptibilidad, operabilidad, comprensibilidad y robustez de los sistemas web. En el contexto chileno, estas directrices se integran mediante la Ley N.º 20.422, que regula la igualdad de oportunidades e incorpora exigencias específicas para tecnologías asistivas. La plataforma desarrollada en este proyecto adopta estos lineamientos para garantizar compatibilidad con lectores de pantalla, navegación por teclado, descripciones alternativas, controles auditivos accesibles y una estructura de interacción que facilite la exploración autónoma.

El cumplimiento de estos estándares es esencial para asegurar que la solución propuesta no solo sea técnicamente avanzada, sino también verdaderamente inclusiva y centrada en las necesidades de la comunidad usuaria.

# Estado del Arte

## Modelos de Lenguaje Multimodales

Durante los últimos años los modelos multimodales de lenguaje han experimentado avances significativos, integrando capacidades de procesamiento de texto, imagen, audio y video. Este progreso ha sido impulsado por arquitecturas unificadas basadas en Transformers que permiten compartir un espacio de representación común entre modalidades. Avances como los *Vision-Language Encoders* (VLMs), los *Audio-Text aligners* y la *Tokenización Unificada* han permitido integrar imágenes, audio y texto como secuencias compatibles. Además, técnicas como el *cross-attention multimodal*, la *late fusion optimizada* y el *modality routing* han mejorado la coherencia entre modalidades, habilitando tareas complejas de razonamiento y generación conjunta en múltiples formatos.

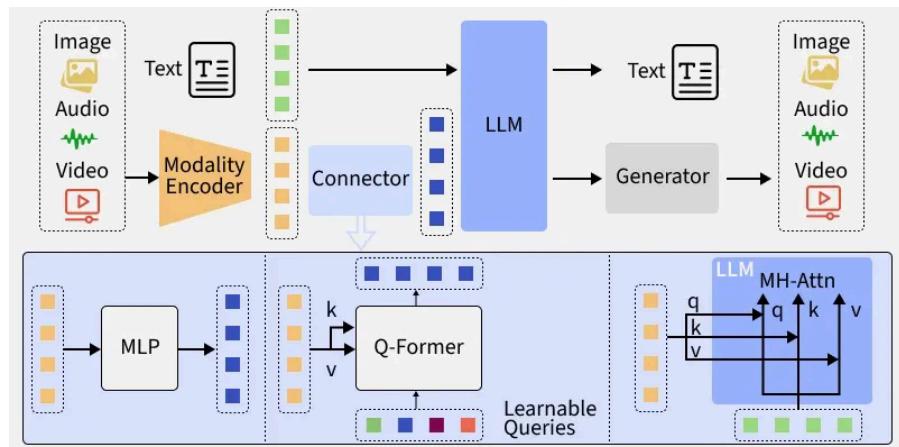


Figura 1: Multimodal LLMs

Entre los avances destacados hasta la fecha (2025) se encuentran:

- **Llama v4 Maverick (Meta)**: Modelo multimodal con descripción semántica detallada de imágenes y análisis de escenas. Pesos liberados para investigación.
- **GPT-4o (OpenAI) y Gemini 1.5 (Google)**: Modelos con integración nativa de texto, imagen y audio. Sin pesos liberados.
- **Voice-to-Voice (V2V)**: Sistemas como Whisper v3 (OpenAI) y SeamlessM4T (Meta) para transcripción y traducción en tiempo real. Voicebox (Meta) genera voz sintética con control de emociones. Whisper v3 tiene pesos liberados; SeamlessM4T y Voicebox no.

## Modelos de Texto a Voz

En los últimos años, arquitecturas como Tacotron 2, VITS, Kokoro TTS y modelos basados en difusión/transformers han ganado relevancia. Estos modelos, generalmente ligeros en cantidad de parámetros, son razonables de autohostear con GPUs de grado consumidor e incluso en CPU en algunos casos. Generan voces naturales con control sobre prosodia, entonación y emociones. Kokoro TTS, de código abierto, destaca por su expresividad y ajuste de voces mediante manipulación del espacio latente, evitando reentrenamiento.

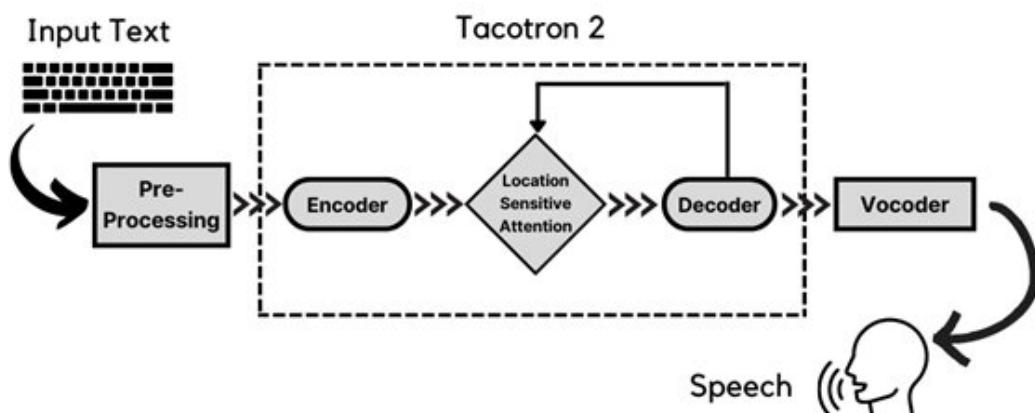


Figura 2: Tacotron 2 Arquitectura

Además de los modelos de código abierto, proveedores comerciales como Google Cloud, Amazon Polly, Microsoft Azure, ElevenLabs y OpenAI ofrecen soluciones TTS de alta calidad. Estas plataformas incorporan voces multilingües, opciones de personalización avanzada y capacidades de clonación vocal.

No obstante los avances observados, subsisten desafíos técnicos comunes en las alternativas investigadas, como la generación de expresividad controlable, la adaptación a entornos acústicos adversos y la síntesis eficiente de secuencias de larga duración.

## Modelos Generativos de Ambiente Sonoro

La generación de ambientes sonoros de alta calidad sigue en fase experimental, sin soluciones comerciales consolidadas. Modelos como CLAP permiten mapear texto y audio en un espacio semántico compartido, aunque no generan audio directamente.

Entre los modelos evaluados para la plataforma se encuentran:

- **I Hear Your True Colors:** La arquitectura propuesta en este paper combina el uso de un VQVAE, transformers y CLIP. El VQVAE extrae representaciones jerárquicas como secuencias discretas, los transformers las modelan de forma autorregresiva y CLIP alinea el audio con lo visual. Aunque el código es abierto, no existen fuentes públicas de pesos preentrenados. Sin versiones comerciales.
- **AudioLDM:** Utiliza difusión para generar espectrogramas condicionados por texto, incorporando embeddings de CLAP para mejorar la alineación semántica entre descripciones textuales y contenido sonoro. Presenta resultados prometedores en coherencia semántica y diversidad acústica. El modelo preentrenado se puede encontrar con pesos abiertos.

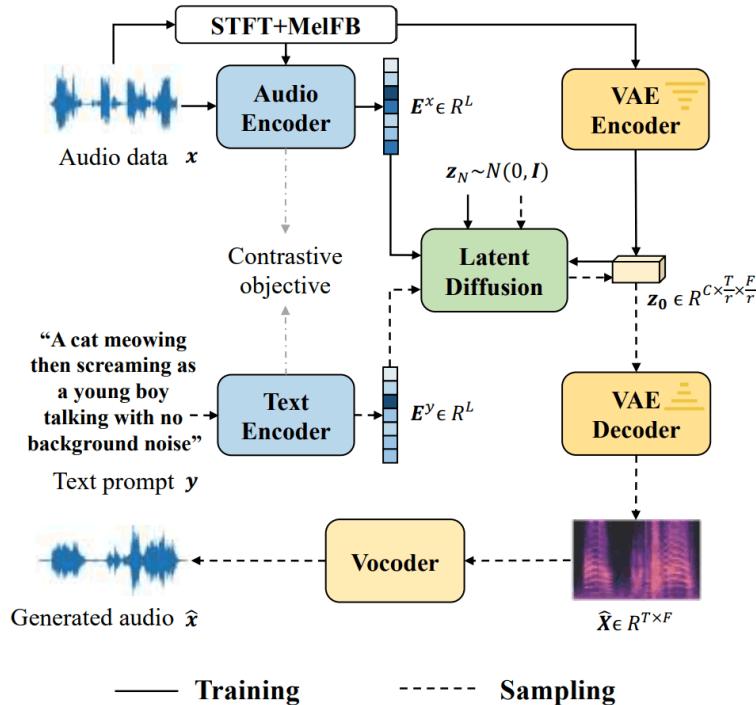


Figura 3: AudioLDM

## Desarrollo de la Plataforma

La arquitectura propuesta sigue un modelo cliente-servidor e incorpora un *API Gateway* como componente central. Este elemento cumple dos funciones principales: por un lado, centraliza las operaciones computacionalmente intensivas (principalmente inferencia de modelos de aprendizaje automático y las solicitudes a APIs externas), y por otro, gestiona un catálogo centralizado de recursos. Además, se desarrolla un *frontend* que aplica principios de accesibilidad web, con especial atención a las necesidades de usuarios con discapacidad visual.

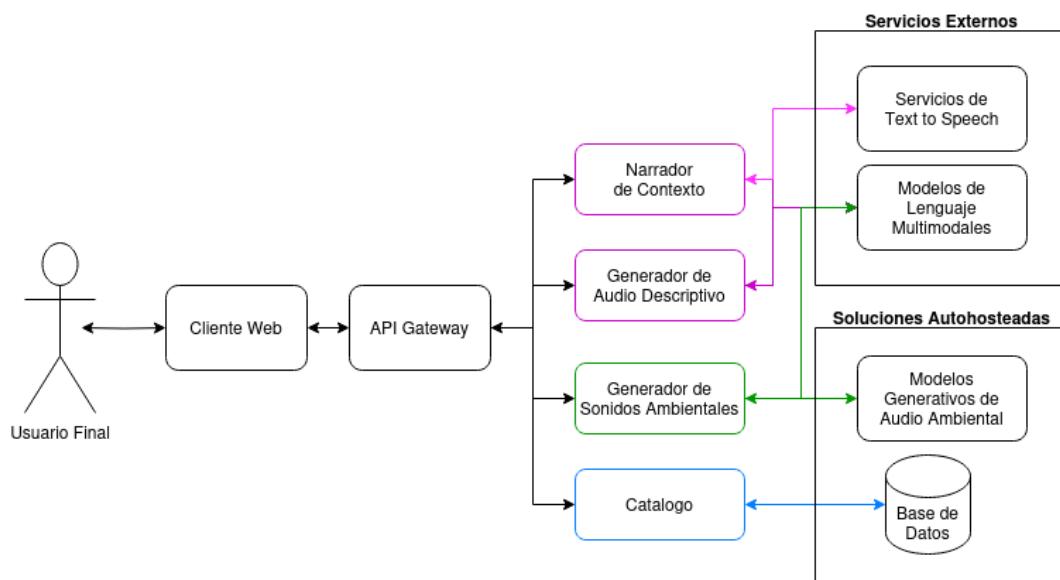


Figura 4: Arquitectura General de Sistema

## **API Gateway**

La API Gateway consiste en un servidor de Django, encargado de controlar 4 servicios:

### **Generador de Audio Descriptivo**

Este módulo se encarga de producir descripciones textuales detalladas y estructuradas de los contenidos visuales presentes en las obras. Para ello, emplea modelos de lenguaje multimodales, los cuales procesan tanto elementos gráficos como metadatos asociados mediante prompts especializados. Dichos prompts están diseñados para extraer información semántica relevante, garantizando una representación fiel y contextualizada del material original (Código 13). Posteriormente, las descripciones generadas son convertidas a formato de audio mediante sistemas de síntesis de voz (TTS) (Código 18), optimizados para ofrecer una experiencia auditiva clara y accesible.

### **Generador de Narraciones de Contexto**

Este módulo genera narraciones auditivas que contextualizan históricamente cada obra. Para ello, se recopila información de fuentes autoritativas (en este caso, se procesó manualmente un conjunto de datos con artículos de Wikipedia vinculados a cada pieza) (Código 7). A partir de estos datos, se elabora un relato contextualizado y adaptado al caso de uso (Código 12), que posteriormente se convierte en audio mediante modelos de síntesis de voz (Text to Speech) (Código 18), asegurando una experiencia auditiva coherente y accesible.

### **Generador de Sonidos Ambientales**

Este módulo se encarga de la generación de ambientes sonoros que representan los contenidos literales (no abstractos) de sus imágenes de entrada. La heurística utilizada consiste en:

- Dividir el espacio en cuadrantes, en nuestro caso 9 (Código 14), como se puede apreciar en Figura 5.
- Procesar cada cuadrante con LLMs multimodales, obteniendo salidas estructuradas en JSON (Código 15) representando cada elemento detectado.
- Por cada cuadrante generar una mezcla de sonido integrando todos los elementos detectados en cada respectiva sección. (Código 16)



Figura 5: Ejemplo de segmentación de imagen en cuadrantes

Obra: El Macho Cabrío, Francisco de Goya

## Catálogo de Obras/Imagenes

Se elaboró un conjunto de datos en formato JSON que incluye 30 obras artísticas, conteniendo todo el material requerido para el funcionamiento de los sistemas previamente descritos. Este dataset incorpora, además, enlaces a los artículos correspondientes de Wikipedia asociados a cada obra artística. (**ANEXO**)

Una vez procesado el conjunto de datos, sus campos son expuestos a través de la **API** mediante endpoints específicos. Los datos textuales se almacenan en una base de datos SQLite, mientras que las imágenes y archivos de audio generados se guardan en el sistema de archivos local del backend.

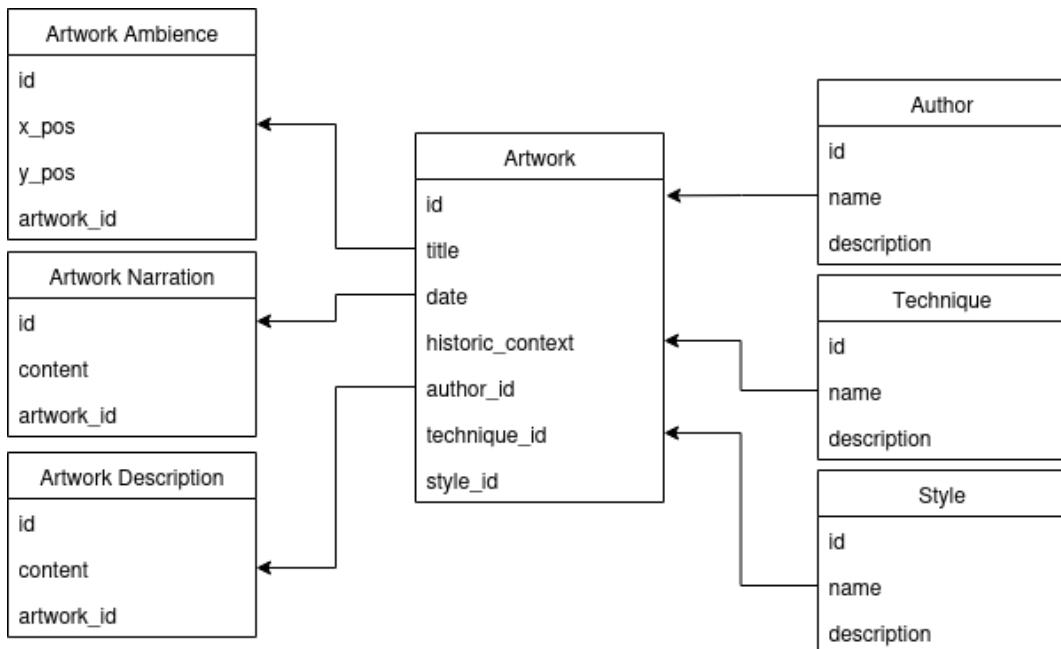


Figura 6: Modelo de Datos de la Plataforma

## Frontend

El frontend se implementó utilizando NextJS para proporcionar acceso al contenido gestionado por la API Gateway. En su diseño, se incorporaron textos alternativos y etiquetas HTML semánticas con el objetivo de garantizar una integración completa con herramientas de asistencia para usuarios con discapacidad visual. Asimismo, se desarrolló una interfaz de usuario que facilita la navegación mediante atajos de teclado, optimizando la accesibilidad y usabilidad del sistema.

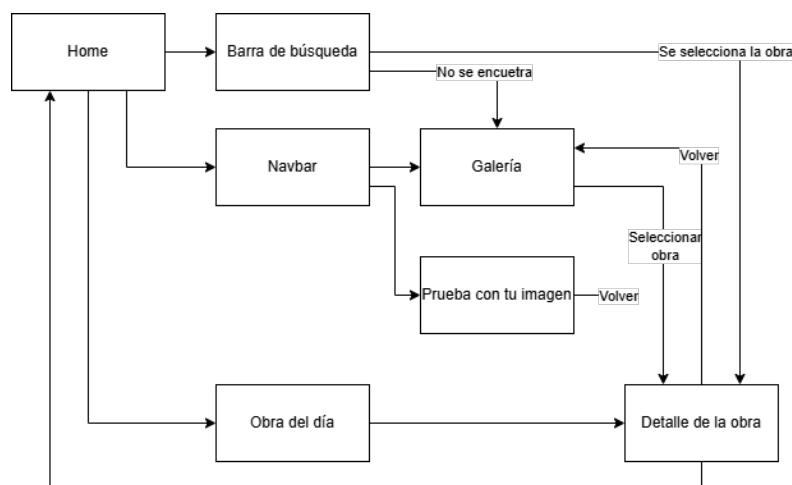


Figura 7: Flujo de Navegación Frontend

## Limitaciones

Por limitaciones computacionales durante el desarrollo, los módulos no se integraron directamente en la API Gateway. En su lugar, se implementó un prototipo funcional en un cuadernillo Jupyter para validar el diseño y flujo de procesamiento. Esto permitió generar y preprocesar los conjuntos de datos necesarios, que fueron incorporados a la base de datos para su uso en la versión final.

**Cuadernillo:** <https://www.kaggle.com/code/alexeymitjaew/sound-generation-pipeline>

# Resultados

## API Gateway

El sistema opera correctamente, incluye autodocumentación en *Swagger* y *OpenAPI* y un conjunto completo de operaciones CRUD para todos los modulos de la plataforma.

Debido a las limitaciones computacionales mencionadas previamente, los endpoints solamente habilitan operaciones de lectura y escritura en la base de datos y sistema de archivos para facilitar la subida de elementos preprocesados.

El sistema permite tanto la consulta general de los elementos registrados (obras, autores, períodos históricos, entre otros) como la gestión individual de estos en cada uno de los módulos correspondientes.

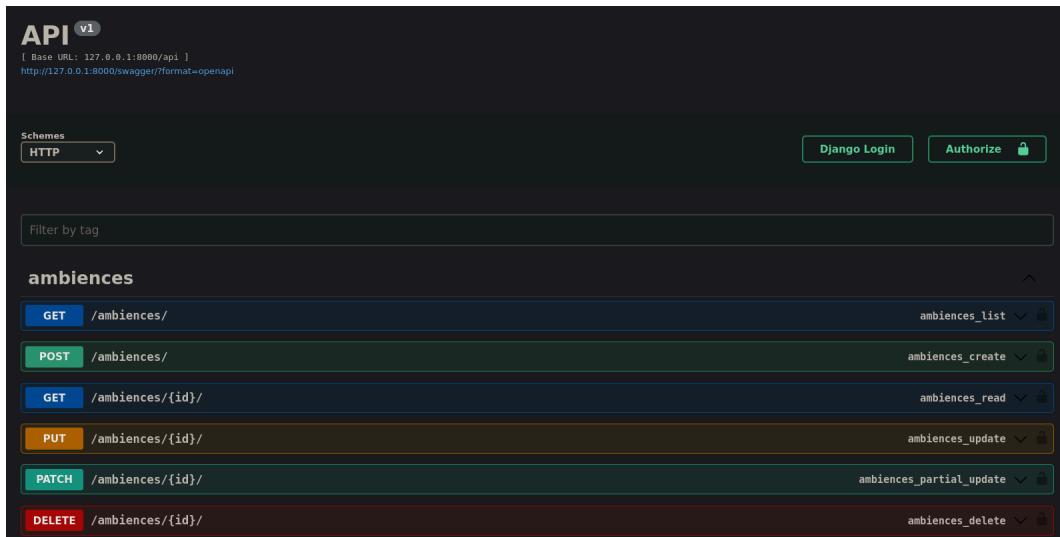


Figura 8: Documentación Swagger de la API

Para garantizar compatibilidad con APIs externas de servicios de cómputo, los datos no estructurados (imágenes y audio) se transmiten codificados en base64. Este método, aunque aumenta el volumen de datos (alrededor del 30%), facilita la integración con SDKs de proveedores de inferencia compatibles con la API de OpenAI (como Groq, Novita o TogetherAI). Como mejora futura, se propone adoptar multipart-form-data en la subida y carga de archivos para optimizar el rendimiento de la plataforma.

## Frontend

Aunque no se realizaron pruebas con los usuarios objetivo, se evaluó el funcionamiento de la navegación dentro del sitio mediante herramientas de accesibilidad diseñadas para usuarios con discapacidad visual.

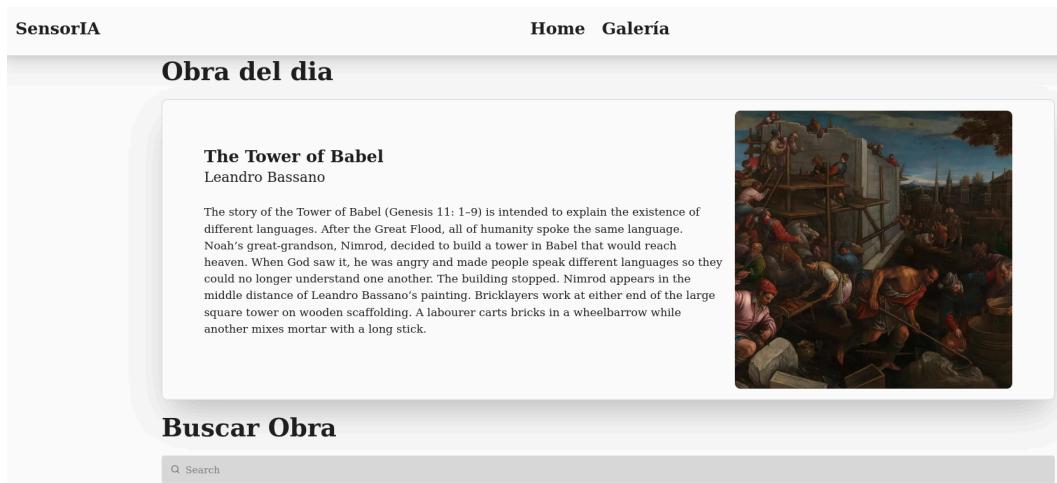


Figura 9: Frontend de la Plataforma

La funcionalidad de navegación implementa correctamente búsquedas por obra, autor, período y técnica. Sin embargo, el sistema emplea un mecanismo de coincidencia de cadenas que no considera variaciones ortográficas como los acentos, lo que introduce cierta rigidez en la experiencia de usuario.

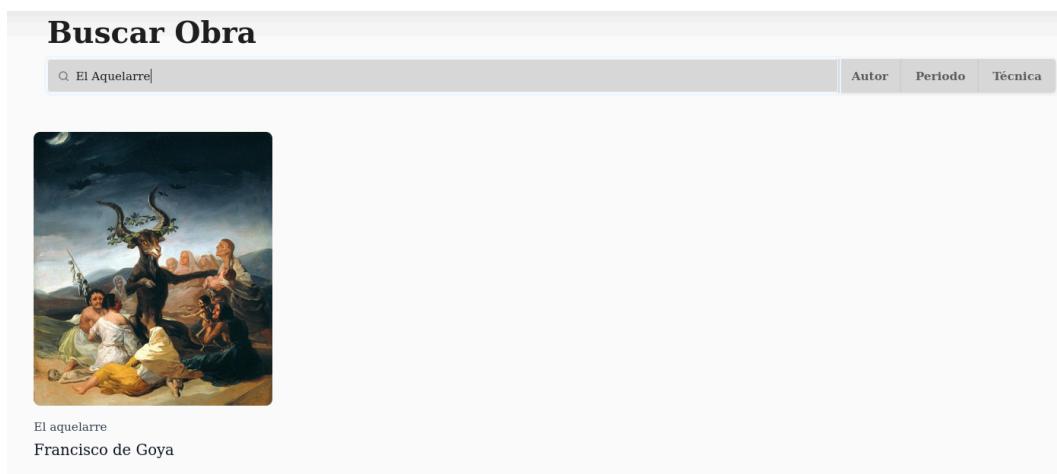


Figura 10: Funcionalidad de Búsqueda en Frontend

La implementación de la visualización de obras y sus módulos asociados opera de manera adecuada, logrando una integración efectiva del contenido audiovisual.

## Módulos Generativos

A continuación se exponen los resultados obtenidos en los módulos generativos de la plataforma. Tal como se indicó en la sección anterior, estos componentes fueron implementados en un entorno independiente (cuadernillo hosteado en Kaggle) como parte de un desarrollo modular complementario al sistema principal.

Como caso de estudio representativo, se ha seleccionado la obra **El Aquejarre** de Francisco de Goya para ilustrar el funcionamiento de los módulos.

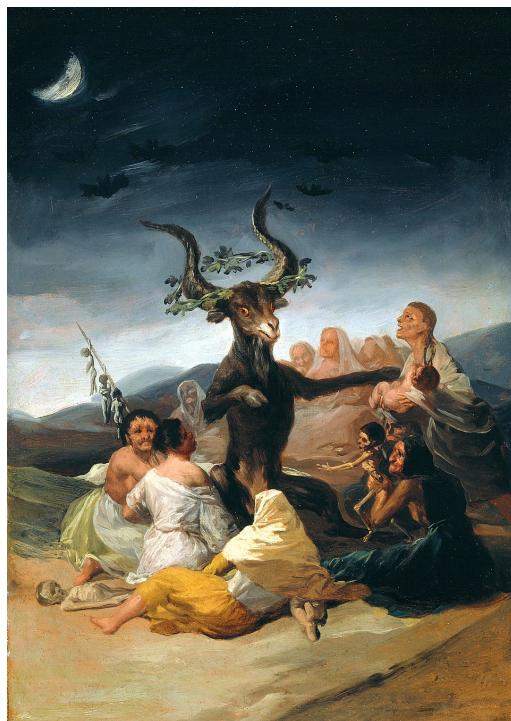


Figura 11: Obra «El Aquejarre» de Francisco de Goya

A continuación, se evalúan los siguientes aspectos de los módulos implementados

- La calidad del texto en los módulos de *Audio Descriptivo* y *Narraciones de Contexto*.
- La fidelidad y realismo del audio producido por el *Generador de Sonidos Ambientales*.
- La naturalidad y claridad de la voz sintetizada empleada en las pruebas de concepto.

## **Generación de Texto de Audio Descriptivo**

Los resultados obtenidos demuestran una notable correspondencia entre las imágenes de entrada y las descripciones generadas, las cuales mantienen una estructura sintáctica y una cadencia adecuadas para su posterior procesamiento mediante modelos de *Text to Speech*. Este hallazgo sugiere que una estrategia basada en *prompting directo* resulta suficiente para que un modelo de lenguaje multimodal genere descripciones precisas del contenido visual de una obra, sin requerir fuentes autoritativas externas (como artículos de Wikipedia) de referencia para complementar el análisis.

### **El Aquellare:**

En la imagen se puede observar una representación de un ritual o acto de adoración hacia un macho cabrío. La figura del macho cabrío viste una corona de hojas y se encuentra en el centro de la composición. Está rodeado de personas que parecen estar participando en algún tipo de ceremonia.

Al fondo de la escena se puede ver una luna creciente en el cielo nocturno, lo que sugiere que la escena se desarrolla durante la noche. Detrás del macho cabrío hay un grupo de personas vestidas con túnicas, algunas de las cuales tienen la cara cubierta.

La escena general parece representar una temática relacionada con prácticas antiguas o rituales paganos.

Texto 1: Descripción de Obra El Aquelearre de Francisco de Goya

## **Generación de Texto de Narraciones de Contexto**

Como es señalado en Código 12, la generación de la narración toma por entrada un artículo de Wikipedia, en este caso

[https://es.wikipedia.org/wiki/El\\_aquelarre\\_\(1798\)](https://es.wikipedia.org/wiki/El_aquelarre_(1798))

Una comparación cruzada entre el texto generado y la información contenida en el artículo de Wikipedia demuestra una adecuada correspondencia con la fuente original, logrando transformar el contenido enciclopédico en una narración fluida y de carácter natural.

El pipeline cumple sus objetivos al generar narraciones contextuales fluidas y coherentes. Aunque necesita ajustes para casos como anglicismos o siglas, evita alucinaciones comunes en consultas directas a modelos de lenguaje.

### **El Aquellare:**

En el palacio de recreo de los duques de Osuna, situado en la finca de la Alameda de Osuna, a las afueras de Madrid, Francisco de Goya pintó seis pequeños cuadros entre 1797 y 1798. Uno de ellos, "El aquellarre", es una obra que representa un ritual de brujería presidido por un gran macho cabrío, forma del demonio, rodeado de brujas ancianas y jóvenes que entregan niños, según la superstición de la época, para alimentar al demonio. La escena se desarrolla bajo un cielo nocturno iluminado por la luna, con animales nocturnos volando en el fondo, creando una atmósfera siniestra y oscura.

La serie a la que pertenece "El aquellarre" forma parte de la estética del "sublime terrible", una corriente del prerromanticismo que buscaba generar desasosiego y una sensación de pesadilla en el espectador. Goya estaba trabajando en "Los Caprichos" en ese momento, una obra estrechamente relacionada temáticamente con la serie de cuadros para los duques de Osuna. La brujería era un tema de actualidad entre los ilustrados españoles amigos de Goya, y su interés en este tema se refleja en la ambientación nocturna y los tonos oscuros que predominan en "El aquellarre". La obra forma parte de la colección de la Fundación Lázaro Galdiano, después de ser adquirida por José Lázaro Galdiano en 1928.

Texto 2: Narración de Obra El Aquellarre de Francisco de Goya

## Generador de Sonidos Ambientales

El proceso de detección de elementos acústicos en la obra presenta un desempeño adecuado, aunque se observan ocasionales alucinaciones menores en la identificación de componentes específicos. No obstante, los elementos detectados mantienen coherencia temática con la obra analizada, resultado atribuible al mapeo en el espacio latente implementado en la pipeline visual de Llamav4 Maverick.

Segmento	Sonidos Generados						
	<table border="1"> <tr> <td><b>Objeto</b></td><td>goat bleating</td></tr> <tr> <td><b>Fondo</b></td><td>tree branch creaking, people whispering</td></tr> <tr> <td><b>Link</b></td><td>POR HACER</td></tr> </table>	<b>Objeto</b>	goat bleating	<b>Fondo</b>	tree branch creaking, people whispering	<b>Link</b>	POR HACER
<b>Objeto</b>	goat bleating						
<b>Fondo</b>	tree branch creaking, people whispering						
<b>Link</b>	POR HACER						
	<table border="1"> <tr> <td><b>Fondo</b></td><td>sand</td></tr> <tr> <td><b>Objeto</b></td><td>puppy sleeping</td></tr> <tr> <td><b>Link</b></td><td>POR HACER</td></tr> </table>	<b>Fondo</b>	sand	<b>Objeto</b>	puppy sleeping	<b>Link</b>	POR HACER
<b>Fondo</b>	sand						
<b>Objeto</b>	puppy sleeping						
<b>Link</b>	POR HACER						

Tabla 1: Muestra de Elementos Detectados en Obra «El Aquejarre»

Respecto a la calidad del audio generado se puede decir que aunque la mayoría de los cuadrantes producen resultados verosímiles y coherentes con las escenas representadas, se observa un grado variable de ruido y artefactos en las mezclas de sonido y se identifican *outliers* donde los ambientes degeneran en ruido. Este comportamiento sugiere limitaciones en la generalización del modelo ante elementos visuales ambiguos y/o composiciones atípicas dentro de las obras.

## **Audio Generado con TTS**

La implementación del módulo de Text-to-Speech (TTS) empleando Kokoro TTS demostró un nivel de naturalidad adecuado para la generación de audio en español, con una entonación fluida y una pronunciación generalmente clara de los textos descriptivos y narrativos. El tono neutral adoptado por el modelo resulta apropiado para una aplicación de accesibilidad cultural, ya que evita sesgos emocionales que podrían interferir con la percepción objetiva del contenido.

No obstante, se identificaron dificultades específicas en la pronunciación de anglicismos y siglas. Términos y nombres propios de otros idiomas son interpretados con reglas fonéticas del español, generando resultados poco intuitivos para el oyente. Esta limitación sugiere la necesidad de implementar un preprocesamiento que genere descripciones alternativas para dicho contenido (lo que fué implementado parcialmente en las estrategias de prompting de la generación de narraciones de contexto, ver Código 12) o, alternativamente, incorporar un diccionario de excepciones para preservar la inteligibilidad en contextos técnicos.

**añadir link a wav**

## Conclusiones

Este trabajo presentó el diseño, implementación y evaluación de una plataforma para mejorar el acceso al patrimonio artístico mediante inteligencia artificial generativa, con especial atención a la accesibilidad para personas con discapacidad visual.

Los resultados obtenidos evidencian que los modelos multimodales empleados son capaces de generar descripciones visuales precisas y estructuradas, así como narraciones de contexto históricamente coherentes, sin incurrir en alucinaciones significativas cuando se utilizan estrategias de prompting y fuentes autoritativas controladas. Asimismo, la generación de ambientes sonoros, si bien presenta limitaciones en términos de ruido y estabilidad en casos específicos, logra aportar una dimensión sensorial adicional que enriquece la experiencia auditiva y refuerza la comprensión del contenido visual de las obras.

Desde el punto de vista de la accesibilidad, el frontend desarrollado incorpora buenas prácticas de diseño inclusivo, tales como el uso de semántica HTML, textos alternativos y elementos optimizados para navegación por teclado, lo que permite una integración efectiva con herramientas de asistencia. Aunque no se realizaron pruebas con usuarios finales, las evaluaciones preliminares indican que la plataforma cumple con los objetivos funcionales propuestos y sienta una base sólida para futuras validaciones con el público objetivo.

Finalmente, las limitaciones computacionales que obligaron a separar los módulos generativos del sistema principal no invalidan la propuesta, aunque plantean el desafío de incorporar servicios que integren de manera eficiente tanto los módulos de inferencia autohosteados (AudioLDM y Kokoro TTS en este caso) como servicios externos de inferencia. Como trabajo futuro, se propone la integración completa de los módulos en la API Gateway, la optimización del manejo de archivos multimedia, la mejora en la robustez del generador de sonidos ambientales y la realización de estudios de usabilidad con usuarios con discapacidad visual.

[1] said [2] said

## Bibliografía

- [1] Ö. Aksin *et al.*, «Effect of immobilization on catalytic characteristics of saturated Pd-N-heterocyclic carbenes in Mizoroki-Heck reactions», *J.~Organomet. Chem.*, vol. 691, n.º 13, pp. 3027-3036, 2006.
- [2] G. Westfahl, «The True Frontier». pp. 55-65.

## Anexo

### Utilidades Generales

En la prueba de concepto se emplearon modelos de lenguaje proporcionados por la plataforma *Groq*, accedidos mediante el cliente de Python de OpenAI. Este estándar es compatible con la mayoría de los proveedores de servicios de inferencia para modelos de lenguaje.

```
1 client = openai.OpenAI(  
2     base_url = "https://api.groq.com/openai/v1",  
3     api_key = user_secrets.get_secret("GROQ_API_KEY")  
4 )
```

Python

Código 3: Inicialización de Cliente

Adicionalmente a lo anterior, se implementó una abstracción genérica para el procesamiento de **prompts**. Esta función permite definir parámetros mediante notación de llaves (por ejemplo, {clave}) para su posterior sustitución mediante argumentos nombrados al invocar la función (ejemplo: PARSE\_PROMPT(prompt, clave=valor)).

```
1 def PARSE_PROMPT(BASE_PROMPT, **kwargs):  
2     for key in kwargs:  
3         BASE_PROMPT = BASE_PROMPT.replace(  
4             '{' + str(key) + '}',  
5             str(kwargs[key]))  
6     )  
7     return BASE_PROMPT
```

Python

Código 4: Utilidad de Procesado de Prompts

## Pipeline de Extracción de Fuentes

Para extraer información procesada de fuentes autoritativas se procedió con el desarrollo de utilidades para extraer datos desde una URL y procesarlo en forma de bulletpoint para presentar los resultados de manera estructurada y clara en forma de viñetas.

```
1  def extract_text_from_url(url: str) -> str: Python
2      headers = {
3          "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:140.0)
4          Gecko/20100101 Firefox/140.0",
5      }
6      response = requests.get(url, headers=headers).raise_for_status()
7      soup = BeautifulSoup(response.text, 'html.parser')
8
9      for tag in soup(['script', 'style']):
10         tag.decompose()
11
12     paragraphs = [p.get_text(strip=True) for p in soup.find_all('p')]
13
14     return '\n\n'.join(paragraphs)
```

Código 5: Scraper Extractor de Información

```
SOURCE_EXTRACTOR_PROMPT = """Eres un extractor de
información precisa y concisa. Tu tarea es leer el texto de
1  un artículo y devolver una lista de viñetas (*) con todos Python
los hechos y datos relevantes, escritos en español natural
y claro.
2
3  Instrucciones:
4      1. Identifica los hechos principales, cifras, declaraciones, fechas,
5      nombres y conclusiones clave del texto.
6      2. No incluyas opiniones, lenguaje publicitario o frases
7      irrelevantes.
8      3. Si el texto está en inglés u otro idioma, traduce los puntos al
9      español correctamente.
10     4. Usa frases breves pero completas, cada una iniciando con un punto
11     (*).
12     5. Mantén el tono informativo y objetivo.
13
14  Texto del artículo: {TEXT}
15  Tu respuesta:"""
```

Código 6: Prompt de Extracción de Información

```

1  def extract_url_context(url):
2      text = extract_text_from_url(url)[:8_000]
3
4      completion = client.chat.completions.create(
5          model="openai/gpt-oss-120b",
6          messages=[{
7              "role": "user",
8              "content": [{{
9                  "type": "text",
10                 "text": PARSE_PROMPT(SOURCE_EXTRACTOR_PROMPT,
11                 TEXT=text)
12             }]}
13         ],
14         temperature=0,
15         max_completion_tokens=2048,
16         top_p=1,
17         stream=False,
18         stop=None,
19     )
20     content = completion.choices[0].message.content
21
22     return content

```

Python

Código 7: Script de Extracción de Información

## Generación de Contenido General

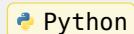
Para los metadatos generales de la plataforma, aunque no visibles en el **frontend** de la prueba de concepto, se pobló la base de datos con períodos, técnicas pictóricas y breves biografías de los autores. Estos metadatos, aunque no visibles en el **frontend** de la prueba de concepto, proporcionan información contextual esencial para la interpretación y categorización de las obras de arte que en una etapa posterior del proyecto podrían enriquecer la experiencia del usuario.

Pasa la generación de estos se usaron dos enfoques distintos. Para técnicas pictóricas y períodos artísticos, se empleó generación directa con modelos de lenguaje. En cambio, para biografías de autores se extrajo información de Wikipedia para evitar imprecisiones en datos biográficos complejos, priorizando la calidad de la fuente.

```

1 TECHNIQUE_PROMPT = \
2     """Expícame en detalle una técnica de pintura o dibujo.
3 Considera lo siguiente:
4 - Debe estar en un formato adecuado para narración.
5 - Debe ser conciso y corto en duración.
6 - NO digas en resumen.
7 - NO seas redundante.
8     - NO utilices títulos ni subtítulos, solo escribe en párrafos
9     narrados.
10 Técnica: {TECHNIQUE}
11 Explicación:"""
12
13 def get_technique(technique):
14     completion = client.chat.completions.create(
15         #model="llama-3.1-8b-instant",
16         model="llama-3.3-70b-versatile",
17         messages=[{
18             "role": "user",
19             "content": [{{
20                 "type": "text",
21                 "text": PARSE_PROMPT(TECHNIQUE_PROMPT,
22                                     TECHNIQUE=technique)
23             }]
24         }],
25         temperature=0,
26         max_completion_tokens=1024,
27         top_p=1,
28         stream=False,
29         stop=None,
30     )
31     content = completion.choices[0].message.content
32     return content

```



Código 8: Generador de Texto de Técnica

Como se ha detallado previamente, la generación de este contenido requiere únicamente el nombre de la técnica o estilo pictórico a describir. Además, se han implementado consideraciones específicas en cuanto a la cadencia y el estilo del texto generado, con el objetivo de optimizar su procesamiento mediante modelos de síntesis de voz (TTS).



```
1 PERIOD_PROMPT = \  
2 """Expícame en detalle un estilo pictórico en particular.  
3 Considera lo siguiente:  
4 - Debe estar en un formato adecuado para narración.  
5 - Debe ser conciso y corto en duración.  
6 - NO digas en resumen.  
7 - NO seas redundante.  
8 - NO utilices títulos ni subtítulos, solo escribe en párrafos  
narrados.  
9  
10 Estilo pictórico: {PERIOD}  
11 Explicación:  
12  
13 def get_period(period):  
14     completion = client.chat.completions.create(  
15         #model="llama-3.1-8b-instant",  
16         model="llama-3.3-70b-versatile",  
17         messages=[{  
18             "role": "user",  
19             "content": [{  
20                 "type": "text",  
21                 "text": PARSE_PROMPT(PERIOD_PROMPT, PERIOD=period)  
22             }]  
23         },  
24         temperature=0,  
25         max_completion_tokens=1024,  
26         top_p=1,  
27         stream=False,  
28         stop=None,  
29     )  
30     return completion.choices[0].message.content
```

Código 9: Generador de Texto de Período

En la generación de la biografía utilizó el script de extracción de información (Código 7) para procesar un breve texto narrado de la vida del autor, tomando datos de Wikipedia.

```
1 AUTHOR_BIO_PROMPT = \Python
2 """Eres un experto en redacción biográfica y narrativa.
   Tu tarea es leer el siguiente texto y redactar una biografía breve y
3 fluida del autor o artista mencionado, adecuada para acompañar una
   ficha de obra de arte.
4 Instrucciones:
5   1. Resume los aspectos esenciales de la vida y obra del autor:
6     formación, estilo, periodo histórico, aportes y relevancia artística.
7   2. Mantén un tono natural, informativo y narrativo, sin usar listas
8     ni formato enciclopédico.
9   3. Evita redundancias, tecnicismos innecesarios y frases genéricas.
10  4. No incluyas fechas o datos no presentes en el texto fuente.
11  5. No menciones el proceso de redacción ni expresiones como "esta
12    biografía trata sobre".
13  6. Si el texto está en otro idioma, tradúcelo al español de manera
14    natural.
15  7. Extensión máxima: dos párrafos.
16  8. Evita completamente el uso de caracteres especiales
17
18  Texto: {TEXT}
19  Tu respuesta:"""
20
21  def get_bio(url):
22      text = extract_url_context(url)
23      completion = client.chat.completions.create(
24          model="llama-3.3-70b-versatile",
25          messages=[{
26              "role": "user",
27              "content": [{

28                  "type": "text",
29                  "text": PARSE_PROMPT(AUTHOR_BIO_PROMPT, TEXT=text)
30              }]
31          },
32          temperature=0,
33      )
34
35      return completion.choices[0].message.content
```

,

Código 10: Author Bio Generation

## Utilidades Generales de Procesamiento de Imágenes

A continuación se definen utilidades de procesamiento de imágenes para el preprocesamiento básico antes de realizar llamadas a proveedores de servicios de inferencia o su uso en modelos autohospedados.

- **resize\_if\_oversized** ajusta imágenes manteniendo proporciones, evitando que excedan un tamaño máximo para optimizar almacenamiento y rendimiento.
- **download\_img** descarga imágenes desde URLs usando un **user-agent** personalizado (para evitar bloqueos de conexión) y las redimensiona si superan el límite, garantizando su adaptación al sistema.
- **image\_to\_base64** convierte imágenes PIL a base64, facilitando su transmisión y almacenamiento como texto plano para integración en APIs o bases de datos.

```
1  def resize_if_oversized(image, max_dim=1024):          Python
2      width, height = image.size
3      if max(width, height) <= max_dim:
4          return image # nothing to do
5
6      # scale preserving proportions
7      scale = max_dim / max(width, height)
8      new_size = (int(width * scale), int(height * scale))
9      return image.resize(new_size, Image.Resampling.LANCZOS)
10
11 def download_img(url, max_dim: int = 1024):
12     headers = { "User-Agent": "Chrome/51.0.2704.103 Safari/537.36" }
13     r = requests.get(url, headers=headers)
14     im = Image.open(BytesIO(r.content))
15     im = resize_if_oversized(im, max_dim)
16     return im
17
18 def image_to_base64(image: Image.Image, format="PNG") -> str:
19     """Convert PIL Image to base64 string."""
20     buffered = BytesIO()
21     image.save(buffered, format=format)
22     return base64.b64encode(buffered.getvalue()).decode("utf-8")
```

Código 11: Utilidades de Procesamiento de Imágenes

## Narrador de Contexto

Este modulo toma como entrada una *URL* de un articulo de Wikipedia para generar una narración didáctica sobre el contexto en que una obra se produjo.

Utiliza el módulo descrito en Código 7 para la extracción de la información.

```
1 SOURCE_SUMMARY_PROMPT = \
2     """Eres un experto en redacción narrativa. Tu tarea es leer el
3     siguiente texto y escribir un resumen breve (máximo dos párrafos) en
4     un estilo natural, fluido y adecuado para narración oral o escrita.
5
6 4 Instrucciones:
7     1. Mantén un tono narrativo, como si estuvieras contando los hechos
8     de manera clara y atractiva.
9     2. Sé conciso: evita repeticiones, rodeos o frases innecesarias.
10    3. No incluyas títulos, subtítulos ni listas.
11    4. No uses expresiones como "en resumen", "este artículo trata
12      sobre", ni menciones al proceso de resumen.
13    5. Si el texto está en otro idioma, tradúcelo al español con
14      naturalidad.
15    6. Enfócate en los hechos principales y el contexto histórico de la
16      producción de la obra.
17
18 11
19 12 Texto: {TEXT}
20 13 Tu respuesta:"""
21
22 14
23 15 def get_narration(url):
24     text = extract_url_context(url)
25     completion = client.chat.completions.create(
26         model="llama-3.3-70b-versatile",
27         messages=[{
28             "role": "user",
29             "content": [
30                 {
31                     "type": "text",
32                     "text": PARSE_PROMPT(SOURCE_SUMMARY_PROMPT,
33                                         TEXT=text)
34                 }
35             ],
36             "temperature=0,
37         })
38     return completion.choices[0].message.content
```

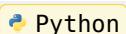
Código 12: Generación de texto para narraciones

## Generación de Audio Descriptivo

Este modulo toma como entrada una *URL* a una imagen, para luego procesarla en una descripción objetiva de los elementos representados.

Para el procesamiento de las imágenes se utilizan las utilidades definidas en Código 11.

Para la generación del texto se emplea un **prompt** estructurado que guía al modelo en la creación de descripciones literales del contenido de la obra con cadencia de narración oral.

```
1 DESCRIPTION_PROMPT = \ Python
2 """Describe los elementos retratados en la imagen.
3 Considera lo siguiente:
4 - Debe estar en un formato adecuado para narración.
5 - Debe ser conciso y corto en duración.
6 - NO digas en resumen.
7 - NO seas redundante.
8     - NO utilices títulos ni subtítulos, solo escribe en párrafos
9 narrados.
10 Elementos retratados: """
11
12 def get_description(url):
13     image = download_img(url)
14     b64_image = image_to_base64(image)
15     completion = client.chat.completions.create(
16         model="meta-llama/llama-4-scout-17b-16e-instruct",
17         messages=[{
18             "role": "user",
19             "content": [
20                 { "type": "text",
21                   "text": DESCRIPTION_PROMPT },
22                 { "type": "image_url",
23                   "image_url": {
24                     "url": f"data:image/png;base64,{b64_image}"
25                 } }
26             ]
27         }],
28         temperature=0,
29     )
30     return completion.choices[0].message.content
```

Código 13: Generación de texto para audios descriptivos

## Generación de Sonidos Ambientales

La euristica utilizada para la generación de sonidos ambientales consiste en 3 pasos:

1. **Segmentación de la imagen** (Código 14): La obra se divide en cuadrantes mediante un algoritmo de particionamiento espacial. Cada cuadrante se utiliza posteriormente como entrada para los pasos posteriores.
2. **Extracción semántica de elementos** (Código 15): Mediante Llama 4 Scout 17B se analizan cada cuadrante para detectar objetos, seres vivos y elementos con potencial sonoro. La salida en JSON incluye descripción y clasificación (fondo/primer plano) de los elementos detectados.
3. **Síntesis de audio ambiental** (Código 16): Utilizando modelos generativos de audio basados en difusión (AudioLDM), se produce una composición sonora multicanal que integra los elementos detectados. Cada componente sonoro se ajusta en términos de volumen y posición espacial relativa según su ubicación en la cuadrícula original, creando una experiencia auditiva coherente con la distribución visual de la obra.

```
1  def get_quadrants(image: Image, N:int=3):  
2      total_width, total_height = image.size  
3      w, h = total_width//N, total_height//N  
4      w_pos = [(w*i, w*(i+1)) for i in range(N)]  
5      h_pos = [(h*j, h*(j+1)) for j in range(N)]  
6      boxes = [  
7          (w_pos[i][0], h_pos[j][0], w_pos[i][1], h_pos[j][1])  
8          for j in range(N) for i in range(N)  
9      ]  
10     cropped = [ image.crop(box)  
11                 for box in boxes ]  
12     coords= [  
13         ( (i/N + (i+1)/N)/2,  
14             (j/N + (j+1)/N)/2 )  
15         for j in range(N) for i in range(N)  
16     ]  
17  
18     return cropped, coords
```

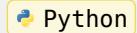
Python

Código 14: Segmentación de imagen en cuadrantes

```

1 EXTRACTOR_PROMPT = \
2     """Extract audio ambience generation keys from the following image.
3
4 Consider the following:
5 - Your output should be a list of json objects containing the keys:
6     {"is_background": bool, "object": string}
7 - Only indicate objects with sound ambience relevance.
8 - Ignore abstract elements
9 - Indicate 3 elements at most
10 - Group elements when there are more than 1 depicted
11 - You should only output ONE list
12     - Alive elements should be added with their corresponding sound
13     description (bear growling, dog barking)
14     - Non alive elements should be specified as the element itself
15
16 JSON RESULT:"""
17
18 def get_semantic_elements(image: Image, max_attempts=3):
19     b64_image = image_to_base64(image)
20     for _ in range(max_attempts):
21         try:
22             completion = client.chat.completions.create(
23                 model="meta-llama/llama-4-scout-17b-16e-instruct",
24                 messages=[{
25                     "role": "user",
26                     "content": [
27                         { "type": "text",
28                             "text": EXTRACTOR_PROMPT },
29                         { "type": "image_url",
30                             "image_url": {
31                                 "url": f"data:image/png;base64,{b64_image}" } }
32                         ],
33                     }],
34                     temperature=0,
35                 )
36             content = completion.choices[0].message.content
37             return json.loads(content)
38         except:
39             continue
40     return []

```

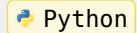


Código 15: Extracción semántica de elementos en la obra

```

1  from diffusers import AudioLDMPipeline
2  from pydub import AudioSegment
3  import numpy as np
4  import tempfile
5  import os
6  import torch
7  VOLUME_DB_BACKGROUND = -20
8  VOLUME_DB_FOREGROUND = -5
9
10 pipe = AudioLDMPipeline.from_pretrained(
11     "cvssp/audioldm"
12     torch_dtype=torch.float16
13 ).to("cuda")
14
15 def tensor_to_audiosegment(audio_tensor, sample_rate=16000):
16     samples = (audio_tensor * 32767).astype(np.int16)
17     return AudioSegment(
18         samples.tobytes(),
19         frame_rate=sample_rate,
20         sample_width=2, # 16-bit = 2 bytes
21         channels=1
22     )
23
24 def generate_ambient_sound(scene, filename):
25     final_mix = AudioSegment.silent(duration=20000)
26     for i, obj in enumerate(scene):
27         prompt = obj['object']
28         negative_prompt = ["low quality, average quality"]
29         result = pipe(
30             [prompt],
31             negative_prompt=negative_prompt,
32             num_inference_steps=30,
33             audio_length_in_s=20,
34             return_dict=True
35         )
36         audio_tensor = result audios[0].squeeze()
37         audio_segment = tensor_to_audiosegment(audio_tensor)
38         volume_db = VOLUME_DB_BACKGROUND if obj["is_background"] else
39         VOLUME_DB_FOREGROUND
40         audio_segment = audio_segment + volume_db
41         final_mix = final_mix.overlay(audio_segment)
42     final_mix.export(f"{filename}.wav", format="wav")

```



Código 16: Generación de sonido ambiental con AudioLDM

## Text to Speech

Para el módulo de **Text-to-Speech** (TTS) uno de los modelos que se utilizó fué **Kokoro**, para el cual existe una biblioteca en Python con módulos de inferencia preimplementados.

```
1 # !pip install -q kokoro>=0.9.4 soundfile
2 # !apt-get -qq -y install espeak-ng > /dev/null 2>&1
3
4 from kokoro import KPipeline
5 from IPython.display import display, Audio
6 import soundfile as sf
7 import torch
8
9 tts_pipeline = KPipeline(lang_code='e')
```

Python

Código 17: Setup Kokoro TTS

Nuestro módulo de generación de TTS considera detalles como el añadido de silencios entre saltos de línea y pausas para mantener una cadencia fluida en la narración y la concatenación de las secciones generadas por *Kokoro*.

```
1 def tts(text, filename):
2     generator = tts_pipeline(
3         text,
4         voice='em_santa',
5         speed=1, split_pattern=r'\n+'
6     )
7
8     pause_duration = 0.5
9     silence = np.zeros(int(24000 * pause_duration), dtype=np.float32)
10    all_audio = []
11
12    for i, (gs, ps, audio) in enumerate(generator):
13        all_audio.append(audio)
14        all_audio.append(silence)
15
16    final_audio = np.concatenate(all_audio)
17    sf.write(f'{filename}.wav', final_audio, 24000)
```

Python

Código 18: Generación de audio con Kokoro TTS