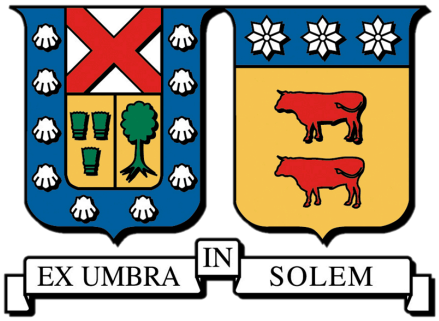
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO – CHILE



Diseño y Desarrollo de Plataforma de Apreciación Artistica 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, integrando descripciones narrativas detalladas, contexto histórico-artístico y atmósferas auditivas por medio del uso de modelos multimodales de lenguaje, sistemas de síntesis de voz y modelos de generación sonora basada en texto.

La solucion 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*

I) Lista de Figuras

II) Lista de Tablas

III) Contenidos

IV) 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 que integren narrativa, sonido y contextualización cultural.

Este proyecto desarrolla una alternativa de accesibilidad para consumo de contenido orientado a personas con discapacidad visual. En particular se plantea utilizar modelos de lenguaje multimodales, sistemas de síntesis de voz y modelos generativos de audio para una pipeline integrada de producción de piezas de audio que comprendan desde narraciones habladas hasta paisajes sonoros que representen los contenidos de las obras.

Dado que esta investigación se fundamenta en los principios de accesibilidad universal y diseño inclusivo, el acceso al contenido se implementará a través de un cliente web que considere estándares establecidos en la materia.

IV.1) 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 y descripciones auditivas significativas.

IV.2) Objetivos Particulares

- **Proveer un catálogo curado de obras de dominio público**, basado en fuentes autoritativas y verificadas.
- **Generar audios descriptivos** que contengan una representación objetiva y comprensible del contenido visual de las obras.
- **Crear audios narrativos con contenido histórico/biográfico** fundamentadas en fuentes reconocidas, que contextualizen la obra, su autor(a) y su relevancia artística.
- **Producir ambientes sonoros inmersivos** que recreen la atmósfera de las obras con piezas de audio evocativo y coherente con el contenido representado.
- **Diseñar una interfaz centrada en accesibilidad**, que habilite una experiencia de navegación cómoda para personas con discapacidad visual.

IV.3) Estructura

El presente análisis incluye:

- ***Revisión del estado del arte*** *sobre modelos generativos y estandares 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****.*

V) Marco Teórico

V.1) 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 contenido multimedial como imágenes y audio.

Estos nos permiten generar descripciones detalladas de escenas, objetos o composiciones artísticas, traduciendo elementos visuales en narrativas estructuradas. Además, su capacidad multimodal se puede utilizar para 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.

V.2) Modelos Generativos de Audio

Los modelos generativos de audio representan sistemas capaces de crear sonidos y música a partir de prompts de texto. Estos modelos, basados en arquitecturas de aprendizaje profundo, pueden sintetizar desde sonidos de animales y objetos (como el canto de pájaros o el sonido de una tormenta) hasta composiciones musicales, 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.

V.3) Modelos de Text to Speech

Los modelos de *Text to Speech* (TTS) constituyen sistemas computacionales que transforman texto escrito en habla sintética mediante técnicas de procesamiento de lenguaje natural y síntesis de voz. Los modelos de estado del arte están fundamentados en arquitecturas de aprendizaje profundo (incuyendo por lo general el uso de arquitecturas *Difussion* y *Transformers*), analizan la estructura lingüística, semántica y prosódica del input textual para generar audio con entonación, ritmo y naturalidad cercanos a la expresión humana. Su aplicación abarca desde asistentes virtuales y herramientas de accesibilidad hasta sistemas de comunicación aumentativa, permitiendo la generación de voces personalizadas, multilingües y adaptadas a contextos específicos, lo que facilita la interacción con usuarios en entornos digitales y físicos.

V.4) 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.° 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 sea inclusiva y centrada en las necesidades de los usuarios objetivo.

VI) Estado del Arte

VI.1) Modelos de Lenguaje Multimodales

Durante los últimos años los modelos múltimodales 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.

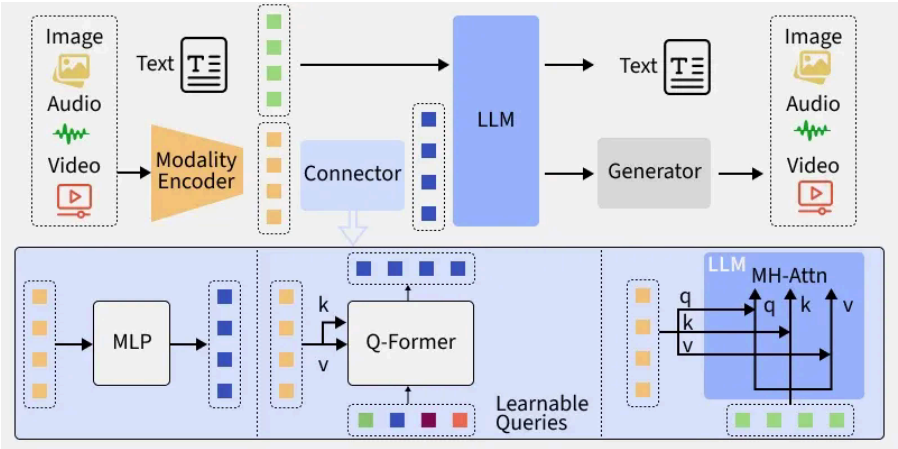


Figura 1: Multimodal LLMs

Algunos modelos destacados a la fecha son:

- **Llama v4 Maverick (Meta)**: Modelo multimodal con pesos liberados capaz de elaborar descripciones semánticas de imágenes por medio de codificación con *MetaCLIP* [1].
- **GPT-4o (OpenAI) y Gemini 1.5 (Google)**: Modelos con integración nativa de texto, imagen y audio. Sin pesos liberados.
- **DeepSeek-OCR (DeepSeek)**: Modelo multimodal de código abierto especializado en OCR y documentos visuales. Usa compresión óptica 2D para procesar contextos largos con alta precisión y bajo uso de tokens [2].

VI.2) Modelos de Texto a Voz

En los últimos años, arquitecturas basadas en RNNs como *Tacotron 2* [3], *VITS* [4] y *StyleTTS2* [5] han ganado relevancia. Estos modelos, generalmente ligeros en cantidad de parámetros, son razonables de autohostear con GPUs de grado consumidor e incluso (en algunos casos) directo en CPU a coste de mayor tiempo de inferencia. Generan voces con entonación natural aunque con expresividad limitada. Kokoro TTS [6], basado en [5], destaca por generar voces de alta calidad con 82 millones de parametros, actualmente soporta 3 voces preentrenadas para el español.

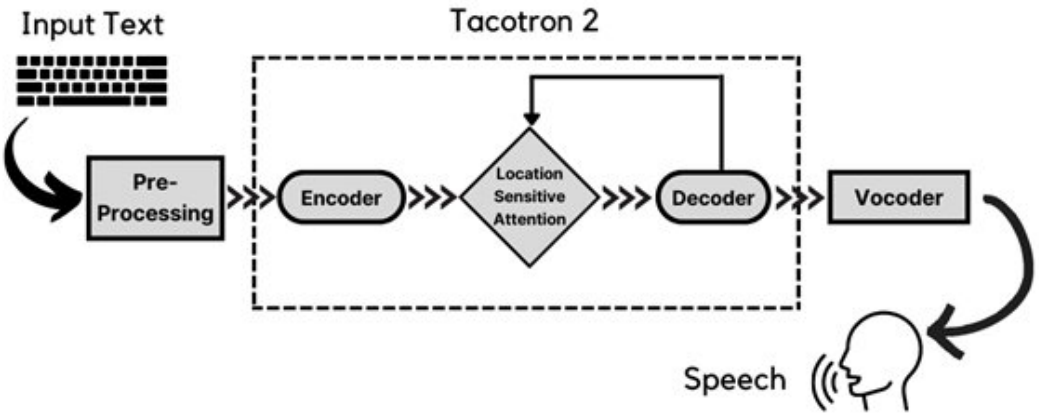


Figura 2: Tacotron 2 Arquitecture

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.

A pesar de los avances, persisten desafíos técnicos en los modelos evaluados, como la generación de expresividad controlable y la síntesis eficiente de secuencias largas. Esto último se puede mitigar dividiendo el texto en fragmentos (solución implementada en la librería de inferencia de Kokoro). En cuanto a la expresividad, VibeVoice [7] surge como alternativa capaz de generar voces altamente expresivas en secuencias largas de texto, no obstante a coste de resultados ruidosos y poco consistentes.

VI.3) 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 inicialmente considerados para la plataforma se encuentran:

- **I Hear Your True Colors** [8]: 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** [9]: 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.

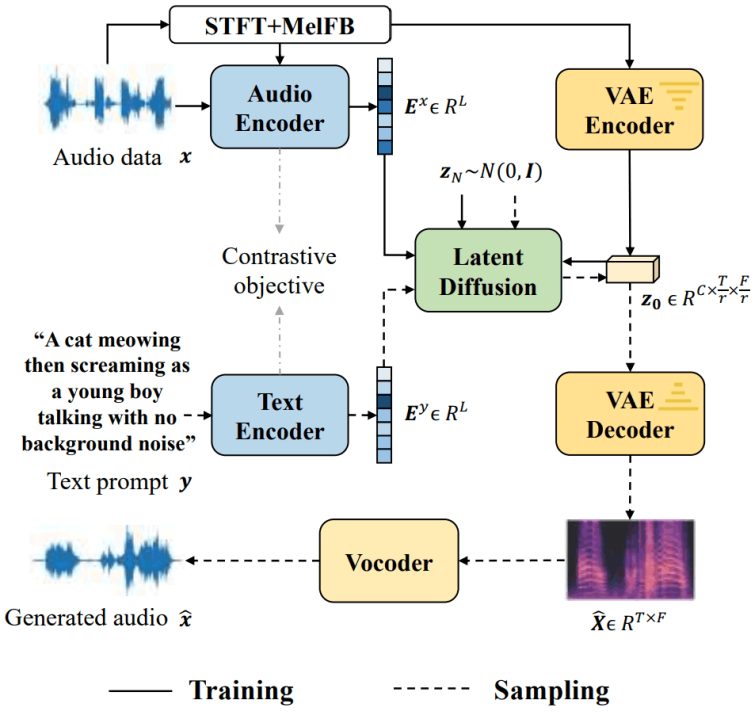


Figura 3: AudioLDM

VII) Desarrollo de la Plataforma

La arquitectura propuesta sigue un modelo cliente-servidor e incorpora una *API Gateway* como componente central. Esta 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, con especial atención a las necesidades de usuarios con discapacidad visual.

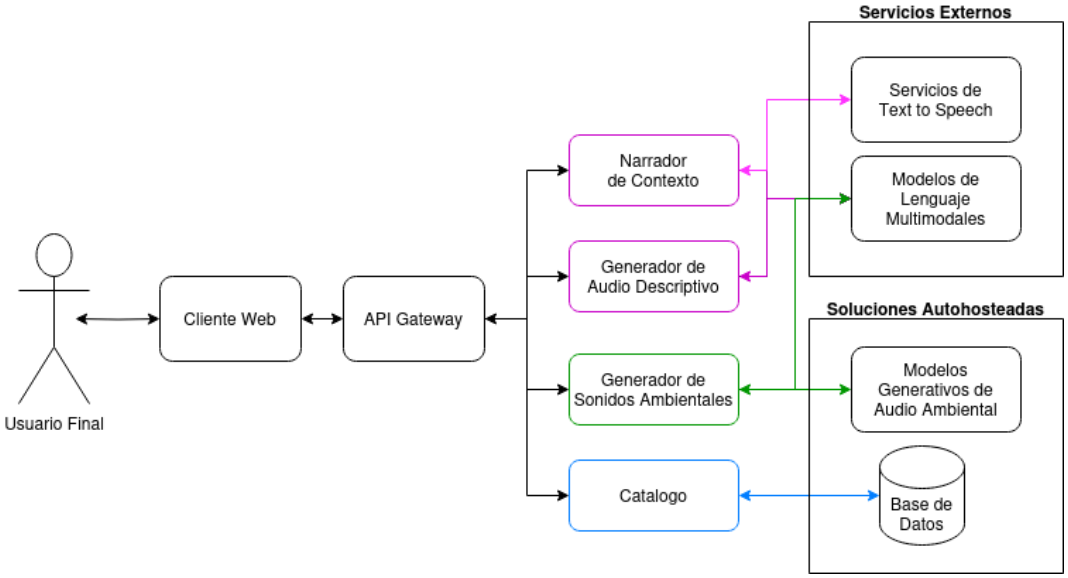


Figura 4: Arquitectura General de Sistema

VII.1) API Gateway

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

VII.1.a Generador de Audio Descriptivo

Este módulo se encarga de producir descripciones textuales detalladas de los contenidos visuales presentes en las obras. Para ello, emplea modelos de lenguaje multimodales, en particular *Llama V4 Maverick* [1] para evaluar entrada de imagenes y texto.

La imagen se evalua junto a un *prompt* diseñado para extraer información de los contenidos de la obra, además se añaden instrucciones para mantener una cadencia de redacción fluida (Código 12).

Posteriormente, las descripciones generadas son convertidas a formato de audio mediante sistemas de síntesis de voz (TTS) (Código 18) utilizando *Kokoro 82M* [6].

VII.1.b 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 13), que posteriormente se convierte en audio mediante modelos de síntesis de voz (Text to Speech) (Código 18).

VII.1.c Generador de Sonidos Ambientales

Este módulo se encarga de la generación de ambientes sonoros que representan los contenidos literales (no abstractos) de sus imagenes de entrada. La heuristica 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: Segmentación por Cuadrantes de Obra «El Aquelarre»

VII.1.d Catálogo de Obras/Imágenes

Se elaboró un conjunto de datos en formato JSON que incluye 30 obras artísticas [10], conteniendo todo el material requerido para generar el contenido de la plataforma. Este dataset incorpora, además, enlaces a los artículos de Wikipedia asociados a cada obra artística para tener referencias autoritativas en la generación de narraciones historicas sobre la producción de las mismas.

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.

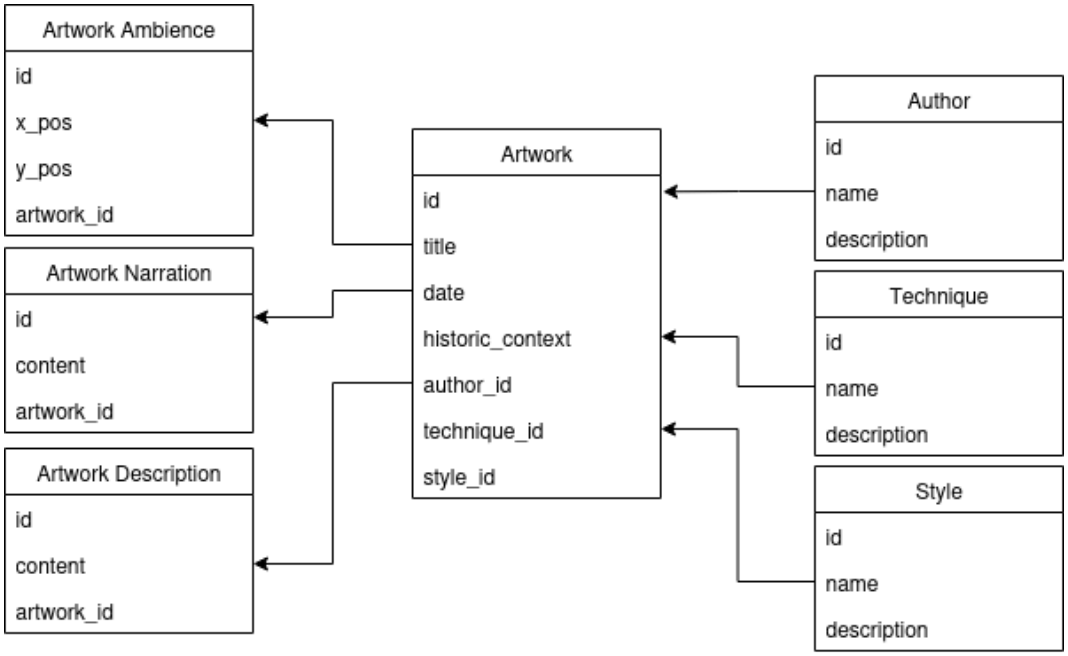


Figura 6: Modelo de Datos de la Plataforma

VII.2) 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 descriptivas 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.

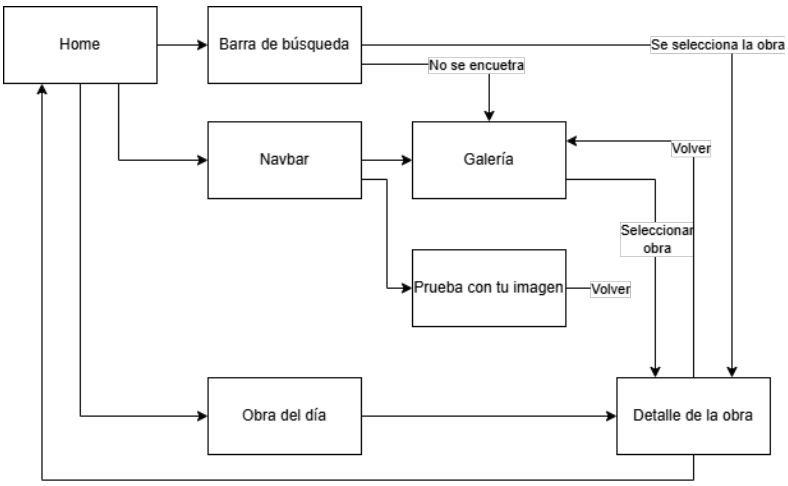


Figura 7: Flujo de Navegación Frontend

VII.3) 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 [11] 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.

VIII) Resultados

VIII.1) 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.

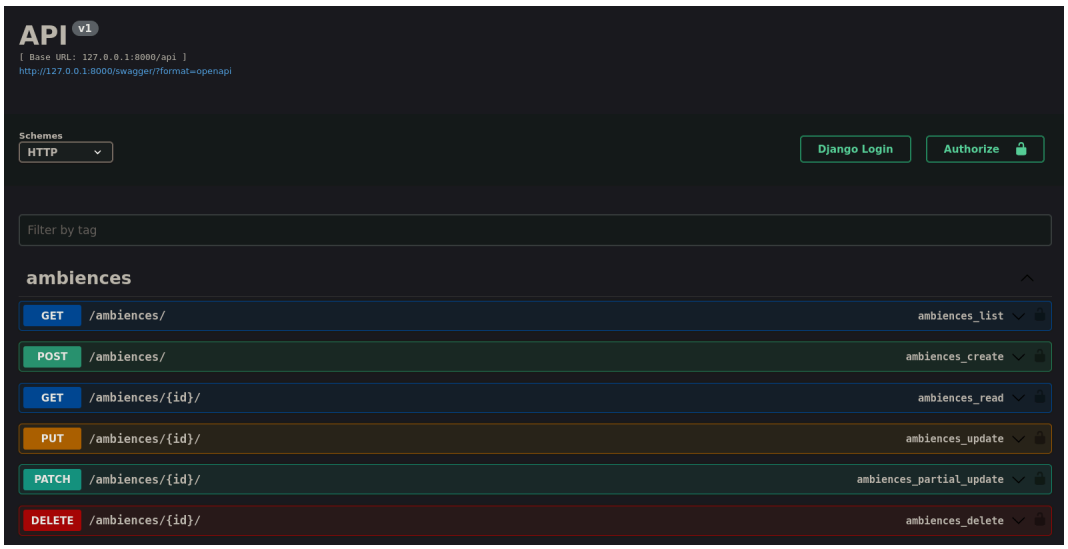


Figura 8: Documentación Swagger de la API

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 tamaño de los datos enviados (alrededor del 33%), 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.

VIII.2) 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.

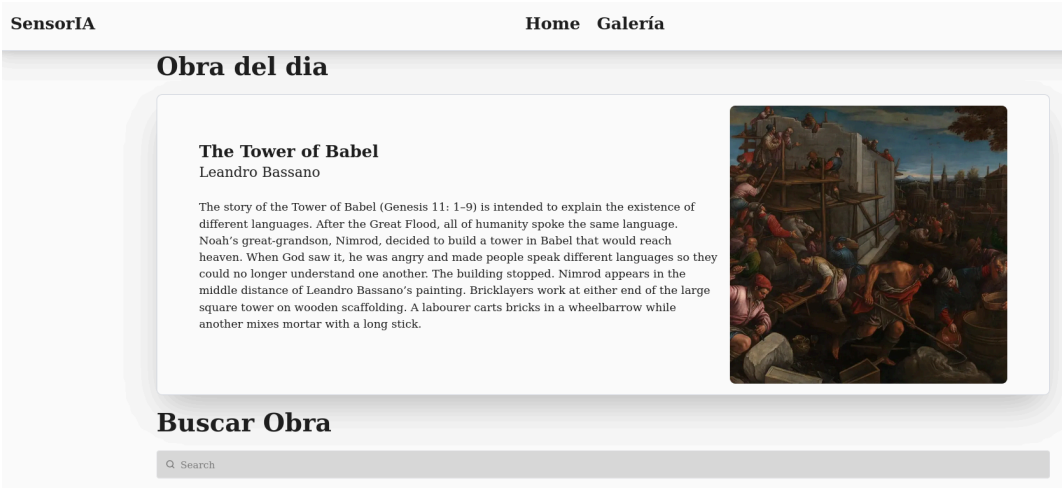


Figura 9: Frontend de la Plataforma

La funcionalidad de navegación implementa correctamente búsquedas por obra, autor, período y técnica. Sin embargo, el sistema emplea un mecanismo de coincidencia de strings que espera un match de acentuación, lo que hace rígida la utilización de la barra de busqueda.

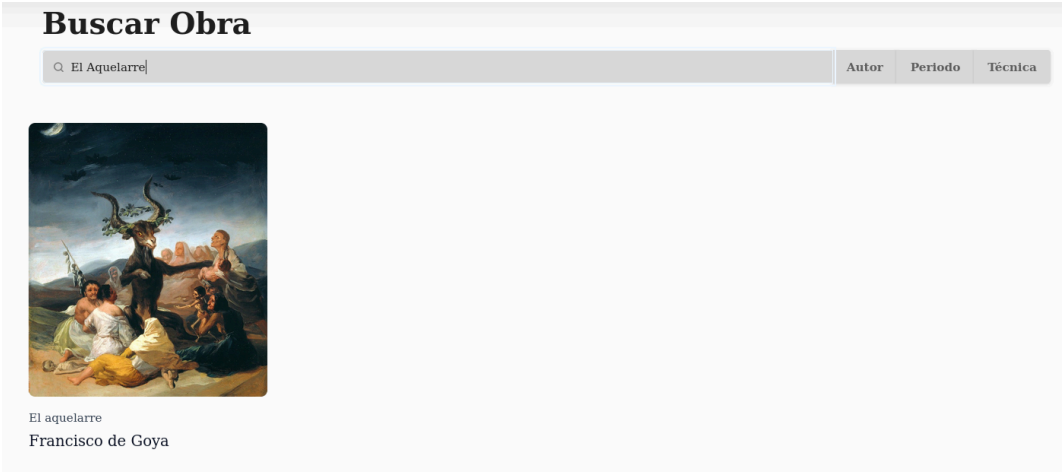


Figura 10: Funcionalidad de Búsqueda en Frontend

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

VIII.3) Módulos Generativos

A continuación se exponen los resultados obtenidos en los módulos generativos de la plataforma. Tal como se indicó en la sección anterior, estas componentes fueron implementadas en un entorno independiente [11] como parte de un desarrollo modular complementario al sistema principal.

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

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

A continuación, se evalúan los siguientes aspectos:

- 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.

VIII.3.a 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 Aquelarre de Francisco de Goya

La suficiencia de los modelos de lenguaje sin necesidad de incluir información extraída de fuentes autoritativas contrasta con lo observado en el siguiente módulo.

VIII.3.b Generación de Texto de Narraciones de Contexto

Como es señalado en Código 13, para evitar alucinaciones y producir información rica en contexto se toma por entrada un articulo de Wikipedia, para el caso ilustrado:

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 genera narraciones contextuales y evita alucinaciones vistas en prototipos anteriores de este módulo al utilizar consultas directas a modelos de lenguaje.
Un trabajo pendiente es la corrección de casos borde, en particular anglicismos y siglas.

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 aquelarre", 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 aquelarre" 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
aquelarre". 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 Aquelarre de Francisco de Goya

VIII.3.c Generador de Sonidos Ambientales

El proceso de detección de elementos acústicos en la obra presenta un desempeño adecuado, aunque se observan alucinaciones ocacionales en la identificación de componentes específicos, como puede apreciarse en el objeto detectado en segunda imagen de Tabla 1. No obstante, por lo general los elementos detectados mantienen coherencia temática con el contenido representado.

<table>
<tr><th>Segmento</th><th>Sonidos Generados</th></tr>
<tr><td>  </td><td><table><tr><td>Objeto</td><td>goat bleating</td></tr><tr><td>Fondo</td><td>tree branch creaking, people whispering</td></tr><tr><td>Audio</td><td>https://youtu.be/dqOECGiEiBI</td></tr></table></td></tr>
<tr><td>  </td><td><table><tr><td>Fondo</td><td>sand</td></tr><tr><td>Objeto</td><td>puppy sleeping</td></tr><tr><td>Audio</td><td>https://youtu.be/mT7UBvVd9m8</td></tr></table></td></tr>
</table>

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

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.

VIII.3.d Audio Generado con TTS

La implementación del módulo de Text-to-Speech (TTS) (Código 18) utilizando *Kokoro82M* [6] 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 13) o, alternativamente, incorporar un diccionario de excepciones para preservar la inteligibilidad en contextos técnicos.

Los audios generados para la descripción y la narración de «El Aquelarre» se pueden encontrar en los siguientes vínculos:

- **Descripción**: https://youtu.be/7NwKlbHQKeo
- **Narración**: https://youtu.be/P9EOz7Au4ZA

IX) Conclusiones

Los resultados obtenidos evidencian la utilidad de los modelos de lenguaje multimodales para casos de uso de accesibilidad, al demostrar que estos sistemas son capaces de interpretar de manera integrada información visual y textual y transformarla en representaciones comprensibles para usuarios con discapacidad visual. En particular, la capacidad de los modelos para generar descripciones detalladas y coherentes a partir de imágenes artísticas complejas permite sustituir, en gran medida, la percepción visual directa por una mediación auditiva rica en contenido semántico. Asimismo, la adaptación del estilo narrativo y la cadencia del texto generado facilita su posterior conversión a audio mediante sistemas de síntesis de voz, asegurando una experiencia de escucha fluida y cognitivamente accesible.

Adicionalmente, el uso de modelos multimodales reduce la dependencia de anotaciones manuales extensivas y de metadatos previamente estructurados, lo que representa una ventaja significativa en términos de escalabilidad y mantenimiento de plataformas culturales accesibles. La posibilidad de combinar análisis visual, generación de lenguaje natural y producción de audio en un mismo flujo de procesamiento permite construir soluciones integrales que responden a diversas necesidades de accesibilidad, desde la descripción literal de una obra hasta la contextualización histórica y la recreación de ambientes sonoros. En conjunto, estos resultados respaldan el potencial de los modelos de lenguaje multimodales como herramientas clave para democratizar el acceso al patrimonio cultural, ampliando la inclusión y la participación de públicos tradicionalmente excluidos de la experiencia artística visual.

Por otra parte, la utilización de modelos generativos de audio, si bien presenta limitaciones en su capacidad de generalización y estabilidad, demuestra un alto potencial para la creación de contenido apreciativo e inmersivo; no obstante, los resultados obtenidos ponen de manifiesto la necesidad de incorporar mecanismos de revisión y control de calidad con participación humana (human-in-the-loop) para garantizar la coherencia, fidelidad y utilidad final del contenido generado.

X) Bibliografía

[1] Meta, «Llama 4: Multimodal Intelligence». [En línea]. Disponible en: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[2] H. Wei, Y. Sun, y Y. Li, «DeepSeek-OCR: Contexts Optical Compression». [En línea]. Disponible en: https://arxiv.org/abs/2510.18234

[3] J. Shen *et al.*, «Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions». [En línea]. Disponible en: https://arxiv.org/abs/1712.05884

[4] J. Kim, J. Kong, y J. Son, «Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech». [En línea]. Disponible en: https://arxiv.org/abs/2106.06103

[5] Y. A. Li, C. Han, V. S. Raghavan, G. Mischler, y N. Mesgarani, «StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models». [En línea]. Disponible en: https://arxiv.org/abs/2306.07691

[6] hexgrad, «Kokoro-82M». [En línea]. Disponible en: https://huggingface.co/hexgrad/Kokoro-82M

[7] Microsoft, «VibeVoice». [En línea]. Disponible en: https://github.com/microsoft/VibeVoice

[8] R. Sheffer y Y. Adi, «I Hear Your True Colors: Image Guided Audio Generation». [En línea]. Disponible en: https://arxiv.org/abs/2211.03089

[9] H. Liu *et al.*, «AudioLDM: Text-to-Audio Generation with Latent Diffusion Models». [En línea]. Disponible en: https://arxiv.org/abs/2301.12503

[10] Alexey Mitjaew, «Artwork Sensotia Dataset». [En línea]. Disponible en: https://www.kaggle.com/datasets/alexeymitjaew/artwork-sensotia

[11] Alexey Mitjaew, «SensorIA Generation Pipeline». [En línea]. Disponible en: https://www.kaggle.com/code/alexeymitjaew/sound-generation-pipeline

XI) Anexo

XI.1) Utilidades Generales

En nuestra 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.

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

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)`).

```python
def PARSE_PROMPT(BASE_PROMPT, **kwargs):
    for key in kwargs:
        BASE_PROMPT = BASE_PROMPT.replace(
            '{' + str(key) + '}',
            str(kwargs[key])
        )
    return BASE_PROMPT
```

Código 4: Utilidad de Procesado de Prompts

XI.2) 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 (Código 5) y procesarlos en forma de *bullet points* para presentar los resultados de manera estructurada y clara (Código 7).

```python
def extract_text_from_url(url: str) -> str:
    headers = {
        "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0",
    }
    response = requests.get(url, headers=headers).raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')

    for tag in soup(['script', 'style']):
        tag.decompose()

    paragraphs = [p.get_text(strip=True) for p in soup.find_all('p')]
    return '\n\n'.join(paragraphs)
```

Código 5: Scraper Extractor de Información

```python
SOURCE_EXTRACTOR_PROMPT = """Eres un extractor de información precisa y concisa. Tu tarea es leer el texto de un artículo y devolver una lista de viñetas (•) con todos los hechos y datos relevantes, escritos en español natural y claro.

Instrucciones:
1. Identifica los hechos principales, cifras, declaraciones, fechas, nombres y conclusiones clave del texto.
2. No incluyas opiniones, lenguaje publicitario o frases irrelevantes.
3. Si el texto está en inglés u otro idioma, traduce los puntos al español correctamente.
4. Usa frases breves pero completas, cada una iniciando con un punto (•).
5. Mantén el tono informativo y objetivo.

Texto del artículo: {TEXT}
Tu respuesta:"""
```

Código 6: Prompt de Extracción de Información

```python
def extract_url_context(url):
    text = extract_text_from_url(url)[:8_000]

    completion = client.chat.completions.create(
        model="openai/gpt-oss-120b",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(SOURCE_EXTRACTOR_PROMPT,
                TEXT=text)
            }]
        }],
        temperature=0,
        max_completion_tokens=2048,
        top_p=1,
        stream=False,
        stop=None,
    )
    content = completion.choices[0].message.content
    return content
```

Código 7: Script de Extracción de Información

XI.3) 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 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 con una base mas robusta de contenido.
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.

```python
TECHNIQUE_PROMPT = \
"""Explícame en detalle una técnica de pintura o dibujo.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Técnica: {TECHNIQUE}
Explicación:"""

def get_technique(technique):
    completion = client.chat.completions.create(
        #model="llama-3.1-8b-instant",
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(TECHNIQUE_PROMPT, TECHNIQUE=technique)
            }]
        }],
        temperature=0,
        max_completion_tokens=1024,
        top_p=1,
        stream=False,
        stop=None,
    )
    content = completion.choices[0].message.content
    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).

```python
PERIOD_PROMPT = \
"""Explícame en detalle un estilo pictórico en particular.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Estilo pictórico: {PERIOD}
Explicación:"""

def get_period(period):
    completion = client.chat.completions.create(
        #model="llama-3.1-8b-instant",
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(PERIOD_PROMPT, PERIOD=period)
            }]
        }],
        temperature=0,
        max_completion_tokens=1024,
        top_p=1,
        stream=False,
        stop=None,
    )
    return completion.choices[0].message.content
```

Código 9: Generador de Texto de Período

En la generación de la biografía se utilizó un script de extracción de información (Código 7) para procesar un breve texto biográfico del autor.

```python
AUTHOR_BIO_PROMPT = \
"""Eres un experto en redacción biográfica y narrativa.
Tu tarea es leer el siguiente texto y redactar una biografía breve y fluida del autor o artista mencionado, adecuada para acompañar una ficha de obra de arte.
Instrucciones:
1. Resume los aspectos esenciales de la vida y obra del autor: formación, estilo, periodo histórico, aportes y relevancia artística.
2. Mantén un tono natural, informativo y narrativo, sin usar listas ni formato enciclopédico.
3. Evita redundancias, tecnicismos innecesarios y frases genéricas.
4. No incluyas fechas o datos no presentes en el texto fuente.
5. No menciones el proceso de redacción ni expresiones como “esta biografía trata sobre”.
6. Si el texto está en otro idioma, tradúcelo al español de manera natural.
7. Extensión máxima: dos párrafos.
8. Evita completamente el uso de caracteres especiales

Texto: {TEXT}
Tu respuesta:"""

def get_bio(url):
    text = extract_url_context(url)
    completion = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(AUTHOR_BIO_PROMPT, TEXT=text)
            }]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

,

Código 10: Author Bio Generation

XI.4) Utilidades Generales de Procesamiento de Imágenes

A continuación, se definen utilidades de preprocesamiento general de imágenes que se aplican antes de realizar llamadas a proveedores de servicios de inferencia o de emplearlas 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.

```python
def resize_if_oversized(image, max_dim=1024):
    width, height = image.size
    if max(width, height) <= max_dim:
        return image  # nothing to do

    # scale preserving proportions
    scale = max_dim / max(width, height)
    new_size = (int(width * scale), int(height * scale))
    return image.resize(new_size, Image.Resampling.LANCZOS)

def download_img(url, max_dim: int = 1024):
    headers = { "User-Agent": "Chrome/51.0.2704.103 Safari/537.36" }
    r = requests.get(url, headers=headers)
    im = Image.open(BytesIO(r.content))
    im = resize_if_oversized(im, max_dim)
    return im

def image_to_base64(image: Image.Image, format="PNG") -> str:
    """Convert PIL Image to base64 string."""
    buffered = BytesIO()
    image.save(buffered, format=format)
    return base64.b64encode(buffered.getvalue()).decode("utf-8")
```

Código 11: Utilidades de Procesamiento de Imágenes

XI.5) 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 imagenes 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 del contenido representado utilizando una cadencia adecuada para su narración por voz.

```python
DESCRIPTION_PROMPT = \
"""Describe los elementos retratados en la imagen.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Elementos retratados: """

def get_description(url):
    image = download_img(url)
    b64_image = image_to_base64(image)
    completion = client.chat.completions.create(
        model="meta-llama/llama-4-scout-17b-16e-instruct",
        messages=[{
            "role": "user",
            "content": [
                { "type": "text",
                  "text": DESCRIPTION_PROMPT },
                { "type": "image_url",
                  "image_url": {
                    "url": f"{b64_image}"
                  } }
            ]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

Código 12: Generación de texto para audios dscriptivos

XI.6) 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.

```python
SOURCE_SUMMARY_PROMPT = \
"""Eres un experto en redacción narrativa. Tu tarea es leer el siguiente texto y escribir un resumen breve (máximo dos párrafos) en un estilo natural, fluido y adecuado para narración oral o escrita.

Instrucciones:
1. Mantén un tono narrativo, como si estuvieras contando los hechos de manera clara y atractiva.
2. Sé conciso: evita repeticiones, rodeos o frases innecesarias.
3. No incluyas títulos, subtítulos ni listas.
4. No uses expresiones como “en resumen”, “este artículo trata sobre”, ni menciones al proceso de resumen.
5. Si el texto está en otro idioma, tradúcelo al español con naturalidad.
6. Enfócate en los hechos principales y el contexto historico de la producción de la obra.

Texto: {TEXT}
Tu respuesta:"""

def get_narration(url):
    text = extract_url_context(url)
    completion = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(SOURCE_SUMMARY_PROMPT, TEXT=text)
            }]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

Código 13: Generación de texto para narraciones

XI.7) Generación de Sonidos Ambientales

La eurística 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 de acuerdo a si es categorizado como un objeto de fondo.

```python
def get_quadrants(image: Image, N:int=3):
    total_width, total_height = image.size
    w, h = total_width//N, total_height//N
    w_pos = [(w*i, w*(i+1)) for i in range(N)]
    h_pos = [(h*i, h*(i+1)) for i in range(N)]
    boxes = [
        (w_pos[i][0], h_pos[j][0], w_pos[i][1], h_pos[j][1])
        for j in range(N) for i in range(N)
    ]
    cropped = [ image.crop(box)
                for box in boxes ]
    coords= [
      ( (i/N + (i+1)/N)/2,
        (j/N + (j+1)/N)/2 )
      for j in range(N) for i in range(N)
    ]

    return cropped, coords
```

Código 14: Segmentación de imagen en cuadrantes

```python
EXTRACTOR_PROMPT = \
"""Extract audio ambience generation keys from the following image.

Consider the following:
- Your output should be a list of json objects containing the keys:
    {"is_background": bool, "object": string}
- Only indicate objects with sound ambience relevance.
- Ignore abstract elements
- Indicate 3 elements at most
- Group elements when there are more than 1 depicted
- You should only output ONE list
- Alive elements should be added with their corresponding sound description (bear growling, dog barking)
- Non alive elements should be specified as the element itself
- Output just the required JSON results

JSON RESULT:"""

def get_semantic_elements(image: Image, max_attempts=3):
    b64_image = image_to_base64(image)
    for _ in range(max_attempts):
        try:
            completion = client.chat.completions.create(
                model="meta-llama/llama-4-scout-17b-16e-instruct",
                messages=[{
                "role": "user",
                "content": [
                { "type": "text",
                    "text": EXTRACTOR_PROMPT },
                { "type": "image_url",
                    "image_url": {
                    "url": f"{b64_image}" } }
                ]
                }],
                temperature=0,
            )
            content = completion.choices[0].message.content
            return json.loads(content)
        except:
            continue
    return []
```

Código 15: Extracción semántica de elementos en la obra

```python
from diffusers import AudioLDMPipeline
from pydub import AudioSegment
import numpy as np
import tempfile
import os
import torch
VOLUME_DB_BACKGROUND = -20
VOLUME_DB_FOREGROUND = -5

pipe = AudioLDMPipeline.from_pretrained(
    "cvssp/audioldm"
    torch_dtype=torch.float16
).to("cuda")

def tensor_to_audiosegment(audio_tensor, sample_rate=16000):
    samples = (audio_tensor * 32767).astype(np.int16)
    return AudioSegment(
        samples.tobytes(),
        frame_rate=sample_rate,
        sample_width=2,  # 16-bit = 2 bytes
        channels=1
    )

def generate_ambient_sound(scene, filename):
    final_mix = AudioSegment.silent(duration=20000)
    for i, obj in enumerate(scene):
        prompt = obj['object']
        negative_prompt = ["low quality, average quality"]
        result = pipe(
            [prompt],
            negative_prompt=negative_prompt,
            num_inference_steps=30,
            audio_length_in_s=20,
            return_dict=True
        )
        audio_tensor = result.audios[0].squeeze()
        audio_segment = tensor_to_audiosegment(audio_tensor)
        volume_db = VOLUME_DB_BACKGROUND if obj["is_background"] else VOLUME_DB_FOREGROUND
        audio_segment = audio_segment + volume_db
        final_mix = final_mix.overlay(audio_segment)

    final_mix.export(f"{filename}.wav", format="wav")
```

Código 16: Generación de sonido ambiental con AudioLDM

XI.8) Text to Speech

Para el módulo de **Text-to-Speech** (TTS) se utilizó fué **Kokoro 82M** [6], para el cual existe una biblioteca en Python con módulos de inferencia preimplementados.

```python
# !pip install -q kokoro>=0.9.4 soundfile
# !apt-get -qq -y install espeak-ng > /dev/null 2>&1

from kokoro import KPipeline
from IPython.display import display, Audio
import soundfile as sf
import torch

tts_pipeline = KPipeline(lang_code='e')
```

Código 17: Setup Kokoro TTS

Nuestro función de generación de TTS añade breves silencios entre parrafos para mantener una cadencia fluida en la narración.

```python
def tts(text, filename):
    generator = tts_pipeline(
        text,
        voice='em_santa',
        speed=1, split_pattern=r'\n+'
    )

    pause_duration = 0.5
    silence = np.zeros(int(24000 * pause_duration), dtype=np.float32)
    all_audio = []

    for i, (gs, ps, audio) in enumerate(generator):
        all_audio.append(audio)
        all_audio.append(silence)

    final_audio = np.concatenate(all_audio)
    sf.write(f"{filename}.wav", final_audio, 24000)
```

Código 18: Generación de audio con Kokoro TTS