ESCUELA DE INGENIERÍA DE FUENLABRADA

INGENIERÍA EN TELEMÁTICA

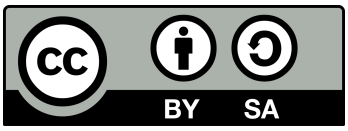
TRABAJO FIN DE GRADO

GENERACIÓN DE ESCENAS EN REALIDAD EXTENDIDA MEDIANTE VOZ

Autor : Rodrigo Javier Alonso-Carriazo Rodríguez

Tutor : Jesús María González Barahona

Curso académico 2025/2026



Dedicado a

mi familia

Agradecimientos

Quiero expresar mi agradecimiento a todas las personas que, de una forma u otra, han formado parte de este camino y han contribuido a que hoy pueda cerrar esta etapa.

A mi familia, por ser siempre mi apoyo incondicional, por los valores que me han transmitido y por estar presentes en cada momento importante.

A mis padres, por su esfuerzo, confianza y por enseñarme a perseverar incluso cuando las cosas no son fáciles.

A mis hermanos, por su cercanía, sus consejos y por ser un ejemplo constante en el que fijarme.

A mi pareja, por su paciencia, comprensión y por acompañarme día a día.

A mis amigos y compañeros, por compartir este recorrido, por el apoyo mutuo y por todos los momentos que han hecho este proceso más llevadero.

A los profesores y tutor del proyecto, por su orientación, dedicación y por haber contribuido a mi formación tanto académica como personal.

Y, finalmente, a mí mismo, por el esfuerzo, la constancia y la capacidad de seguir adelante hasta alcanzar este objetivo.

Resumen

En este este Trabajo Fin de Grado se presenta una investigación sobre la integración de tecnologías web e inteligencia artificial para la creación de escenas en entornos de realidad extendida mediante comandos de voz en lenguaje natural. Los resultados obtenidos demuestran la viabilidad de desarrollar interfaces que simplifican la interacción con entornos tridimensionales, ofreciendo una alternativa más intuitiva frente a los métodos tradicionales basados en interfaces gráficas convencionales.

La solución propuesta combina tecnologías de realidad extendida accesibles desde el navegador con modelos de inteligencia artificial generativa capaces de interpretar instrucciones en lenguaje natural, transformándolas en configuraciones estructuradas que permiten la generación automática de escenas. Este enfoque permite reducir la complejidad en la creación de entornos virtuales, facilitando su uso por parte de usuarios sin conocimientos técnicos avanzados.

El sistema incorpora, además, mecanismos de interacción que permiten no solo la creación de escenas mediante voz, sino también su modificación y control, proporcionando una experiencia más dinámica y flexible. La integración de estas tecnologías da lugar a una arquitectura modular y extensible, capaz de adaptarse a distintos escenarios y necesidades.

Este trabajo contribuye al desarrollo de nuevas formas de interacción entre los humanos y las máquinas en el ámbito de la realidad extendida, estableciendo una base para futuras aplicaciones en ámbitos como la educación, la visualización de información o el diseño de entornos virtuales.

Summary

This thesis presents research into the integration of web technologies and artificial intelligence for the creation of scenes in extended reality environments using natural language voice commands. The results obtained demonstrate the feasibility of developing interfaces that simplify interaction with three-dimensional environments, offering a more intuitive alternative to traditional methods based on conventional graphical interfaces. The proposed solution combines browser-accessible extended reality technologies with generative artificial intelligence models capable of interpreting instructions in natural language, transforming them into structured configurations that enable the automatic generation of scenes. This approach reduces the complexity of creating virtual environments, making them easier to use for users without advanced technical knowledge. The system also incorporates interaction mechanisms that allow not only the creation of scenes via voice, but also their modification and control, providing a more dynamic and flexible experience. The integration of these technologies results in a modular and extensible architecture, capable of adapting to different scenarios and needs. This work contributes to the development of new forms of interaction between humans and machines in the field of extended reality, laying the groundwork for future applications in areas such as education, visual information and virtual environment design.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En los últimos años, la evolución de la inteligencia artificial y de las tecnologías inmersivas ha impulsado nuevas formas de interacción entre los usuarios y los entornos digitales. En particular, el desarrollo de modelos de lenguaje y sistemas de reconocimiento de voz capaces de interpretar instrucciones en lenguaje natural y transformarlas en acciones comprensibles para el sistema.

Paralelamente, la realidad extendida accesible desde el navegador ha experimentado un crecimiento significativo gracias a la consolidación de tecnologías web estandarizadas. Este avance ha facilitado la creación y visualización de entornos tridimensionales sin necesidad de software especializado, ampliando el acceso a experiencias inmersivas a un mayor número de usuarios.

En este contexto, la generación de escenas 3D sigue siendo, en muchos casos, un proceso complejo que requiere conocimientos técnicos y el uso de herramientas específicas. Esto plantea la necesidad de explorar nuevas formas de interacción que permitan simplificar dicho proceso y hacerlo más accesible.

Este Trabajo de Fin de Grado se sitúa en la intersección de estos ámbitos, proponiendo un sistema que permite la creación y modificación de escenas en entornos de realidad extendida mediante comandos de voz en lenguaje natural. A través de la integración de tecnologías web, reconocimiento de voz e inteligencia artificial generativa, se plantea una alternativa a los métodos tradicionales de interacción basados en interfaces gráficas o dispositivos físicos.

El sistema desarrollado permite al usuario describir escenas mediante voz, interpretando dichas instrucciones y transformándolas en entornos tridimensionales generados dinámicamente en el navegador. Además, el usuario puede modificar progresivamente la escena mediante

nuevas órdenes, estableciendo una interacción continua y flexible con el sistema.

Este enfoque no solo simplifica la creación de contenidos 3D, sino que también abre nuevas posibilidades en el diseño de interfaces más intuitivas y accesibles, contribuyendo al desarrollo de nuevas formas de interacción en el ámbito de la realidad extendida.

1.1. Objetivo general

El objetivo de este Trabajo Fin de Grado es investigar nuevas formas de interacción con entornos tridimensionales mediante el uso de lenguaje natural, con especial énfasis en la voz, dentro del contexto de la realidad extendida.

Para ello, se lleva a cabo un proceso de experimentación con distintas herramientas y tecnologías, analizando cómo su combinación puede facilitar la creación y modificación de escenas virtuales de forma más intuitiva.

A lo largo del desarrollo, se explora el alcance real de la interacción por voz en entornos 3D, evaluando tanto sus posibilidades como sus limitaciones.

Como resultado, se busca obtener una solución flexible que permita construir y modificar escenas de realidad virtual de manera dinámica.

1.2. Objetivos específicos

Para alcanzar el objetivo general del proyecto, se han definido los siguientes objetivos específicos:

- Experimentar con distintas tecnologías web orientadas al desarrollo de experiencias inmersivas, analizando sus capacidades y limitaciones en el contexto de la realidad extendida.

- Diseñar y desarrollar un conjunto de herramientas que permita la creación de escenas en realidad extendida

- Analizar e integrar distintas soluciones de reconocimiento de voz en entornos web, explorando diferentes enfoques de implementación —como el uso de capacidades nativas del

navegador o el consumo de APIs externas— y evaluando su funcionamiento en distintos dispositivos y su idoneidad para la interacción natural con el usuario.

- Investigar el uso de APIs externas a través de encaminadores.
- Explorar el uso de técnicas de inteligencia artificial generativa para interpretar descripciones en lenguaje natural y transformarlas en representaciones estructuradas que permitan la generación automática de escenas.
- Diseñar una arquitectura modular basada en componentes que favorezca la reutilización, flexibilidad, mantenibilidad y escalabilidad de la aplicación.
- Implementar un flujo completo que abarque desde la entrada del usuario hasta la visualización de la escena.
- Desarrollar mecanismos de interacción que permitan no solo crear escenas, sino también modificarlas y ejecutar acciones mediante comandos de voz.
- Adaptar la aplicación para su funcionamiento tanto en navegadores de escritorio como en dispositivos de realidad virtual.
- Incorporar un flujo de almacenamiento y despliegue automático de los escenarios generados.

1.3. Estructura de la memoria

La memoria se ha organizado en seis capítulos principales, además de la bibliografía final.

- **Capítulo 1: Introducción.** En este capítulo se presenta la introducción al trabajo, incluyendo el objetivo general, los objetivos específicos y la estructura seguida en la memoria.
- **Capítulo 2: Tecnologías utilizadas y trabajos relacionados.** Se describen las tecnologías utilizadas en el proyecto, incluyendo inteligencia artificial generativa, modelos de lenguaje, tecnologías de reconocimiento de voz, tecnologías web, APIs del navegador para XR, herramientas de renderizado 3D y servicios externos utilizados durante el desarrollo. Asimismo, se revisan algunos trabajos y aplicaciones relacionadas que sirven de referencia para contextualizar la propuesta realizada.

- **Capítulo 3: Desarrollo del proyecto.** El proyecto se desarrolla siguiendo una organización por sprints. En cada uno de ellos se describen los objetivos planteados, las tareas realizadas, los resultados obtenidos y las lecciones aprendidas, mostrando la evolución progresiva del sistema hasta alcanzar una versión funcional completa.

- **Capítulo 4: Descripción del resultado.** Se presenta la funcionalidad del sistema desarrollado, junto con su estructura y la descripción e implementación de los componentes. Además, se mencionan los entornos de ejecución posible del sistema junto con una par de ejemplos.

- **Capítulo 5: Experimentos y validación.** Se detallan los objetivos del experimento, su diseño, los usuarios participantes y los resultados obtenidos, tanto desde un punto de vista cuantitativo como cualitativo, incluyendo la precisión del reconocimiento de voz, tiempos por tarea, grado de satisfacción de los usuarios, opiniones de los usuarios y problemas detectados.

- **Capítulo 6: Conclusiones.** Recoge las conclusiones del trabajo. En este apartado se analiza el grado de consecución de los objetivos planteados, el esfuerzo realizado, las competencias aplicadas, las principales lecciones aprendidas y las posibles líneas de trabajo futuro.

Finalmente, se incluye la **bibliografía**, donde se recopilan las referencias utilizadas a lo largo del desarrollo del proyecto y de la redacción de la memoria.

Capítulo 2

Tecnologías utilizadas y trabajos relacionados

En este capítulo se presentan las principales tecnologías empleadas en el desarrollo del proyecto, así como una revisión de trabajos relacionados en este ámbito.

En primer lugar, se abordan las tecnologías vinculadas a la inteligencia artificial generativa y al procesamiento del lenguaje natural, que permiten interpretar las instrucciones proporcionadas por el usuario y traducirlas en la generación de escenas.

A continuación, se describen las soluciones de reconocimiento de voz utilizadas en el proyecto, fundamentales para permitir la interacción del usuario mediante lenguaje hablado.

Posteriormente, se presentan las tecnologías web sobre las que se sustenta la aplicación, así como aquellas orientadas a la creación y visualización de entornos de realidad extendida en el navegador.

Asimismo, se incluyen las herramientas y mecanismos empleados para la comunicación con servicios externos, necesarios para integrar distintas funcionalidades del sistema.

Por último, se describen los dispositivos de realidad virtual, las herramientas de desarrollo utilizadas y se analizan algunos prototipos relacionados con el proyecto.

2.1. IA Generativa

La inteligencia artificial (IA) es el campo de la informática que se dedica al desarrollo de algoritmos y sistemas que pueden realizar tareas que normalmente requieren inteligencia humana,

como el aprendizaje, la comprensión del lenguaje natural, la toma de decisiones y la resolución de problemas. La IA ha crecido de manera exponencial en los últimos años y se ha aplicado en multitud de sectores como la salud, la banca, telecomunicaciones, fabricación y comercio, aumentando enormemente su productividad. Hay distintos tipos de inteligencia artifical como, por ejemplo, la inteligencia artifical evolutiva, inteligencia artifical simbólica, inteligencia artifical híbrida, entre otros. En este proyecto, nos vamos a centrar en la inteligencia artifical generativa. La **inteligencia artificial generativa** es un subcampo de la inteligencia artifical que se centra en la creación de sistemas capaces de generar datos, contenido o información de manera autónoma y creativa. Estos sistemas utilizan modelos y algoritmos que pueden aprender patrones y estructuras a partir de conjuntos de datos existentes y luego generar nuevos datos que se asemejan a los datos de entrenamiento

2.1.1. Modelos de lenguaje (LLMs)

La inteligencia artificial generativa y los grandes modelos de lenguaje (LLMs, *Large Language Models*) pueden cambiar la forma en que consultamos, procesamos y producimos información. Pero presentan desafíos técnicos y éticos, tales como inconsistencias, sesgos y falta de transparencia. Por ello, conviene tener conciencia de las implicaciones de esta tecnología y de nuestras responsabilidades como usuarios, así como de los principios y las buenas prácticas que deben guiar su uso. Los grandes modelos de lenguaje (LLMs) son sistemas de inteligencia artifical especializados en procesar y generar lenguaje natural, que es el que usamos los humanos para comunicarnos. Se basan en redes neuronales artificiales que están compuestas por unidades interconectadas llamadas ”neuronas artificiales” que trabajan en conjunto para procesar información y realizar tareas específicas, inspiradas en el funcionamiento del cerebro humano, que se entrenan con una ingente cantidad de texto procedente de fuentes como libros, periódicos o páginas web, y pueden producir textos coherentes y fluidos sobre cualquier tema. La arquitectura de redes neuronales está basada, usualmente, en la estructura Transformer[1] [3] en la cual la entrada pasará por una serie de Encoders que se encadenan uno tras otro y luego envían su salida a otra serie de Decoders hasta emitir la salida final.

[1] `chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://secoir.org/wp-content/uploads/2025/05/10.2-Monografia-SECOIR-2025-V1.pdf`

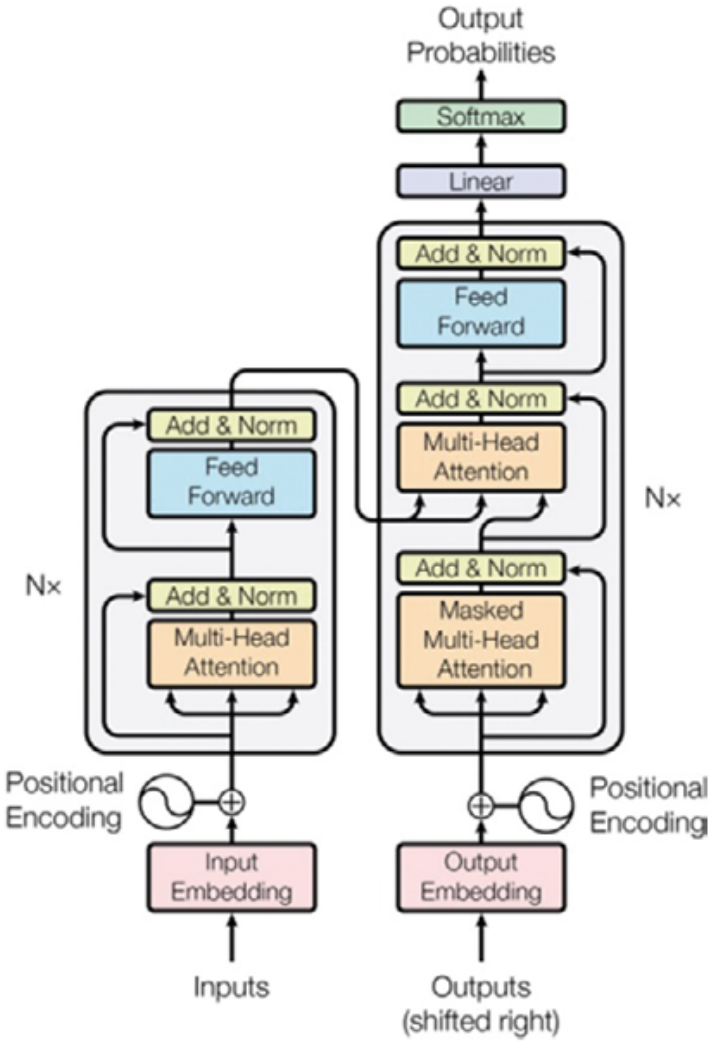


Figura 2.1: Arquitectura Transformer [3]

Los LLM consiguen capturar matices del lenguaje humano como la gramática, la semántica e incluso el contexto cultural, con una alta precisión. Además, el uso de prompts adecuados hace que los usuarios pueden comunicarse con los sistemas y obtener respuestas o resultados específicos según lo que hayan solicitado en su entrada. Un prompt es una instrucción o conjunto de indicaciones que se proporciona a un sistema de inteligencia artificial, especialmente a modelos de lenguaje, con el objetivo de obtener una respuesta o realizar una tarea concreta. Generalmente, se trata de un fragmento de texto en lenguaje natural que define qué se espera del sistema, pudiendo incluir información, contexto o restricciones para guiar su comportamiento.

Los modelos de lenguaje han progresado mucho gracias al desarrollo de arquitecturas complejas y potentes, tales como GPT, de OpenAI; LLaMA, de Meta; PaLM, de Google; o Claude,

de Anthropic. Estas tecnologías ofrecen hoy una interfaz sencilla que se puede interrogar usando nuestro propio lenguaje natural, lo que las hace más accesibles para el público general. Los chatbots o asistentes de IA, como Bard, ChatGPT o Claude, utilizan técnicas de procesamiento de lenguaje natural para ofrecer respuestas pertinentes, en tiempo real, a casi cualquier consulta, simulando una conversación humana. Pero estas posibilidades también nos enfrentan a nuevos retos y responsabilidades. El procesacimento de lenguaje natural (PLN) es una rama de la inteligencia artificial que se centra en la interacción entre los ordenadores y el lenguaje humano. Su objetivo es permitir que las máquinas comprendan, interpreten, manipulen y respondan al lenguaje humano de una manera útil y significativa. A través del PLN, las máquinas pueden reaalizar tareas como la traducción automática, el reconocimiento de voz, la generación de texto, la extracción de la información, la clasificación de documentos, y la respuesta a preguntas, facilitando así la interacción natural y fluida entre humanos y computadoras. Sin embargo, interesa tener presente que los modelos de lenguaje no son infalibles. Pueden arrojar respuestas inexactas o erróneas, aunque suenen convincentes. Por eso, los profesionales de la información tienen que estar al tanto de las posibles imperfecciones de estos sistemas, han de ayudar a los usuarios a identificarlas y corregirlas, y deben concienciarlos sobre la necesidad de verificar el contenido artificial antes de confiar en él.

2.1.2. Tecnologías de encaminamiento

Las tecnologías de encaminamiento en el contexto de modelos de lenguaje permiten seleccionar y gestionar el uso de distintos modelos de inteligencia artificial en función de la tarea a realizar. Estas herramientas actúan como una capa intermedia que decide qué modelo es el más adecuado para procesar una determinada petición.

El uso de este tipo de tecnologías aporta flexibilidad, escalabilidad y eficiencia al sistema dado que permite adaptar el comportamiento de la aplicación en función de factores como el tipo de instrucción, el coste, la latencia o el rendimiento del modelo. Además, facilita la integración de múltiples modelos de lenguaje grande (LLMs) en arquitecturas empresariales.

Estas tecnologías ofrecen un acceso centralizado con multitud de modelos mediante una API estandarizada, facilitando el uso intercambiable de LLMs sin necesidad de modificar el código existente. Las tecnologías de encaminamiento surgen del uso simultáneo de LLMs que conllevaban múltiples APIs, costes distribuidos, falta de resiliencia y complejidad en los flujos

de integración. Por ello, surge la necesidad de una plataforma que ofrece un único punto de acceso a múltiples LLMs mediante una API estandarizada y rutas inteligentes como, por ejemplo, OpenRouter[2] [22] y Groq[3] [7]. Estas plataformas permiten asignar modelos distintos a cada agente según su rol. Además, los agentes pueden cambiar de modelo según el contexto, coste o urgencia. Mejora la resiliencia y ofrece una forma práctica de combinar modelos propietarios y open source.

2.2. Reconocimiento de voz

El reconocimiento de voz tiene sus orígenes en la década de 1950. En 1952, los Bell Laboratories desarrollaron el sistema *Audrey*, capaz de reconocer dígitos pronunciados por una única voz. Años más tarde, IBM presentó el sistema Shoebox, que podía comprender un conjunto limitado de palabras en inglés.

Desde entonces, esta tecnología ha experimentado una evolución constante. No obstante, los avances más significativos se han producido en los últimos años, impulsados por el incremento de la capacidad de cálculo y el desarrollo de técnicas basadas en inteligencia artificial. La disponibilidad de grandes volúmenes de datos de voz, junto con algoritmos más avanzados, ha permitido mejorar notablemente la precisión de estos sistemas.

Como resultado, el reconocimiento de voz se ha convertido en una forma de interacción cada vez más habitual, presente en asistentes virtuales, dispositivos móviles y diversos sistemas interactivos.

En este proyecto se exploran distintas soluciones de reconocimiento de voz accesibles desde el entorno web, con el objetivo de analizar sus capacidades y su integración dentro de la aplicación. Entre ellas se incluyen tanto alternativas basadas en el navegador, como la *Web Speech API*, como modelos más avanzados que pueden ejecutarse de forma local o mediante servicios externos, como *Vosk* o *Whisper*.

[2] `https://openrouter.ai/docs`
[3] `https://console.groq.com/docs/overview`

2.2.1. Vosk-browser

Vosk-browser[4] [8] es una librería de reconocimiento de voz que permite realizar transcripciones directamente en el navegador de forma local sin necesidad de conexión a servicios externos. Está basada en el motor Vosk[5] [2], el cual utiliza modelos de reconocimiento optimizados para funcionar en dispositivos con recursos limitados. Los modelos Vosk son pequeños (50 MB), pero proporcionan transcripción continua de un amplio vocabulario, respuesta de latencia cero con API de transmisión, vocabulario reconfigurable e identificación del hablante. Permite el reconocimiento de voz para más de 20 idiomas y dialectos.

Desde el punto de vista técnico, Vosk emplea modelos de reconocimiento de voz basados en redes neuronales, concretamente modelos acústicos entrenados mediante técnicas de aprendizaje profundo (deep learning), un enfoque de la inteligencia artificial que utiliza redes neuronales con múltiples capas para aprender patrones complejos a partir de grandes volúmenes de datos. En este contexto, es habitual el uso de redes neuronales recurrentes (RNN), un tipo de arquitectura especialmente adecuada para el procesamiento de secuencias, como el audio, ya que permite tener en cuenta la información temporal a lo largo del tiempo.

Estos modelos permiten mapear la señal de audio a secuencias de texto. Vosk puede utilizarse tanto de forma local como integrado en aplicaciones de servidor, donde puede exponerse a través de APIs para su consumo por otros sistemas. Una de las características más relevantes de Vosk Browser es que su ejecución en el navegador es posible gracias al uso de WebAssembly, una tecnología que permite ejecutar código compilado con un rendimiento cercano al nativo dentro del entorno web. Gracias a ello, el motor de reconocimiento puede funcionar completamente en el lado del cliente.

El flujo de funcionamiento puede resumirse en los siguientes pasos:

- Carga del modelo de reconocimiento de voz en el navegador.
- Captura del audio del usuario a través del micrófono.
- Procesamiento local de la señal de audio mediante el modelo.
- Generación de la transcripción en texto.

[4] `https://github.com/jonbgamble/vosk-browser`
[5] `https://alphacephei.com/vosk/`

Este enfoque presenta varias ventajas. Por un lado, reduce la latencia al evitar el envío de datos a servidores externos. Por otro, mejora la privacidad del usuario, ya que el audio no abandona el dispositivo. Además, permite el funcionamiento sin conexión a internet una vez cargado el modelo.

Sin embargo, también existen limitaciones. El tamaño de los modelos puede ser considerable, lo que implica tiempos de carga inicial elevados. Asimismo, la precisión puede ser inferior en comparación con soluciones basadas en modelos más recientes ejecutados en la nube, y el rendimiento depende directamente de los recursos disponibles en el dispositivo del usuario.

2.2.2. Whisper

Whisper[6] [21] es un modelo de reconocimiento de voz desarrollado por OpenAI, diseñado para ofrecer una alta precisión en la transcripción de audio en múltiples idiomas , incluso en ambientes con ruido o variabilidad en el habla.

Se trata de un modelo moderno basado en arquitecturas de tipo transformer encoder-decoder, entrenado con grandes volúmenes de datos de audio multilingüe. Gracias a ello, es capaz de realizar tareas como la transcripción automática, la detección de idioma o la traducción de audio.

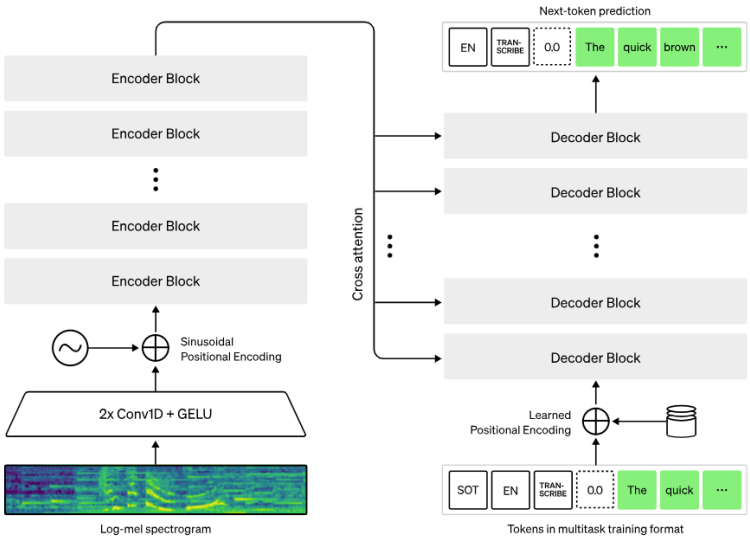


Figura 2.2: Arquitectura Whisper [21]

[6] https://openai.com/es-ES/index/whisper/

Desde el punto de vista de su utilización, Whisper puede emplearse de dos formas principales:

- Dentro del navegador cargando anteriormente el modelo.
- Mediante API, delegando el procesamiento en servicios externos que ejecutan el modelo.

Whisper ofrece distintos tamaños de modelo, lo que permite adaptar su uso en función del equilibrio entre precisión, velocidad y recursos disponibles. Los modelos de mayor tamaño ofrecen mejores resultados, pero requieren más capacidad de procesamiento y presentan mayor latencia.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~10x
base	74 M	`base.en`	`base`	~1 GB	~7x
small	244 M	`small.en`	`small`	~2 GB	~4x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x
turbo	809 M	N/A	`turbo`	~6 GB	~8x

Figura 2.3: Modelos Whisper [9]

A pesar de sus ventajas, el uso de Whisper en entornos locales presenta ciertas limitaciones, como el elevado consumo de CPU/GPU y el tamaño de los modelos, lo que dificulta su integración directa en aplicaciones web o en dispositivos con recursos limitados.

En este proyecto, Whisper se utiliza a través de la plataforma Groq, accediendo al modelo mediante API. Este enfoque contrasta con el uso de Vosk Browser, donde el procesamiento se realiza de forma local en el navegador.

El uso de APIs externas permite simplificar el desarrollo, evitando la necesidad de gestionar infraestructura propia y facilitando la escalabilidad del sistema. Además, plataformas como Groq están optimizadas para la ejecución de modelos de inteligencia artificial, lo que permite obtener tiempos de respuesta reducidos y un alto rendimiento.

Característica	Vosk	Whisper (local)
Precisión	Alta	Muy alta
Velocidad	Rápido	Medio-lento
Almacenamiento	Bajo (50MB-2GB)	Alto (1-10GB)
Idiomas	+20	99
Configuración	Fácil	Fácil
Tamaño	30MB-2GB	100MB-3GB
Compatibilidad con GPU	No	Sí

Cuadro 2.1: Comparativa entre Vosk y Whisper

2.2.3. Web Speech API

La Web Speech API[7] [18] permite a un navegador web acceder a características relacionadas con la voz. Constituye una de las principales tecnologías para reconocimiento de voz en los navegadores. Esta API está compuesta de dos componentes principales: por un lado, *SpeechSynthesis*, encargado de la conversión de texto a voz (síntesis de voz) y, por otro lado, *SpeechRecognition*, que permite el reconocimiento de voz asíncrono, es decir, este componente es el que se encarga de detectar cuándo se recibe voz del micrófono del dispositivo y tranformarla en texto. Una de las características principales de esta tecnología es que es bastante sencilla de usar porque tiene servicios de reconocimiento de voz integrados en el navegador y requiere pocas líneas de código para su implementación.

Entre sus principales ventajas destacan su facilidad de integración en aplicaciones web y su capacidad para ofrecer resultados en tiempo real, incluyendo transcripciones parciales mientras el usuario está hablando.

No obstante, no todos los navegadores soportan el uso de Web Speech API. Aunque, muchos de ellos ofrecen soporte, este puede variar en funcionalidad y implementación de uno a otro. Por ejemplo, en navegadores basados en Chromium como Google Chrome (desde la versión 33), el soporte está disponible mediante interfaces con prefijo, como *webkitSpeechRecognition*.

La Web Speech API se emplea como una de las soluciones principales para capturar la voz del usuario y convertirla en texto de forma rápida y sencilla. Su integración con JavaScript fa-

[7] `https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API`

Figura 2.4: Compatibilidad con navegadores WeebSpeechAPI [18]

cilita la emisión de eventos dentro de la aplicación, que posteriormente son procesados por el sistema para generar la escena correspondiente. Sin embargo, esta tecnología presenta limitaciones en determinados entornos, como los navegadores integrados en dispositivos de realidad virtual o aumentada, por ejemplo, las Meta Quest 2, donde no existe soporte. Por este motivo, el uso de esta tecnología se limita al entorno de escritorio. Otra de las limitaciones de esta tecnología es que tiene la necesidad de conexión a Internet, ya que el procesamiento de voz depende de servicios externos. Además, el comportamiento puede variar entre distintos navegadores, y su precisión es inferior en comparación con soluciones más avanzadas basadas en modelos de inteligencia artificial, como Whisper, especialmente en entornos con ruido. Finalmente, se ha seleccionado esta tecnología debido a su sencillez de implementación, ya que no requiere el uso de librerías externas, se ejecuta directamente en el navegador y resulta especialmente adecuada para prototipos, demostraciones o aplicaciones ligeras.

2.3. Comunicación con servicios externos

En esta sección se describen las tecnologías utilizadas para la comunicación entre el cliente (navegador) y los servicios externos necesarios para su funcionamiento, como modelos de lenguaje o sistemas de reconocimiento de voz. Para ello, se utiliza principalmente el protocolo HTTP junto con APIs de tipo REST para el intercambio de información.

2.3.1. HTTP

HTTP (HyperText Transfer Protocol)[8] [4] es el protocolo de comunicación utilizado para el intercambio de información entre clientes y servidores en la web. Se basa en un modelo de petición-respuesta, en el que el cliente envía una solicitud a un servidor y este devuelve una respuesta, normalmente en formatos estructurados como JSON o texto plano. Cada interacción HTTP se compone de una petición, que incluye información como la URL del recurso, el método utilizado (por ejemplo, GET o POST) y posibles datos adicionales, y una respuesta que contiene el resultado del procesamiento realizado por el servidor.

HTTP se utiliza para enviar peticiones desde la aplicación web a servicios externos, como OpenRouter o Groq. Estas peticiones incluyen información como el texto introducido por el usuario o el audio capturado, que es procesado por dichos servicios.

Las respuestas recibidas contienen la información necesaria para continuar con el flujo de la aplicación. Este intercambio de datos se realiza principalmente mediante el uso de JavaScript y la función `fetch`, que permite gestionar peticiones HTTP de forma asíncrona sin bloquear la ejecución de la aplicación.

2.3.2. APIs REST

Las APIs REST (Representational State Transfer)[9] [20] son un estilo de arquitectura que permite la comunicación entre sistemas a través de HTTP, utilizando operaciones estándar como:

- **GET**: para obtener información.
- **POST**: para enviar datos al servidor.
- **PUT / PATCH**: para actualizar recursos.
- **DELETE**: para eliminar información.

Estas APIs proporcionan una forma estructurada y sencilla de interactuar con servicios externos facilitando la integración entre diferentes sistemas.

[8] `https://datatracker.ietf.org/doc/html/rfc7231`
[9] `https://developer.mozilla.org/en-US/docs/Glossary/REST`

En este proyecto, la comunicación con servicios externos se realiza mediante peticiones HTTP desde el navegador utilizando JavaScript. Estas peticiones permiten interactuar con APIs que proporcionan funcionalidades clave del sistema.

El flujo de comunicación es el siguiente:

- El usuario introduce una instrucción mediante voz o texto.
- El sistema procesa esta entrada y realiza una petición HTTP a un servicio externo (por ejemplo, un modelo de lenguaje a través de Groq).
- El servicio procesa la solicitud y devuelve una respuesta en formato estructurado.
- La respuesta se interpreta y se transforma en una representación de la escena (en formato YAML).
- Finalmente, esta información se utiliza para actualizar dinámicamente la escena 3D en el navegador.

Este enfoque permite delegar el procesamiento complejo en servicios externos, reduciendo la carga en el cliente y facilitando el desarrollo de la aplicación. Además, el uso de APIs REST favorece la modularidad y escalabilidad del sistema, permitiendo integrar fácilmente nuevos servicios en el futuro.

La comunicación mediante HTTP y APIs REST permite conectar el entorno web con servicios avanzados de inteligencia artificial, haciendo posible que funcionalidades como el reconocimiento de voz o la generación de escenas 3D se integren de forma transparente en la aplicación.

2.4. Tecnologías web

En esta sección se presentan las tecnologías web utilizadas como base del desarrollo de la aplicación. En concreto, HTML5 y JavaScript permiten definir la estructura y la lógica del sistema, facilitando la creación de una aplicación interactiva capaz de generar y gestionar escenas de realidad virtual en el navegador.

2.4.1. HTML5

HTML5 (HyperText Markup Language)[10] [27] es el lenguaje de marcado utilizado para estructurar el contenido de las páginas web. Proporciona la base sobre la que se construye la aplicación web, definiendo los distintos elementos que componen la interfaz como texto, imágenes, enlaces o contenido multimedia.

Este lenguaje fue introducido originalmente por *Tim Berners-Lee* a principios de los años 90 en el CERN (Organización Europea para la Investigación Nuclear), con el objetivo de facilitar el intercambio de información entre científicos a través de una red basada en hipertexto. Desde entonces, HTML ha evolucionado significativamente hasta su versión actual, HTML5, adaptándose a las necesidades de las aplicaciones web modernas.

HTML se basa en el uso de etiquetas (*tags*) estructurando el contenido de forma jerárquica. Estas etiquetas indican al navegador cómo debe interpretar y mostrar la información.

HTML5 introduce importantes mejoras respecto a versiones anteriores como, por ejemplo, el uso de etiquetas semánticas, la integración de audio y vídeo de forma nativa, la posibilidad de renderizar gráficos dinámicos en el navegadr, la compatibilidad con WebGL y el uso de APIs avanzadas.

Además, HTML trabaja en conjunto con tecnologías como CSS, encargado de la presentación visual, y JavaScript, responsable de la lógica e interacción de la aplicación.

En este proyecto, HTML5 se utiliza como base estructural para definir la interfaz y el entorno de realidad virtual. En particular, se emplea junto con el framework A-Frame, que permite describir escenas tridimensionales mediante una sintaxis declarativa basada en etiquetas HTML, como `<a-scene>` o `<a-entity>`.

Este enfoque facilita la creación y comprensión del entorno 3D, ya que los elementos se definen de forma clara y estructurada. Posteriormente, estos elementos son manipulados mediante JavaScript para generar dinámicamente la escena en función de la interacción del usuario.

Asimismo, HTML5 permite integrar otros componentes de la interfaz, como menús, botones o scripts externos, favoreciendo una separación clara entre estructura, presentación y lógica. Esto contribuye a un desarrollo más organizado y mantenible. Su capacidad de integración con tecnologías como WebGL, WebXR o frameworks como A-Frame permite desarrollar experiencias interactivas avanzadas directamente en el navegador, incluyendo entornos de realidad

[10] `https://html.spec.whatwg.org/`

virtual y aumentada. Gracias a estas características, HTML5 se ha consolidado como una tecnología fundamental en el desarrollo de aplicaciones web modernas, especialmente en aquellos proyectos que requieren interactividad, multimedia y gráficos avanzados.

2.4.2. JavaScript

JavaScript[11] [19] es uno de los lenguajes de programación más utilizados en el desarrollo web y constituye un elemento fundamental para la creación de aplicaciones interactivas en el navegador. Se trata de un lenguaje de alto nivel, interpretado y orientado a eventos, que se ejecuta directamente en el navegador sin la necesidad de su previa compilación.

Se combina con tecnologías web como HTML, define la estructura, CSS define la presentación y JavaScript implementa la lógica y el comportamiento de la aplicación. Gracias a su integración con el navegador, permite interactuar con el Modelo de Objetos del Documento (DOM), lo que facilita la modificación del contenido y la apariencia de la aplicación en tiempo real.

Una de las características más relevantes de JavaScript es su capacidad para gestionar eventos, como clics, entradas de texto o interacciones del usuario, permitiendo crear interfaces dinámicas y reactivas. Además, ofrece soporte para programación asíncrona mediante mecanismos como *fetch API* o AJAX, lo que posibilita la comunicación con servidores sin recargar la página.

JavaScript se rige por el estándar ECMAScript, que define su sintaxis y funcionalidades, garantizando su evolución y compatibilidad entre diferentes entornos. Aunque originalmente fue diseñado para ejecutarse en el lado del cliente (front-end), actualmente también puede utilizarse en el servidor mediante entornos como Node.js, ampliando significativamente su ámbito de aplicación.

En este proyecto, JavaScript desempeña un papel central, ya que todo el comportamiento funcional del sistema se implementa mediante este lenguaje. Se encarga de coordinar los distintos componentes y tecnologías utilizadas. El uso de JavaScript permite la interacción con el usuario, la comunicación con APIs externas, el procesamiento de las respuestas y la actualización de la escena 3D en tiempo real.

[11] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

Además, JavaScript permite implementar una arquitectura basada en componentes, lo que facilita la modularidad, reutilización y mantenimiento del código. Gracias a su flexibilidad y capacidad de integración, JavaScript ha sido una herramienta clave para unificar todas las funcionalidades del sistema en un único entorno de ejecución.

2.5. APIs del navegador para XR

En esta sección se describen las principales APIs del navegador empleadas para el desarrollo de experiencias de realidad extendida (XR) en la web. Estas tecnologías permiten aprovechar las capacidades del navegador para crear y visualizar escenas en 3D interactivas sin necesidad de instalar software adicional.

En conjunto, estas APIs constituyen la base sobre la que se construyen las experiencias de realidad virtual en el navegador, permitiendo integrar de forma eficiente el renderizado, la visualización y la interacción dentro de la aplicación desarrollada.

2.5.1. WebXR

WebXR (Web Extended Reality)[12] [26] es una API estandarizada por el W3C del navegador diseñada para el desarrollo de experiencias de realidad virtual (VR) y realidad aumentada (AR) directamente en el navegador. Su objetivo principal es proporcionar una interfaz estándar que permita crear aplicaciones inmersivas en múltiples dispositivos, como gafas de realidad virtual, dispositivos móviles o equipos de escritorio, sin necesidad de instalar software adicional.

WebXR actúa como una capa de abstracción entre el navegador y el hardware del dispositivo, permitiendo a los desarrolladores acceder a funcionalidades avanzadas sin necesidad de implementar soluciones específicas para cada plataforma. Internamente, se apoya en tecnologías nativas del sistema, pero ofrece una interfaz unificada que simplifica el desarrollo multiplataforma.

Entre sus principales características destacan:

- **Renderizado de escenas 3D:** permite mostrar escenas en 3D en los dispositivos a altas velocidades para garantizar una experiencia inmersiva.

[12] `https://immersive-web.github.io/webxr/`

- **Seguimiento del movimiento:** detecta la posición y orientación del usuario y de los controladores para una mejor interactividad.

- **Gestión de dispositivos de entrada:** soporta distintos tipos de controladores y sistemas de interacción con el usuario.

- **Compatibilidad multiplataforma:** funciona en diferentes navegadores y dispositivos sin necesidad de plugins.

- **Integración con tecnologías web:** se apoya en HTML, JavaScript y WebGL para la creación de experiencias inmersivas.

Figura 2.5: Escena WebXR

Gracias a estas capacidades, WebXR permite desarrollar aplicaciones que pueden ejecutarse directamente desde el navegador, adaptándose automáticamente al dispositivo utilizado por el usuario. En combinación con frameworks como A-Frame, WebXR simplifica considerablemente la creación de experiencias complejas, permitiendo a los desarrolladores centrarse en la lógica de la aplicación mientras la API gestiona la interacción con el hardware.

[12]Ejemplos WebXR: `https://immersiveweb.dev/`

2.5.2. WebGL

WebGL (Web Graphics Library)[13] [12] es una API basada en JavaScript que permite renderizar gráficos bidimensionales y tridimensionales directamente en el navegador, haciendo uso de la capacidad de procesamiento de GPU del dispositivo. Esta tecnología está desarrollada por el consorcio Khronos Group y se basa en OpenGL ES 2.0, una versión adaptada de OpenGL para dispositivos embebidos.

Gracias a WebGL, es posible desarrollar aplicaciones 3D complejas sin necesidad de instalar complementos adicionales, ya que se integra directamente con tecnologías web estándar como HTML, CSS y JavaScript. Esto facilita la creación de entornos interactivos en el navegador, combinando elementos visuales con lógica de aplicación de forma eficiente.

Una de sus principales características es la capacidad de generar gráficos interactivos en tiempo real, lo que permite actualizar escenas y objetos tridimensionales de manera inmediata en respuesta a las acciones del usuario. Esto resulta fundamental en aplicaciones que requieren interacción continua, como simulaciones, videojuegos o entornos de realidad virtual.

Entre sus características más destacadas se encuentran:

- **Aceleración por hardware:** acceso directo a la GPU mediante el uso de shaders, lo que mejora significativamente el rendimiento.
- **Renderizado en tiempo real:** permite la actualización dinámica de escenas 3D de forma fluida.
- **Integración con tecnologías web:** funciona junto a HTML, CSS y JavaScript, facilitando el desarrollo de aplicaciones completas en el navegador.
- **Uso del elemento `<canvas>`:** proporciona un lienzo donde se renderizan los gráficos.
- **Compatibilidad multiplataforma:** soportado por la mayoría de navegadores modernos, independientemente del sistema operativo.

En este proyecto, WebGL actúa como la base tecnológica encargada del renderizado de la escena 3D generada dinámicamente. A partir de las instrucciones del usuario, procesadas

[13] `https://www.khronos.org/webgl/`

mediante reconocimiento de voz y modelos de lenguaje, se construye una representación estructurada de la escena que posteriormente es visualizada en el navegador.

Aunque WebGL no se utiliza directamente a bajo nivel, su funcionalidad es aprovechada a través de bibliotecas como Three.js y frameworks como A-Frame, que abstraen su complejidad y permiten desarrollar aplicaciones de forma más sencilla. Estas herramientas permiten definir objetos, transformaciones y animaciones de manera declarativa, mientras que WebGL se encarga de ejecutar el renderizado final utilizando la GPU.

De este modo, WebGL constituye el nivel más bajo de la pila tecnológica utilizada en el proyecto, siendo el motor gráfico que permite la representación de las escenas en 3D y su interacción en tiempo real.

2.6. Renderizado 3D y Realidad Virtual (VR)

En esta sección se presentan las tecnologías empleadas para el renderizado de gráficos en tres dimensiones y la construcción de entornos de realidad virtual en el navegador. En particular, se describen Three.js y A-Frame, dos herramientas fundamentales en el desarrollo del proyecto. Ambas tecnologías se complementan entre sí, permitiendo desarrollar entornos interactivos en realidad virtual de manera eficiente, combinando control técnico y facilidad de uso.

2.6.1. Three.js

Three.js[14] [24] es una biblioteca de JavaScript que permite la creación y renderizado de gráficos en 3D directamente en el navegador. Su principal objetivo es simplificar el uso de WebGL, proporcionando una capa de abstracción que facilita el desarrollo de aplicaciones 3D interactivas sin necesidad de trabajar directamente con una API de bajo nivel.

Gracias a esta abstracción, Three.js permite a los desarrolladores construir escenas 3D complejas de forma más sencilla, incorporando elementos como modelos 3D, iluminación, sombras y animaciones sin necesidad de gestionar aspectos técnicos como shaders, matrices o buffers gráficos. Por este motivo, Three.js es ampliamente utilizada en ámbitos como la visualización de datos, videojuegos, simulaciones y aplicaciones web interactivas.

[14] `https://threejs.org/`

A diferencia de WebGL, que requiere un conocimiento técnico más profundo, Three.js ofrece un conjunto de clases y utilidades que simplifican la manipulación de objetos dentro de una escena. Entre estos elementos destacan:

- **Escena (Scene):** contenedor principal donde se agrupan todos los elementos 3D.
- **Cámara (Camera):** define el punto de vista desde el cual se visualiza la escena.
- **Renderizador (Renderer):** encargado de transformar la escena en una imagen visible en el elemento `<canvas>` mediante WebGL.
- **Geometrías y mallas (Mesh):** representan los objetos 3D, combinando formas básicas o modelos complejos con materiales.
- **Materiales (Material):** determinan la apariencia visual de los objetos (color, textura, iluminación).
- **Luces (Light):** permiten simular iluminación realista dentro de la escena.

Además, Three.js permite importar modelos en formatos como GLTF, aplicar animaciones y gestionar interacciones. También, optimiza el renderizado, ya que implementa técnicas que mejoran el rendimiento, como la gestión eficiente de objetos visibles o el uso adecuado de recursos de la GPU, lo que permite crear experiencias fluidas en escenas complejas.

Three.js se sitúa como una capa intermedia entre WebGL, motor de renderizado que ejecuta las operaciones gráficas en la GPU y, A-Frame, que se sitúa como una capa de mayor nivel sobre Three.js. Cada entidad definida en A-Frame se traduce en objetos de Three.js que son finalmente renderizados mediante WebGL. En resumen, Three.js proporciona el control y la potencia gráfica, mientras que A-Frame facilita su uso en entornos web.

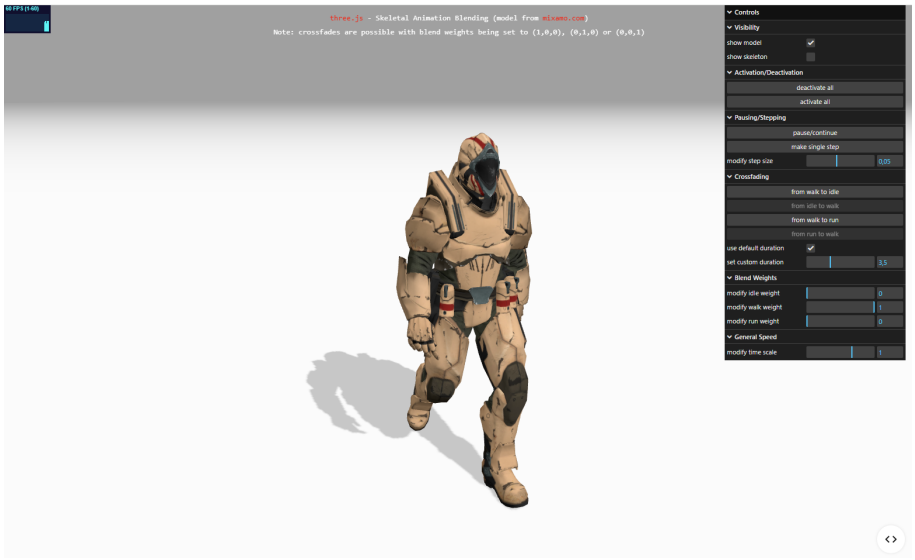


Figura 2.6: Escena Three.js robot animado

En la Figura 2.6 se puede ver una escena en Three.js que muestra un robot 3D animado caminando sobre un fondo simple. A la derecha se observa un panel de control para gestionar animaciones, transiciones y velocidad del modelo.

En este proyecto, Three.js actúa como motor gráfico responsable del renderizado y de todas las transformaciones aplicadas en la escena. Aunque el desarrollador no interactúe directamente con Three.js en la mayoría de los casos, esta biblioteca constituye un componente esencial dentro de la arquitectura del sistema. Su estabilidad, rendimiento y compatibilidad multiplataforma garantizan que las escenas 3D se representen correctamente y con fluidez en el navegador.

2.6.2. A-Frame

A-Frame[15] [23] es un framework web de código abierto que permite el desarrollo de escenas en 3D y experiencias en realidad virtual (VR) y realidad aumentada (AR) desde el navegador web. Está construido sobre Three.js y utiliza una arquitectura basada en entidades[16] y componentes [17], lo que facilita la creación de escenas interactivas de forma modular y flexible. Permite definir escenas 3D utilizando HTML, lo que lo hace especialmente accesible a cualquier desarrollador. Este framework es compatible con una amplia variedad de dispositivos de realidad

[14]Ejemplos Three.js: `https://threejs.org/examples/`
[15]`https://aframe.io/`
[16]Entidades A-Frame: `https://aframe.io/docs/1.6.0/core/entity.html`
[17]Componentes A-Frame: `https://aframe.io/docs/1.6.0/core/component.html`

virtual como HTC Vive, Oculus Rift, Oculus Go, Windows Mixed Reality o Meta Quest, así como con dispositivos móviles y ordenadores de escritorio. Además, también puede emplearse para realidad aumentada. A continuación, se muestra un ejemplo básico de una escena en A-Frame, donde se puede observar que mediante etiquetas HTML se definen los distintos elementos que componen la escena en realidad virtual.

```html
<html>
  <head>
    <script src="https://aframe.io/releases/1.7.1/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
      <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

Figura 2.7: Ejemplo A-Frame HTML

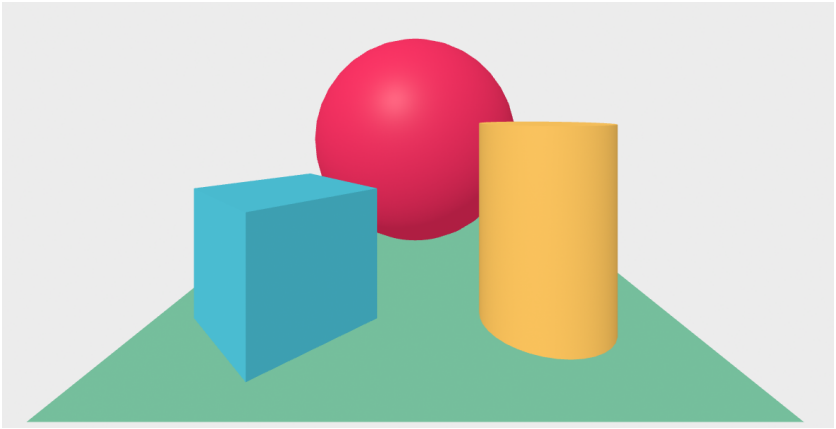


Figura 2.8: Ejemplo A-Frame en navegador correspondiente al código de la Figura 2.7

Las principales características de A-Frame son las siguientes:

1. **Rendimiento optimizado**: Aunque A-Frame utiliza el DOM para definir los elementos de la escena, su diseño está optimizado para trabajar con gráficos 3D sin interferir con el rendimiento del navegador, apoyándose en WebGL a través de Three.js.

2. **Escalabilidad**: La estructura modular basada en componentes permite desarrollar escenas cada vez más complejas manteniendo una buena organización del código y un rendimiento adecuado

[17]Ejemplos A-Frame: `https://aframe.io/`

3. **Sistema de componentes**: Los componentes son una pieza clave en A-Frame, ya que permiten añadir comportamientos específicos a las entidades. Gracias a ellos, es posible implementar funcionalidades como animaciones, iluminación o interacción de forma reutilizable y organizada.

4. **Arquitectura Entidad-Componente**: A-Frame sigue un modelo basado en Three.js compuesta por una estructura entidad-componente declarativa, donde cada entidad puede incorporar diferentes comportamientos mediante componentes reutilizables. Este enfoque favorece la modularidad y la organización del código, permitiendo crear funcionalidades complejas de manera estructurada.

5. **Uso de HTML declarativo**: Las escenas y los objetos en 3D se describen mediante HTML, lo que facilita su lectura y comprensión. Esto permite desarrollar entornos 3D de forma intuitiva, especialmente para desarrolladores web.

6. **VR multiplataforma**: Las aplicaciones desarrolladas con A-Frame pueden ejecutarse en distintos dispositivos, desde ordenadores o dispositivos móviles hasta visores de realidad virtual, garantizando una amplia accesibilidad.

7. **Inspector visual integrado**: Dispone de una herramienta de inspección que permite visualizar y modificar la escena en tiempo real, facilitando el desarrollo y la depuración de las escenas en realidad virtual.

8. **Facilidad para el desarrollo de VR**: A-Frame simplifica considerablemente la creación de experiencias de realidad virtual. Basta con incluir la librería en el documento HTML y definir una escena mediante las etiquetas `<a-scene>` y `<script>` para comenzar a trabajar, reduciendo la complejidad técnica inicial.

2.7. Dispositivos de realidad virtual

2.7.1. Meta Quest

Meta Quest[18] [16] es una familia de dispositivos de realidad virtual desarrollados por Meta (anteriormente Facebook), que permiten experimentar entornos inmersivos sin necesidad de

[18] `https://www.meta.com/es/quest/`

conexión a un ordenador o consola. Estos dispositivos, introducidos inicialmente bajo la marca Oculus, han evolucionado hacia sistemas autónomos que integran tanto el hardware como el software necesarios para ejecutar aplicaciones de realidad virtual directamente en el propio visor.

La principal ventaja de los dispositivos Meta Quest es su capacidad de funcionamiento independiente, lo que facilita el acceso a experiencias de realidad virtual de forma sencilla y sin configuraciones complejas. Gracias a su diseño autónomo, el usuario puede interactuar con entornos virtuales directamente desde el navegador o mediante aplicaciones específicas, lo que los convierte en una herramienta especialmente adecuada para el desarrollo de experiencias inmersivas.

Además, los dispositivos Meta Quest incorporan sistemas de seguimiento posicional que permiten detectar la orientación y el movimiento del usuario en el espacio, mejorando la sensación de presencia dentro del entorno virtual. Esto contribuye a una experiencia más realista, ya que el usuario puede moverse y observar la escena desde distintas perspectivas.

Las versiones más recientes de la familia, como Meta Quest 3 y Meta Quest 3S, han mejorado tanto el rendimiento como la calidad visual, incorporando procesadores más potentes, pantallas de mayor resolución y tasas de refresco más elevadas. Estas mejoras permiten ejecutar aplicaciones más complejas y ofrecer experiencias más fluidas e inmersivas.

Figura 2.9: Gafas Meta Quest 2

En la Figura 3.1 se observan, a los lados izquierdo y derecho, los controladores de las gafas de realidad virtual, mientras que en el centro se encuentra el visor.

2.8. Otras tecnologías utilizadas

2.8.1. YAML

YAML (*YAML Ain't Markup Language*)[19] [28] es un formato de serialización de datos que se caracteriza por ser legible y fácilmente interpretable por humanos. Se utiliza comúnmente para representar configuraciones y estructuras de datos de forma clara y sencilla. Este formato permite estructurar la información de manera jerárquica, facilitando su posterior procesamiento mediante JavaScript.

2.8.2. GitHub

GitHub[20] [6] es una plataforma de desarrollo colaborativo de software basada en Git que permite almacenar, versionar y gestionar código fuente en repositorios remotos. Además de servir como sistema de control de versiones, facilita la colaboración, el seguimiento de cambios y la integración con herramientas de automatización. Un repositorio en GitHub almacena el código de manera eficiente y ordenada. GitHub se utiliza como repositorio principal para almacenar el código de la aplicación.

2.8.3. Integración continua

La integración continua es una práctica dentro del desarrollo de software que consiste en la integración frecuente de los cambios realizados en el código en un repositorio común. Cada una de estas integraciones se valida automáticamente mediante la ejecución de procesos como la compilación, pruebas o verificación del sistema.

El objetivo principal de la integración continua es detectar errores de forma temprana, mejorar la calidad del software y facilitar el desarrollo colaborativo. Al automatizar tareas repetitivas, se reduce la probabilidad de errores humanos y se agiliza el ciclo de desarrollo.

[19] `https://yaml.org/`

[20] `https://docs.github.com/es/repositories/creating-and-managing-repositories/quickstart-for-repositories`

2.8.3.1. GitHub Actions

GitHub Actions[21] [5] es la plataforma de automatización integrada en GitHub, orientada a la ejecución de flujos de trabajo definidos mediante archivos YAML. Estos flujos permiten automatizar tareas de integración y despliegue continuo, como compilar, probar o publicar una aplicación cuando se produce un determinado evento en el repositorio, por ejemplo un *push* o un *commit*.

2.8.4. Visual Studio Code

Visual Studio Code (VS Code)[22] [17] es un editor de código fuente desarrollado por Microsoft ampliamente utilizado en el desarrollo de aplicaciones web. Se trata de una herramienta ligera pero muy potente, que permite trabajar con múltiples lenguajes de programación y ofrece un entorno altamente configurable mediante extensiones.

Una de sus principales ventajas es su facilidad de uso junto con funcionalidades avanzadas como el resaltado de sintaxis, autocompletado de código, depuración integrada y control de versiones mediante Git. Estas características permiten mejorar la productividad y facilitan el desarrollo de aplicaciones complejas.

2.8.5. LaTeX

LaTeX[23] [13] es un sistema de composición de textos de alta calidad ampliamente utilizado para documentos científicos y técnicos. Se basa en un lenguaje de marcado que permite estructurar el contenido de forma precisa. En este proyecto, LaTeX se ha utilizado para la redacción de la memoria, permitiendo mantener una estructura clara, uniforme y visual.

2.9. Trabajos relacionados

En este apartado se mencionan aplicaciones que, desde distintos puntos de vista, están relacionadas con el proyecto o han servido de inspiración para su desarrollo.

[21] `https://docs.github.com/es/actions/get-started/understand-github-actions`
[22] `https://code.visualstudio.com/docs`
[23] `https://www.latex-project.org/`

2.9.1. vTime XR

vTime XR[24] [25] es una plataforma social en realidad virtual que permite a los usuarios interactuar en entornos tridimensionales compartidos, accesibles desde dispositivos VR, móviles o PC.

Figura 2.10: Ejemplo de entorno en vTime XR

Esta aplicación se basa en la creación de espacios virtuales en forma de habitaciones o escenarios en los que los usuarios pueden reunirse, comunicarse y explorar el entorno. Los espacios están predefinidos y pueden personalizarse en cierta medida, permitiendo elegir diferentes ambientaciones o configuraciones.

vTime XR demuestra el potencial de los entornos virtuales como espacios habitables y dinámicos. Sin embargo, la creación y modificación de estos entornos se realiza mediante interfaces tradicionales y opciones limitadas, sin integración de mecanismos avanzados de generación automática o interacción mediante lenguaje natural.

2.9.2. Waltz of the Wizard

Waltz of the Wizard[25] [1] es una experiencia inmersiva en realidad virtual que permite al usuario interactuar con un entorno mágico mediante diferentes mecanismos, incluyendo el uso de comandos de voz.

[24] https://vtime.net/

[25] https://www.waltzvirtual.com/

Figura 2.11: Ejemplo de interacción en Waltz of the Wizard

En esta aplicación, el usuario puede emitir órdenes verbales para realizar acciones dentro del entorno, como interactuar con objetos o activar efectos. Este tipo de interacción permite una experiencia más natural, reduciendo la dependencia de controladores físicos.

No obstante, los comandos de voz en este tipo de sistemas suelen estar basados en un conjunto limitado de instrucciones predefinidas, lo que restringe la flexibilidad del usuario y la complejidad de las acciones que se pueden realizar.

2.9.3. Sistemas basados en lenguaje natural

Los sistemas de interacción basados en lenguaje natural, ya sea hablado o escrito, están cobrando una gran relevancia en ámbitos como el diseño asistido, la generación de contenido y la creación de entornos tridimensionales. Estas herramientas permiten a los usuarios describir lo que desean mediante lenguaje natural, delegando en modelos de inteligencia artificial la generación del contenido correspondiente.

Entre las soluciones más destacadas se encuentran herramientas como Meshy AI[26] [15], que permite generar modelos 3D a partir de descripciones textuales o imágenes, facilitando la creación rápida de objetos sin necesidad de conocimientos técnicos avanzados. De forma

[26] `https://www.meshy.ai/`

similar, Luma AI[27] [14] permite reconstruir escenas tridimensionales a partir de datos visuales, utilizando técnicas avanzadas de renderizado neuronal para generar entornos realistas.

Figura 2.12: Ejemplo de modelo 3D generado mediante Meshy AI

Estos sistemas se apoyan en modelos de lenguaje grandes (LLMs) y técnicas de generación condicional para interpretar las instrucciones del usuario y producir resultados coherentes. En la mayoría de los casos, la interacción se realiza mediante texto y a través de interfaces tradicionales.

El proyecto desarrollado en este trabajo introduce una propuesta diferenciadora, al combinar el reconocimiento de voz con modelos de lenguaje, permitiendo generar y modificar escenas tridimensionales de forma dinámica directamente desde el navegador. Además, el sistema se centra en la manipulación de modelos genéricos ya cargados, lo que simplifica la interacción y reduce la complejidad para el usuario.

[27] https://lumalabs.ai/

Capítulo 3

Desarrollo del proyecto

El desarrollo de este proyecto se ha llevado a cabo siguiendo una estrategia basada en iteraciones sucesivas, inspirada en los principios de las metodologías ágiles[1] [11] y, en particular, en SCRUM[2] [10]. Aunque no se ha implementado de forma estricta, sí se ha utilizado su marco de trabajo como guía para facilitar el avance del proyecto de forma incremental.

Figura 3.1: Metodología SCRUM seguida en el proyecto

Para ello, el proyecto se ha dividido en diferentes sprints, cada uno con una duración li-

[1] https://agilemanifesto.org/iso/es/principles.html
[2] https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf

mitada y orientado a la consecución de objetivos específicos. Al término de cada uno de estos ciclos se ha obtenido un resultado funcional, lo que ha permitido evaluar el estado del proyecto, identificar posibles mejoras y reajustar la planificación de los siguientes pasos.

En este contexto, el tutor ha asumido un papel fundamental, asumiendo la responsabilidad en la definición y evaluación de los objetivos. Por otro lado, el desarrollo técnico y la ejecución de las tareas han sido mi responsabilidad, lo que ha permitido una gestión directa y continua del proceso.

Cada sprint se ha estructurado en torno a una planificación previa, en la que se han definido las tareas necesarias para alcanzar los objetivos planteados, así como una revisión posterior en la que se han analizado los resultados obtenidos. Dentro de cada sprint, se ha llevado a cabo una reunión semanal enfocada al seguimiento del progreso, lo que ha favorecido la detección temprana de problemas y la incorporación continua de mejoras a lo largo del proceso.

En las secciones siguientes se detalla el desarrollo completo del proyecto, organizado por sprints. En cada uno de ellos se describen los objetivos propuestos, las tareas realizadas, los resultados obtenidos y las lecciones aprendidas, proporcionando así una visión clara del proceso de evolución del proyecto.

3.1. Sprint 1

3.1.1. Objetivos

El objetivo principal de este primer sprint ha sido adquirir conocimientos básicos de las tecnologías base del proyecto, familiarizarse con el **uso de A-Frame** para el desarrollo de escenarios en realidad virtual accesibles desde el navegador. Para ello, se requería la creación de escenas básicas que permitieran comprender el funcionamiento del framework A-Frame con algún componente específico y con objetos primitivos. Asimismo, se ha querido evaluar la compatibilidad de estas soluciones en distintos navegadores. De forma complementaria, se ha definido como objetivo la introducción a los sistemas de reconocimiento de voz, en concreto, la investigación e implementación de **Web Speech API**, cuya tecnología es compatible con navegadores de escritorio.

3.1.2. Tareas realizadas

- **Investigación de la documentación oficial de A-Frame**
  Se ha realizado un estudio inicial de la documentación oficial de A-Frame con el objetivo de comprender su funcionamiento, su arquitectura basada en entidades y componentes, y las posibilidades que ofrece para el desarrollo de entornos de realidad virtual en navegador.

- **Creación de escenas con el componente aframe-lounge y con objetos primitivos de A-Frame**
  Se han desarrollado las primeras escenas utilizando el componente `aframe-lounge`, permitiendo generar habitaciones básicas de ejemplo. Además, se han construido escenas simples empleando primitivas de A-Frame como cubos, cilindros, esferas y planos, con el fin de comprender la creación, manipulación y propiedades de los elementos.

- **Ejecución de escenas en distintos navegadores de escritorio**
  Se han realizado pruebas de ejecución en diferentes navegadores como *Google Chrome*, *Firefox* y *Microsoft Edge* para evaluar la compatibilidad de A-Frame.

- **Investigación sobre sistemas de reconocimiento de voz en aplicaciones web**
  Se han analizado distintas alternativas de sistemas de reconocimiento de voz en entorno web, evaluando su viabilidad y nivel de integración con aplicaciones basadas en navegador y su posterior funcionamiento en las gafas de realidad virtual.

- **Implementación de un prototipo usando Web Speech API**
  Se ha implementado un primer prototipo de reconocimiento de voz utilizando la Web Speech API, permitiendo transcribir la voz del usuario.

3.1.3. Resultados

En este sprint se ha conseguido una primera toma de contacto con la realidad virtual mediante el uso de A-Frame. Se han creado distintas escenas básicas en las que se han incluido tanto objetos primitivos, como cubos y esferas de distintos colores y tamaños, como escenas más complejas con el uso de componentes específicos, lo que ha permitido comprender algunas de las posibilidades del framework.

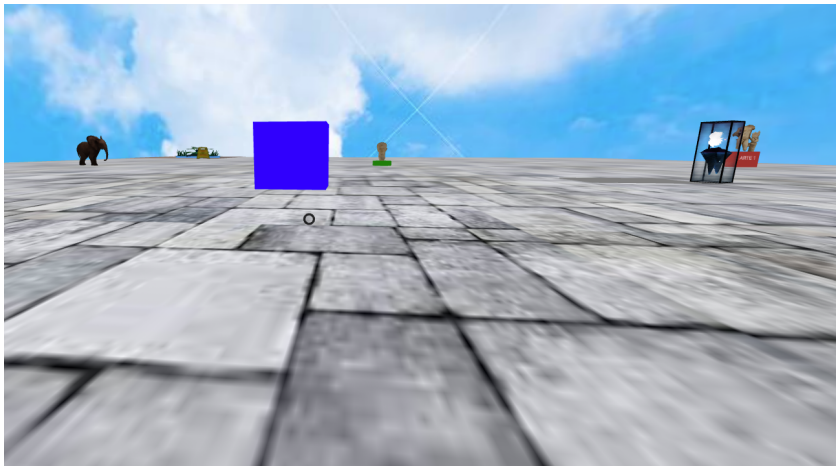


Figura 3.2: Ejemplo de escena A-Frame

Además, en la creación de estas distintas escenas se ha construido una habitación utilizando el componente *aframe-lounge*, integrando distintos elementos dentro del mismo entorno virtual.

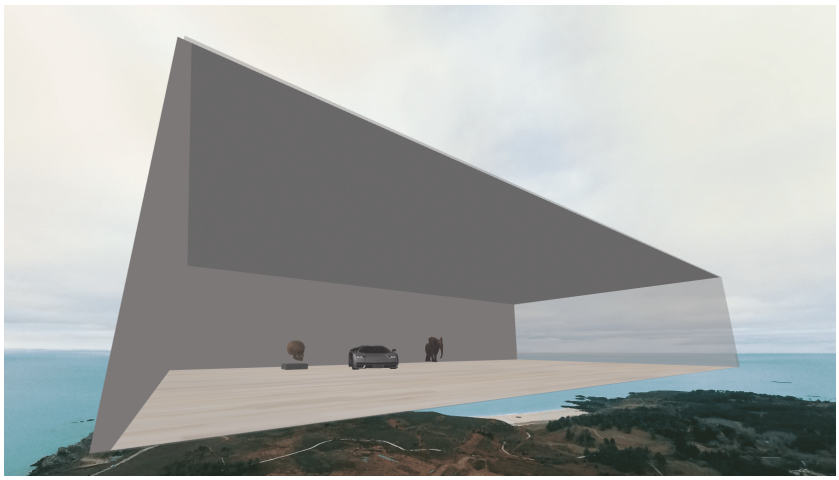


Figura 3.3: Ejemplo de habitación con componente aframe-lounge

En cuanto a la compatibilidad de los navegadores, se ha verificado el correcto funcionamiento de las escenas, aunque se han identificado ciertas diferencias en el soporte de algunas funcionalidades. Por otro lado, se ha implementado con éxito un primer sistema de reconocimiento de voz basado en la Web Speech API, capaz de capturar y transformar la entrada de voz en texto dentro del navegador sacando el resultado por consola.

3.1.4. Lecciones aprendidas

Durante el desarrollo de este sprint se han aprendido las siguientes lecciones:

- Adquisición de conocimientos sobre el uso del framework A-Frame para la creación de escenas en realidad virtual mediante HTML, lo que permite definir y editar elementos de forma declarativa simplificando el desarrollo
- Comprensión del empleo y de las propiedades de los objetos primitivos (cubos, esferas, planos, entre otros) como base para la construcción de escenas en realidad virtual
- Identificación de las limitaciones en los objetos primitivos, ya que, no permiten representar modelos complejos
- Integración de Web Speech API para el reconocimiento de voz de manera sencilla
- Detección de limitaciones en Web Speech API, ya que, no es compatible con todo tipo de navegadores y, además, no es compatible con navegadores utilizados en dispositivos de realidad virtual
- Necesidad de explorar soluciones alternativas de reconocimiento de voz que sean compatibles con entornos de realidad virtual
- Establecimiento de una base técnica inicial para el desarrollo del proyecto, identificando tanto herramientas útiles como sus limitaciones en el contexto de la realidad virtual

3.2. Sprint 2

3.2.1. Objetivos

El objetivo principal de este segundo sprint ha sido investigar e integrar modelos de lenguaje dentro del proyecto, con el fin de transformar descripciones en lenguaje natural en una representación estructurada de escenas virtuales. Para ello, se ha planteado el uso de los modelos accesibles a través de OpenRouter, estableciendo como objetivo hacer un programa en Python, realizar llamadas desde este programa y obtener respuestas válidas del modelo. A partir de esta base, el siguiente objetivo es la construcción de un prompt capaz de generar una descripción estructurada en formato YAML a partir de la entrada del usuario. En esta parte del proyecto, el trabajo se ha enfocado en la interpretación del lenguaje natural, priorizando la correcta identificación de características básicas como las dimensiones de la habitación. Finalmente, se ha

incluido el modelo Whisper de Groq como nuevo sistema de reconocimiento de voz para el proyecto.

3.2.2. Tareas realizadas

- **Investigación sobre el uso de modelos de lenguaje a través de OpenRouter**
  Se ha realizado un estudio inicial sobre el funcionamiento de OpenRouter y su uso para acceder a distintos modelos de lenguaje desde aplicaciones externas.

- **Realización de llamadas iniciales a distintos modelos para evaluar su funcionamiento y calidad de respuesta**
  Se han llevado a cabo pruebas con diferentes modelos gratuitos como, por ejemplo, *stepfun/step-3.5-flash:free* y *nvidia/nemotron-3-nano-30b-a3b:free* , para analizar su comportamiento, tiempos de respuesta y viabilidad para su uso en el proyecto.

- **Desarrollo de un programa en Python para automatizar las peticiones al modelo de lenguaje a través de OpenRouter**
  Se ha implementado un programa en Python que permite realizar llamadas automatizadas a los modelos y gestionar las respuestas obtenidas.

- **Diseño y ajuste de un prompt orientado a la generación de escenas a partir de descripciones en lenguaje natural**
  Se ha definido un prompt capaz de transformar descripciones textuales en una representación estructurada de una escena en formato YAML.

- **Diseño y Implementación de la interpretación del YAML generado para su representación en una escena**
  Se ha desarrollado un mecanismo para procesar el YAML generado por el LLM y traducirlo a una representación visual en el entorno.

- **Investigación, integración y pruebas del modelo Whisper a través de Groq como sistema de reconocimiento de voz**
  Se ha investigado e integrado el modelo Whisper a través de Groq, realizando pruebas para evaluar su funcionamiento como sistema de reconocimiento de voz en distintos navegadores.

3.2.3. Resultados

En este sprint se han conseguido integrar distintos modelos de lenguaje a través de OpenRouter. Para ello, se ha preparado un programa en Python que hace una llamada vía API al modelo de lenguaje correspondiente a través de OpenRouter para generar su posterior respuesta y su obtención. Se ha desarrollado un prompt inicial capaz de interpretar descripciones en lenguaje natural y generar una representación estructurada de una habitación en formato YAML. Se ha comprobado que no todos los modelos disponibles de OpenRouter funcionan igual para generar esta representación estructurada en formato YAML. Hay algunos modelos que lo generan mejor que otros y, unos son más rápidos y más eficientes. Además, estandarizar la salida en un modelo estructurado ha sido fundamental para garantizar la modularidad del sistema y facilitar el procesamiento por parte de los componentes de la escena. Un ejemplo propuesto es que dado el siguiente comando: ”Quiero una habitación de dimensiones 20x20x20”, produce una salida en estructura YAML como:

```
room:
  width: 20
  depth: 20
  height: 20
  ceiling: true
```

Esta salida YAML será procesada para obtener y asignar los valores correspondientes para poder visualizarlos en la escena.

Por otro lado, se ha implementado con éxito un segundo sistema de reconocimiento de voz basado en el modelo Whisper. Comprobando su funcionamiento tanto en entornos de escritorio como en entornos de realidad virtual, comprobando así que la API de grabación de audio mediante *MediaRecorder* se ejecutaba correctamente en ambos entornos.

3.2.4. Lecciones aprendidas

Durante el desarrollo de este sprint se han aprendido las siguientes lecciones:

- Uso de modelos de lenguaje (LLM) para la transformación de descripciones en lenguaje natural en estructuras formales de datos

- Identificación de la importancia del diseño de *prompts* consistentes para obtener resultados correctos y reproducibles en los LLM
- Estandarización de la salida de los LLM en formato YAML, con el objetivo de facilitar el procesamiento posterior de la información
- Integración de Whisper para el reconocimiento de voz identificando su compatibilidad con dispositivos de realidad virtual y mayor robustez frente a ruido y variaciones en la pronunciación en comparación con Web Speech API
- Detección de limitaciones en Whisper, ya que, requiere el envío a través de una API externa
- Identificación de la necesidad de una solución alternativa de reconocimiento de voz que sea compatible tanto en entornos de escritorio como de realidad virtual, no dependa de servicios externos y pueda ejecutarse directamente en el navegador.

3.3. Sprint 3

3.3.1. Objetivos

El objetivo principal de este tercer sprint ha sido avanzar en la integración de los diferentes sistemas de reconocimiento de voz y su conexión con los modelos de lenguaje, con el fin de construir una escena a partir de la voz. En primer lugar, se ha investigado y se ha definido implantar un sistema de reconocimiento de voz cargado directamente en el navegador llamado vosk-browser, debido a su capacidad de ejecutarse tanto en el navegador como en los dispositivos de realidad virtual. Una vez validado el funcionamiento de este sistema, se ha definido el objetivo de desarrollar una aplicación web que permita unificar distintos sistemas de reconocimiento de voz, incluyendo Web Speech API, Whisper y Vosk-browser, con el propósito de comparar los pros y los contras de cada modelo y disponer de múltiples alternativas de entrada. Posteriormente, se ha buscado integrar el texto que se ha transcrito de lo que ha dicho el usuario mediante voz con modelos de lenguaje a través de OpenRouter, de forma que el texto reconocido pueda ser procesado para generar una descripción estructurada de la escena en formato YAML. Finalmente, se ha establecido como objetivo investigar el uso de GitHub Actions para

automatizar un flujo de trabajo, en el cual el usuario diga un comando y se haga un commit automáticamente del YAML generado por el modelo de lenguaje en el repositorio. En este flujo de trabajo que se ejecuta, la idea es crear un nuevo HTML y recrear la escena que el usuario haya querido guardar. Esta escena se despliega y es accesible mediante GitHub Pages.

3.3.2. Tareas realizadas

- **Investigación, integración y pruebas del modelo vosk-browser como sistema de reconocimiento de voz**
  Se ha investigado e integrado el modelo vosk-browser, evaluando su funcionamiento como solución de reconocimiento de voz ejecutada directamente en el navegador.

- **Realización de pruebas para validar su funcionamiento y compatibilidad con navegadores de escritorio y entornos de realidad virtual**
  Se han realizado pruebas para comprobar la compatibilidad del sistema en distintos navegadores y su comportamiento en entornos de realidad virtual.

- **Desarrollo de una aplicación HTML capaz de integrar distintos sistemas de reconocimiento de voz (Web Speech API, Whisper, vosk-browser)**
  Se ha desarrollado una aplicación web que unifica múltiples tecnologías de reconocimiento de voz, permitiendo comparar su funcionamiento y seleccionar la que el usuario elija.

- **Integración de este flujo con modelos de lenguaje a través de OpenRouter, permitiendo enviar el texto reconocido y recibir una respuesta procesada**
  Se ha integrado el reconocimiento de voz con modelos de lenguaje, estableciendo un flujo que transforma la entrada hablada en una respuesta procesada por el sistema.

- **Adaptación del sistema al contexto del proyecto, permitiendo describir una habitación mediante voz y obtener como salida un YAML representando dicha escena**
  Se ha adaptado el sistema para que el usuario pueda describir una escena mediante voz y obtener como resultado una representación estructurada en formato YAML. En cualquier otro caso, se muestra un error en el escenario.

- **Investigación sobre GitHub Actions y su uso en automatización de procesos**
  Se ha estudiado el funcionamiento de GitHub Actions como herramienta para automatizar tareas dentro del flujo de desarrollo.

- **Configuración del workflow**
  Se ha configurado un workflow que permite automatizar el procesamiento del YAML generado, la ejecución de scripts y el despliegue del resultado en GitHub Pages.

3.3.3. Resultados

En este sprint se ha conseguido integrar con éxito el modelo vosk-browser dentro de una aplicación web, validando su funcionamiento y, pese a su tiempo de carga inicial en el navegador, se coloca como el sistema de reconocimiento indicado para realizar las pruebas en los entornos de realidad virtual, ya que, es compatible y no depende de ninguna API externa. Asimismo, se ha desarrollado una aplicación capaz de utilizar múltiples sistemas de reconocimiento de voz, permitiendo capturar la voz del usuario y transformarla en texto para el posterior paso al modelo de lenguaje. Se ha logrado completar un flujo funcional en el que usuario describe una habitación mediante voz, el sistema transforma esa entrada en texto, la envía a un modelo de lenguaje a través de OpenRouter y recibe como respuesta un YAML que representa la escena. Además, se ha implantado un sistema automatizado mediante GitHub Actions. A partir de un archivo YAML, se crea y se despliega una escena en formato HTML accesible a través de GitHub Pages. Posteriormente, esto se implantará en el proyecto mediante un comando de voz en el usuario, para guardar un escenario concreto del proyecto.

3.3.4. Lecciones aprendidas

Durante el desarrollo de este sprint se han aprendido las siguientes lecciones:

- Selección de tecnologías de reconocimiento de voz en función del entorno de ejecución (navegador de escritorio frente a dispositivos de realidad virtual)

- Evaluación de la librería Vosk-browser, identificando que es compatible con entornos de navegador, puede ejecutarse en dispositivos de realidad virtual y permite el reconocimiento de voz sin dependencia de servicios externos.

- Integración de múltiples sistemas de reconocimiento de voz dentro de la arquitectura del sistema aportando una mayor flexibilidad en función del entorno de ejecución
- Implementación de un flujo que abarca desde la captura de voz hasta el despliegue del resultado en la escena
- Automatización del proceso de integración continua mediante GitHub Actions, observando que reduce la intervención manual y facilita el despliegue de escenas generadas

3.4. Sprint 4

3.4.1. Objetivos

El objetivo principal de este cuarto sprint ha sido trasladar toda la lógica desarrollada en un único programa de Python a un entorno completamente basado en JavaScript, con el fin de facilitar el uso dentro de A-Frame. Para ello, se ha adoptado una arquitectura basada en componentes, lo que ha permitido estructurar el proyecto de forma modular, flexible y escalable. Este enfoque tiene como finalidad facilitar la reutilización de funcionalidades, crear más tipos de escena y mejorar la mantenilidad del sistema. Asimismo, otro de los objetivos que se ha planteado en este sprint es enriquecer la escena virtual mediante la incorporación de objetos en 3D y primitivas de A-Frame dentro de la habitación, mejorando tanto la representación visual como la interacción con el entorno. Finalmente, se ha buscado consolidar un flujo completo dentro del navegador de escritorio, en el que el usuario puede describir un escenario mediante voz, procesarlo a través de un modelo de lenguaje, genere una respuesta con el contenido de la escena y visualizar el resultado directamente en el DOM.

3.4.2. Tareas realizadas

- **Migración de la lógica previamente implementada en Python a JavaScript, adaptando las llamadas a modelos de lenguaje y el procesamiento de datos al entorno web**
  Se ha trasladado la lógica del sistema desde Python a JavaScript, permitiendo su ejecución directa en el navegador e integrando las llamadas a modelos de lenguaje en el entorno web.

- **Diseño e implementación de una arquitectura basada en componentes de A-Frame**
  Se ha definido una estructura modular basada en componentes de A-Frame, facilitando la organización, reutilización y escalabilidad del proyecto.

- **Implementación del flujo completo mediante componentes independientes**
  Se ha desarrollado el flujo completo del sistema dividiéndolo en componentes independientes, desde la captura de voz hasta la generación y visualización de la escena.

- **Incorporación de objetos 3D en formato GLB/GLTF**
  Se han añadido objetos 3D a la escena para mejorar la representación visual y enriquecer el entorno generado.

- **Incorporación de primitivas A-Frame**
  Se han utilizado primitivas de A-Frame para construir y estructurar los elementos básicos de las escenas de forma sencilla y eficiente.

- **Definición de comandos clave para controlar la ejecución del sistema y facilitar la interacción del usuario**
  Se han definido comandos específicos que permiten al usuario interactuar con el sistema y controlar su funcionamiento mediante voz.

- **Mejora progresiva de la calidad de las escenas generadas y de la interpretación de los comandos de voz**
  Se han realizado ajustes en el sistema para mejorar la precisión en la interpretación del lenguaje natural y la calidad de las escenas generadas.

- **Realización de pruebas orientadas al uso en entornos de escritorio, asegurando la estabilidad y funcionalidad del sistema**
  Se han llevado a cabo pruebas en entorno de escritorio para verificar el correcto funcionamiento del sistema y garantizar su estabilidad.

3.4.3. Resultados

En este sprint se ha conseguido transformar el proyecto en una aplicación completamente basada en tecnologías web, facilitando así su integración directa en el navegador. Se ha definido

una arquitectura modular basada en componentes de A-Frame, lo que ha permitido estructurar el proyecto de forma más escalable y flexible. Asimismo, se ha implementado un flujo completo funcional en el que un usuario puede describir un escenario mediante voz, el sistema procesa esa entrada de voz en texto y utiliza un modelo de lenguaje para generar una salida en formato YAML que representa la escena, la cual se visualiza directamente en el DOM. La incorporación de objetos 3D y primitivas ha enriquecido mucho la visualización del escenario, aportando mayor realismo y claridad a la escena. Además, la definición de comandos específicos ha mejorado la experiencia del usuario.

3.4.4. Lecciones aprendidas

Durante el desarrollo de este sprint se han aprendido las siguientes lecciones:

- Definición de una arquitectura basada en componentes utilizando A-Frame
- La arquitectura basada en componentes mejora la modularidad y escalabiliad del sistema
- Definición de comandos para la interacción con la escena, permitiendo mejorar la funcionalidad del sistema
- Mejora de la escena desde el punto de vista funcional y de interacción con el usuario
- Identificación de limitaciones relacionadas con los modelos 3D en formato GLB/GLTF dado que pueden incrementar el consumo de recursos y el rendimiento del sistema
- Validación del funcionamiento del sistema en entornos de escritorio, comprobando su estabilidad en este contexto
- Identificación de la necesidad de adaptar el sistema para su ejecución en dispositivos de realidad virtual como siguiente fase del desarrollo

3.5. Sprint 5

3.5.1. Objetivos

El objetivo principal de este último sprint ha sido adaptar el proyecto desarrollado en entornos de escritorio para su funcionamiento en entornos de realidad virtual, concretamente en las

gafas de realidad virtual Meta Quest 2, permitiendo así una experiencia inmersiva. Asimismo, se ha planteado mejorar la visualización de las escenas generadas, optimizando el rendimiento del sistema. Finalmente, se ha definido como objetivo la preparación de distintos escenarios que permitan realizar demostraciones del proyecto en diferentes contextos.

3.5.2. Tareas realizadas

- **Integración del proyecto con dispositivos VR/AR, específicamente las gafas Meta Quest 2**

  Se ha adaptado el proyecto para su ejecución en las gafas Meta Quest 2, permitiendo su uso en un entorno de realidad virtual inmersivo.

- **Ajuste de los componentes de A-Frame para su correcto funcionamiento tanto en navegadores de escritorio como en entornos de realidad virtual**

  Se han modificado y ajustado los componentes para garantizar su compatibilidad y correcto comportamiento en ambos entornos.

- **Optimización del código**

  Se han realizado mejoras en el código con el objetivo de optimizar el rendimiento y asegurar una ejecución fluida del sistema.

- **Realización de pruebas de usuario en entornos VR**

  Se han llevado a cabo pruebas en realidad virtual para validar la experiencia de usuario y detectar posibles mejoras.

- **Preparación de distintos escenarios, facilitando la ejecución del proyecto de forma sencilla y controlada**

  Se han definido y preparado distintos escenarios para demostraciones, permitiendo una ejecución más sencilla y organizada del proyecto.

3.5.3. Resultados

En este sprint se ha conseguido adaptar el funcionamiento del proyecto para su funcionamiento en las gafas Meta Quest 2, ofreciendo así la posibilidad de ejecutar el proyecto tanto en

entornos de escritorio como en entornos de realidad virtual. Se ha logrado mantener el flujo del sistema dentro el entorno de realidad virtual de principio a fin, gracias a la arquitectura basada en componentes. Asimismo, las mejoras introducidas en la visualización han permitido obtener escenas más claras y mejor estructuradas. Por otro lado, la flexibilidad de la arquitectura del proyecto ha permitido la preparación de distintos escenarios para una serie de demostraciones.

3.5.4. Lecciones aprendidas

Durante el desarrollo de este sprint se han aprendido las siguientes lecciones:

- Adaptación del sistema a entornos de realidad virtual, identificando diferencias respecto a los entornos de escritorio en términos de rendimiento, visualización e interacción.
- Identificación de la necesidad de optimización del código en dispositivos de realidad virtual porque los recursos son más limitados en comparación con entornos de escritorio
- Identificación del impacto de la representación visual en la experiencia de usuario dentro de entornos inmersivos
- Validación del sistema completo en un entorno de realidad virtual, comprobando la correcta integración y funcionamiento conjunto de todos sus componentes

En conjunto, el desarrollo iterativo del proyecto ha permitido evolucionar desde una fase inicial de exploración tecnológica hasta la construcción de un sistema funcional completo, validado tanto en entornos de escritorio como de realidad virtual. La división en sprints ha facilitado la identificación progresiva de limitaciones, la toma de decisiones técnicas fundamentadas y la consolidación de una arquitectura robusta y adaptable.

Capítulo 4

Descripción del resultado

En este capítulo se describe el resultado del proyecto desde el punto de vista de su diseño e implementación. El sistema desarrollado no se plantea como una única aplicación monolítica, sino como un conjunto de componentes y herramientas modulares orientadas a la construcción de aplicaciones de realidad extendida (XR) mediante lenguaje natural.

Este enfoque permite construir una **caja de herramientas reutilizable**, cuyos componentes pueden combinarse para desarrollar distintas aplicaciones en función del contexto de uso. Cada uno de estos componentes encapsula una funcionalidad específica, como el reconocimiento de voz, el procesamiento del lenguaje natural o la generación de escenas tridimensionales.

Gracias a esta arquitectura basada en componentes desacoplados y comunicación mediante eventos, es posible construir distintas aplicaciones combinando los componentes disponibles, adaptándolos a diferentes contextos de uso, dispositivos y necesidades.

4.1. Funcionalidad de la caja de herramientas

El conjunto de herramientas desarrollado proporciona una serie de funcionalidades que permiten la creación y modificación de escenas tridimensionales mediante interacción en lenguaje natural. Cada funcionalidad está implementada por uno o varios componentes especializados, que se comunican entre sí mediante eventos.

En el Cuadro 4.1 se relacionan las funcionalidades principales del sistema con los componentes encargados de implementarlas.

Funcionalidad	**Componente/s**	**Descripción**
Lectura de teclado	`text-input`	Permite introducir instrucciones manualmente mediante texto.
Reconocimiento de voz	`voice-input`, `voice-input-speechapi`, `voice-input-vosk`, `voice-input-groq`	Capturan la voz del usuario y la transforman en texto mediante distintos motores de reconocimiento.
Redirección de comandos	`command-router`	Clasifica la instrucción del usuario como comando de ayuda, guardado o edición de escena.
Consulta LLM	`llm-client`	Envía la instrucción al modelo de lenguaje y obtiene una representación estructurada de la escena.
Gestión del estado	`scene-orchestrator`	Guarda el estado actual de la escena y coordina su actualización.
Renderizado 3D	`room-renderer`, `aframe-lounge`, `primitives-manager`, `objects-catalog`, `star-sky`	Generan visualmente la habitación, el entorno, las luces, los objetos y las primitivas.
Persistencia remota	`push-to-github`	Guarda el estado de la escena en un repositorio remoto.
Ayuda	`help-panel`, `error-panel`, `success-push-panel`, `error-push-panel`	Muestran mensajes de ayuda, error o confirmación dentro del entorno.

Cuadro 4.1: Relación entre funcionalidades y componentes del sistema

El sistema permite al usuario interactuar mediante comandos en lenguaje natural, introducidos por voz o texto según el entorno de ejecución. En navegador de escritorio se dispone de ambas opciones, mientras que en realidad virtual la interacción se realiza exclusivamente por voz, favoreciendo una experiencia más inmersiva.

4.1.1. Estructura de la funcionalidad

La funcionalidad del sistema se organiza como un conjunto de componentes desacoplados que se comunican entre sí mediante eventos. Esta estructura permite definir distintos flujos de ejecución en función de las necesidades de la aplicación, reutilizando únicamente los componentes necesarios en cada caso.

En la Figura 4.1 se muestra el sistema completo, donde se pueden observar todos los componentes conectados entre sí y el flujo de información desde la entrada del usuario hasta la generación y renderizado de la escena. Este diagrama representa el caso de uso más completo, en el que intervienen todos los módulos del sistema.

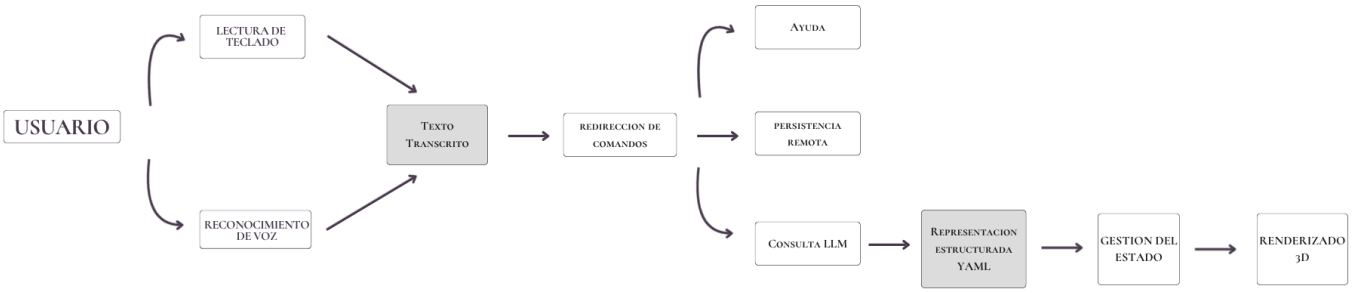


Figura 4.1: Estructura funcional del sistema con todos los componentes

No obstante, uno de los aspectos más relevantes del diseño propuesto es su flexibilidad, ya que los componentes pueden utilizarse de forma independiente para construir flujos más simples o adaptados a otros contextos. A continuación, se presentan algunos ejemplos representativos:

- **Flujo de reconocimiento de voz:** En este caso, el sistema se limita a capturar la voz del usuario mediante el micrófono y transformarla en texto. El flujo se compone de los componentes de entrada de reconocimiento de voz y el motor seleccionado. El resultado es una transcripción que puede ser utilizada por otros sistemas externos sin necesidad de incluir el resto del sistema.

Figura 4.2: Flujo de reconocimiento de voz

- **Flujo de consulta a un modelo de lenguaje (LLM):** En este otro caso, el sistema recibe una entrada textual y la envía directamente al componente de consulta al modelo de lenguaje. El resultado es una respuesta generada por el LLM, sin necesidad de realizar renderizado ni gestión de escenas. Este flujo es útil para aplicaciones centradas únicamente en procesamiento de lenguaje natural.

Figura 4.3: Flujo de consulta a un modelo de lenguaje LLM

Esta organización modular permite reutilizar los componentes en distintos contextos, facilitando la adaptación del sistema a nuevas aplicaciones sin necesidad de modificar el componente.

En la siguiente sección, se presenta una descripción detallada de cada uno de los componentes del sistema, incluyendo su funcionalidad, así como los eventos que emiten y reciben.

4.1.2. Descripción de los componentes

4.1.2.1. Componentes de lectura de teclado

- **text-input**: Permite introducir texto mediante el teclado y enviarlos al sistema mediante un evento. Su función principal es agilizar pruebas y solo está disponible en navegadores de escritorio.

Evento emitido	**Acción**
user-command	Envía el texto introducido por el usuario

Cuadro 4.2: Eventos emitidos del componente text-input

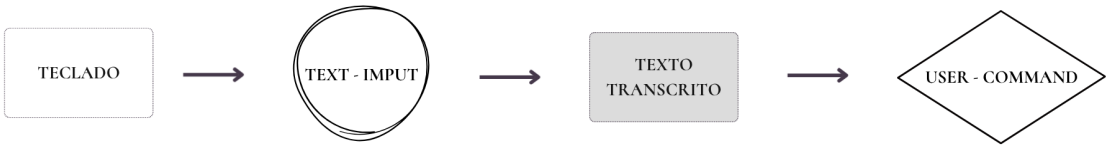


Figura 4.4: Diagrama del componente de lectura de teclado

4.1.2.2. Componentes de reconocimiento de voz

- **voice-input**: Este componente actúa como un configurador de entrada de voz, encargado de seleccionar el motor de reconocimiento utilizado. No realiza el procesamiento directamente, sino que delega la captura y transcripción en el componente correspondiente según la configuración establecida.

Parámetro	**Opciones**
engine	vosk — groq — speechapi

Cuadro 4.3: Parámetro configurable en el componente voice-input

Hay tres tipos de componentes disponibles para generar un texto a través de un audio:

- **voice-input-vosk**: Este componente implementa el reconocimiento de voz en el lado cliente mediante la librería Vosk en su versión para navegador, permitiendo realizar la transcripción de audio de forma local sin depender de servicios externos y emite un evento con ese texto transcrito.

Evento emitido	**Acción**
user-command	Envía el texto procesado

Cuadro 4.4: Eventos emitidos del componente voice-input-vosk

- **voice-input-groq**: Este componente implementa la transcripción de voz a texto mediante el uso del modelo Whisper a través de la API de Groq. Captura el audio desde el micrófono del usuario, lo envía a un servicio externo para su procesamiento y genera co-

mo resultado una transcripción que se integra en el sistema y emite un evento con ese texto transcrito.

Parámetro	**Descripción**	**Valor por defecto**
apiUrl	URL del endpoint de transcripción de Groq	`https://api.groq.com/openai/v1/audio/transcriptions`
apiKey	Clave de acceso utilizada para autenticar la petición contra la API de Groq	API KEY
model	Modelo de transcripción utilizado	whisper-large-v3

Cuadro 4.5: Parámetros configurables en el componente voice-input-groq

Evento emitido	**Acción**
user-command	Envía el texto procesado

Cuadro 4.6: Eventos emitidos del componente voice-input-groq

- **voice-input-speechapi**: Este componente implementa el reconocimiento de voz utilizando la Web Speech API nativa del navegador, basada en *SpeechRecognition*, permitiendo la transcripción directa del audio sin necesidad de servicios externos adicionales y emite un evento con ese texto transcrito.

Evento emitido	**Acción**
user-command	Envía el texto procesado

Cuadro 4.7: Eventos emitidos del componente voice-input-speechapi

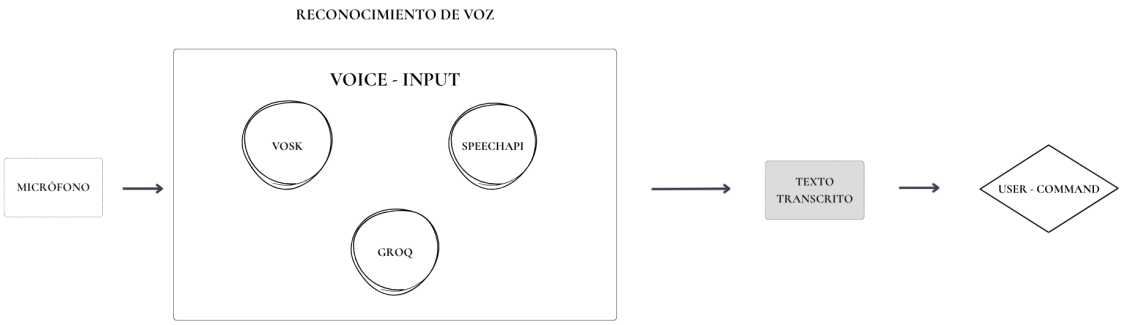


Figura 4.5: Diagrama de componentes de reconocimiento de voz

4.1.2.3. Componentes de redirección de comandos

- **command-router**: Este componente actúa como enrutador central de comandos dentro del sistema. Su función principal es recibir una instrucción del usuario, identificar si corresponde a una acción del sistema o a una modificación de la escena, y redirigirla al componente correspondiente mediante distintos eventos.

Parámetro	**Descripción**	**Valor por defecto**
inputEvent	Evento recibido con texto	`user-command`
helpEvent	Evento emitido al solicitar ayuda	`show-help-panel`
saveEvent	Evento emitido al guardar la escena	`save-scene-command`
editEvent	Evento emitido para modificar la escena	`scene-edit-command`
statusSelector	Selector del elemento de estado en la interfaz	`#status`

Cuadro 4.8: Parámetros configurables en el componente command-router

Evento recibido	**Acción**
user-command	texto generado por voz o texto

Cuadro 4.9: Evento recibido en el componente command-router

Evento emitido	**Acción**
show-help-panel	Ayuda
save-scene-command	Guardar escena
scene-edit-command	Editar escena

Cuadro 4.10: Eventos emitidos del componente command-router

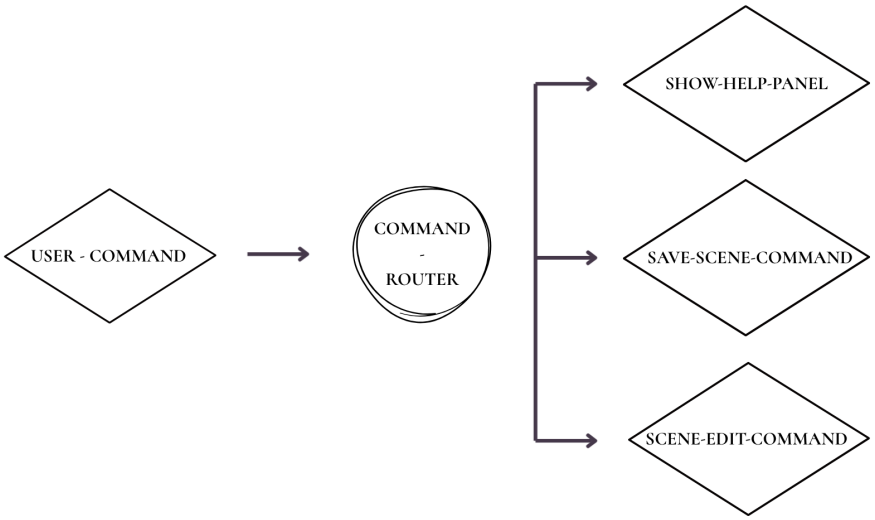


Figura 4.6: Diagrama del componente de redirección de comandos

4.1.2.4. Componentes consulta LLM

- **llm-client**: Este componente se encarga de transformar instrucciones en lenguaje natural en una representación estructurada de la escena en formato YAML, donde se reflejan las distintas propiedades de la escena generable. Para ello, recibe comandos de edición, construye un prompt que incluye el estado actual de la escena y un conjunto de reglas, y realiza una petición a un modelo de lenguaje (LLM) a través de OpenRouter. Finalmente, procesa la respuesta obtenida y emite un evento con la representación generada o, en caso de error, un evento indicando el fallo en la generación.

Parámetro	**Descripción**	**Valor por defecto**
`model`	Modelo de lenguaje utilizado para procesar la instrucción del usuario.	`openrouter/auto`
`url`	Endpoint de la API de OpenRouter al que se envía la petición.	`https://openrouter.ai/api/v1/chat/completions`
`openrouterApikey`	Clave de acceso utilizada para autenticar las peticiones a OpenRouter.	`API KEY`
`inputEvent`	Evento que recibe el componente con el comando de edición de escena.	`scene-edit-command`
`outputEvent`	Evento emitido cuando se genera correctamente la escena estructurada.	`yaml-generated`
`stateSelector`	Selector del componente que mantiene el estado actual de la escena.	`[scene-orchestrator]`
`statusSelector`	Selector del elemento HTML utilizado para mostrar el estado del sistema.	`#status`
`outputSelector`	Selector del elemento donde se muestra el YAML generado.	`#yamlOutput`

Cuadro 4.11: Parámetros configurables en el componente llm-client

Evento recibido	**Acción**
scene-edit-command	Modificación de la escena

Cuadro 4.12: Evento recibido en el componente llm-client

Evento emitido	**Acción**
yaml-generated	Nueva representación estructurada de la escena
show-error-panel	Instrucción no válida

Cuadro 4.13: Eventos emitidos del componente llm-client

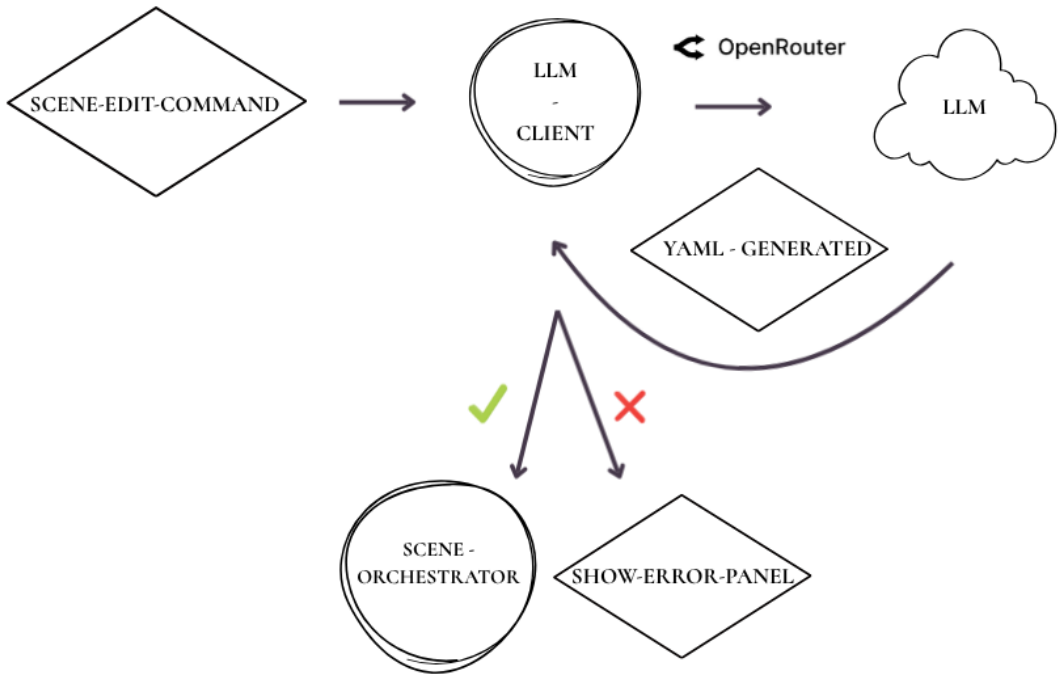


Figura 4.7: Diagrama del componente de consulta LLM

4.1.2.5. Componentes de gestión del estado

- **scene-orchestrator**: Componente encargado de almacenar y gestionar el estado de la escena en formato YAML, actuando como fuente de verdad del sistema. Cuando recibe una nueva representación mediante el evento `yaml-generated`, actualiza su estado interno.

Parámetro	Descripción	Valor por defecto
`initialRoom`	Estado inicial de la habitación. Si no se define, el componente utiliza una escena inicial por defecto.	`null`
`rendererSelector`	Selector del componente encargado de renderizar visualmente la escena.	`[room-renderer]`
`inputEvent`	Evento que recibe el componente cuando se genera una nueva escena estructurada.	`yaml-generated`

Cuadro 4.14: Parámetros configurables en el componente scene-orchestrator

Evento recibido	Acción
yaml-generated	Nueva representación estructurada de la escena

Cuadro 4.15: Evento recibido en el componente scene-orchestrator

4.1.2.6. Componentes renderizado 3D

- **room-renderer**: Componente encargado de transformar el estado estructurado de la escena en entidades visuales de A-Frame. A partir de la representación en formato YAML, construye dinámicamente el entorno tridimensional, incluyendo el cielo, las luces, la habitación y los objetos contenidos en ella.

Evento recibido	Acción
yaml-generated	Nueva representación estructurada de la escena

Cuadro 4.16: Evento recibido en el componente room-renderer

Parámetro	Descripción	Valor por defecto
`yaml`	Cadena de texto en formato YAML. En la implementación actual, el componente renderiza principalmente a partir del objeto recibido en el evento `yaml-generated`.	Cadena vacía

Cuadro 4.17: Parámetros configurables en el componente room-renderer

- **aframe-lounge**: Este componente define la estructura principal de la habitación donde se sitúan los elementos de la escena. A partir de parámetros como dimensiones, tipo de paredes, texturas, techo o punto de entrada, genera automáticamente el espacio tridimensional. Para ello, coordina varios subcomponentes especializados, lo que lo convierte en el bloque fundamental de la representación espacial.
- **primitives-manager**: Componente encargado de la creación y gestión de primitivas geométricas nativas de A-Frame dentro de la escena. Actúa como una capa de abstracción que per-

mite instanciar objetos simples a partir de una representación estructurada, garantizando el uso exclusivo de primitivas previamente definidas y permitidas.

- **objects-catalog**: Este componente define el conjunto de modelos 3D disponibles en el sistema mediante una estructura de datos que actúa como inventario de objetos utilizables en la escena, sirviendo como referencia para su instanciación.

- **star-sky**: Componente encargado de generar un fondo estrellado mediante partículas distribuidas aleatoriamente en el espacio, utilizado como elemento visual del entorno tridimensional.

Parámetro	**Descripción**	**Valor por defecto**
`color`	Color aplicado a las estrellas generadas.	`#FFF`
`radius`	Radio mínimo a partir del cual se distribuyen las estrellas alrededor de la escena.	`300`
`depth`	Profundidad adicional utilizada para distribuir las estrellas entre distintos radios.	`300`
`size`	Tamaño visual de cada estrella.	`1`
`count`	Número total de estrellas generadas.	`10000`
`texture`	Textura opcional aplicada a cada punto del sistema de estrellas.	Vacío

Cuadro 4.18: Parámetros configurables en el componente star-sky

4.1.2.7. Componentes de persistencia remota

- **push-to-github**: Este componente se encarga de la persistencia remota de la escena. Su función principal es obtener el estado actual de la escena desde el componente `scene-orchestrator` y almacenarlo automáticamente en un repositorio de GitHub mediante su API REST.

Parámetro	**Descripción**	**Valor por defecto**
`owner`	Propietario del repositorio de GitHub.	`propietario`
`repo`	Nombre del repositorio donde se almacena la escena.	`repositorio`
`filePath`	Ruta del archivo YAML dentro del repositorio.	`room.yaml`
`branch`	Rama del repositorio sobre la que se realiza la actualización.	`main`
`token`	Token de acceso personal utilizado para autenticar la petición contra GitHub.	Vacío
`inputEvent`	Evento que activa el proceso de guardado.	`save-scene-command`
`stateSelector`	Selector del componente que mantiene el estado actual de la escena.	`[scene-orchestrator]`
`statusSelector`	Selector del elemento HTML utilizado para mostrar el estado del proceso.	`#status`
`outputSelector`	Selector del elemento donde se muestra el YAML generado.	`#yamlOutput`

Cuadro 4.19: Parámetros configurables en el componente push-to-github

Evento recibido	**Acción**
save-scene-command	Guardar la escena

Cuadro 4.20: Evento recibido en el componente push-to-github

Evento emitido	**Acción**
show-success-push-panel	Escenario guardado correctamente en GitHub
show-error-push-panel	Error durante el proceso de guardado en GitHub

Cuadro 4.21: Eventos emitidos del componente push-to-github

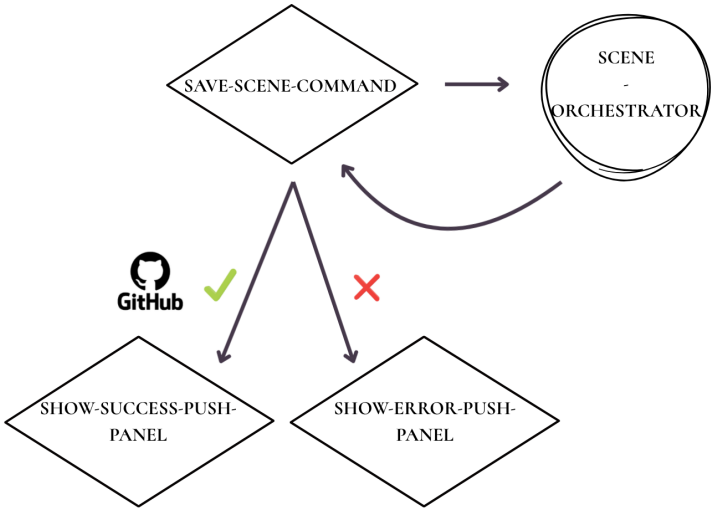


Figura 4.8: Diagrama del componente de persistencia remota

4.1.2.8. Componentes de ayuda

- **help-panel**: Este componente se encarga de mostrar un panel informativo dentro del entorno de realidad virtual que proporciona al usuario instrucciones de uso y ejemplos de comandos válidos durante un tiempo limitado (15 segundos). Este panel actúa como un sistema de ayuda contextual accesible mediante comandos de voz, mejorando la usabilidad del sistema sin necesidad de abandonar la experiencia inmersiva.

Evento recibido	**Acción**
show-help-panel	Panel de ayuda

Cuadro 4.22: Evento recibido en el componente help-panel

- **error-panel**: Este componente se encarga de mostrar mensajes de error dentro del entorno de realidad virtual de forma visual y contextual. Su objetivo es informar al usuario cuando una instrucción no puede ser interpretada o procesada correctamente, proporcionando retroalimentación inmediata sin salir de la experiencia inmersiva durante un tiempo limitado (5 segundos).

Evento recibido	Acción
show-error-panel	Panel de error

Cuadro 4.23: Evento recibido en el componente error-panel

- **error-push-panel**: Este componente se encarga de mostrar mensajes de error específicos relacionados con el proceso de guardado del escenario en GitHub. Su finalidad es proporcionar al usuario una retroalimentación clara cuando ocurre un fallo durante la persistencia remota, diferenciando este tipo de errores de los errores generales del sistema.

Evento recibido	Acción
show-error-push-panel	Panel de error al conectarse a GitHub

Cuadro 4.24: Evento recibido en el componente error-push-panel

- **success-push-panel**: Este componente se encarga de mostrar un mensaje de confirmación al usuario cuando el escenario se ha guardado correctamente en GitHub. Su objetivo es proporcionar retroalimentación positiva tras una operación de persistencia exitosa, reforzando la comprensión del estado del sistema dentro del entorno inmersivo.

Evento recibido	Acción
show-success-push-panel	Push realizado correctamente en GitHub

Cuadro 4.25: Evento recibido en el componente success-push-panel

4.1.3. Implementación de los componentes principales

- **voice-input-vosk**: Este componente implementa reconocimiento de voz completamente en el cliente mediante la librería Vosk, que ejecuta modelos de reconocimiento en el navegador utilizando WebAssembly. Este enfoque permite realizar la transcripción de audio sin necesidad de enviar datos a servicios externos, reduciendo la latencia y mejorando la privacidad.

Durante la fase de inicialización, el componente carga un modelo acústico preentrenado desde una ruta especificada mediante el método `loadVosk`. El modelo utilizado es `vosk-model-small-es-0.42`, un modelo acústico en español preparado para reconocimiento de voz. Una vez cargado mediante `Vosk.createModel()`, el sistema queda listo para iniciar la transcripción. Este modelo contiene la información necesaria para mapear señales de audio a secuencias de texto, incluyendo modelos fonéticos y lingüísticos.

El acceso al micrófono se realiza mediante la API *navigator.mediaDevices.getUserMedia* tras la autorización del usuario. El flujo de audio capturado se conecta a un contexto de audio (*AudioContext*) a través de un nodo *MediaStreamSource*. Posteriormente, se instancia un reconocedor de Vosk mediante *new model.KaldiRecognizer(audioCtx.sampleRate)* utilizando la frecuencia de muestreo del contexto de audio.

Para el procesamiento en tiempo real, el sistema utiliza un nodo *ScriptProcessorNode* con un tamaño de buffer de 2048 muestras, que permite acceder a los buffers de audio entrantes. En cada llamada a `onaudioprocess`, el fragmento de audio recibido se envía al reconocedor mediante el método `acceptWaveform()`.

El sistema gestiona dos tipos de resultados:

- **Resultados parciales**: proporcionan una transcripción provisional del texto reconocido mientras el usuario está hablando.
- **Resultados finales**: representan segmentos de audio completos ya procesados.

El componente acumula los resultados finales en una cadena de texto, evitando duplicados mediante la comparación con el último resultado recibido. Esto permite construir progresivamente la transcripción completa de la intervención del usuario. Cuando el usuario detiene la captura, el componente construye el texto final combinando el texto acumulado y, si existe, el último resultado parcial. Después desconecta el procesador, detiene las pistas del micrófono, cierra el `AudioContext` y reinicia las variables internas para dejar el sistema preparado para una nueva captura. Finalmente, si se ha reconocido texto válido, se emite el evento *user-command* incluyendo el texto final.

- **voice-input-groq**: Este componente implementa un enfoque de reconocimiento de voz basado en inferencia remota. A diferencia de soluciones locales, el procesamiento de la señal de audio se realiza en un servidor externo que ejecuta el modelo Whisper, lo que permite aprovechar modelos de mayor tamaño y precisión.

  El método `startRecording()` comprueba primero si el navegador soporta la API `navigator.mediaDevices.getUserMedia`. Si no está disponible, se informa al usuario de que el micrófono no está soportado. En caso contrario, se solicita acceso al micrófono.

  Una vez concedido el acceso, el flujo de audio se almacena en `stream` y se crea una instancia de `MediaRecorder`. Este objeto permite grabar el audio capturado por el navegador y generar fragmentos binarios de audio.

  Durante la grabación, el evento `ondataavailable` recoge los fragmentos de audio generados por `MediaRecorder` y los almacena en el array `audioChunks`. Estos fragmentos se mantienen en memoria hasta que el usuario detiene la grabación.

  Cuando el proceso se marca como detenido, los fragmentos acumulados se agrupan en un único objeto `Blob` con tipo `audio/webm`:

  Este archivo de audio se envía al método `enviarAGroq()`, encargado de realizar la petición HTTP al servicio de transcripción con el archivo de audio y el modelo Whisper. La petición se realiza mediante `fetch`, utilizando el método `POST` y añadiendo la cabecera de autorización con el formato. El cuerpo de la petición (`body`) se construye utilizando un objeto `FormData`, que permite enviar datos en formato multipart. En este caso, se incluyen dos campos: el archivo de audio capturado (`audioBlob`), que se envía como fichero binario, y el modelo de transcripción a utilizar. Este formato es requerido por la API para procesar correctamente el audio enviado.

```
fetch(https://api.groq.com/openai/v1/audio/transcriptions, {
    method: "POST",
    headers: {
      Authorization: `Bearer <apiKey>`
    },
    body: formData
  });
```

Si la respuesta de la API es correcta, el componente interpreta el cuerpo de la respuesta como JSON y extrae el campo `text`, que contiene la transcripción generada por Whisper. Cuando se obtiene una transcripción válida, el componente actualiza el estado visual del sistema y emite el evento `user-command` Finalmente, el componente libera los recursos utilizados deteniendo las pistas del micrófono, vaciando los fragmentos de audio almacenados y reiniciando el estado interno.

- **voice-input-speechapi**: Este componente utiliza la interfaz `SpeechRecognition` del navegador. Para mejorar la compatibilidad con navegadores basados en Chromium, se contempla también la implementación `webkitSpeechRecognition`. Cuando la API está disponible, se crea una instancia del reconocedor:

```
const recognition = new SpeechRecognition();
```

La captura de audio es gestionada internamente por el navegador. A diferencia de otros enfoques como `MediaRecorder` o `AudioContext`, este componente no accede directamente al flujo de audio ni procesa manualmente los buffers. El navegador se encarga de capturar la señal del micrófono, procesarla mediante su motor de reconocimiento y devolver eventos con las hipótesis de transcripción. Cuando ocurre el evento `onresult`, el componente recorre los resultados recibidos desde `event.resultIndex` hasta el final del array `event.results`, concatenando las transcripciones generadas por el motor. Para detectar cuándo el usuario ha terminado de hablar, el componente implementa un mecanismo de detección de silencio basado en temporizador o pulsando el botón `stop` . Cada vez que se recibe un nuevo resultado, se reinicia el temporizador anterior. Si transcurren 2000 milisegundos sin nuevos resultados, se considera que la intervención ha finalizado. Cuando se detecta este periodo de silencio, el componente detiene el reconocimiento mediante `recognition.stop()`. Si existe texto acumulado, se emite el evento `user-command`.

- **command-router**: Durante la inicialización, el componente obtiene una referencia al elemento de estado indicado mediante el parámetro `statusSelector`. Posteriormente, registra un manejador de eventos sobre la escena de A-Frame para recibir el evento definido en `inputEvent`. Cuando se recibe un comando, el componente extrae el texto del

campo `e.detail.text`. Si el texto está vacío o no existe, la ejecución se interrumpe. En caso contrario, el texto se normaliza antes de ser analizado. Esta normalización permite comparar comandos de forma más robusta, evitando que diferencias como mayúsculas, minúsculas o acentos afecten a la detección de instrucciones. El componente distingue tres tipos de comandos:

- **Comandos de ayuda**: expresiones como `que puedo hacer` o `muestra un panel de ayuda`. En este caso, se actualiza el estado del sistema y se emite el evento `show-help-panel`.
- **Comandos de guardado**: expresiones como `guarda este escenario`, `guardar este escenario`, `guarda el escenario` o `guardar el escenario`. En este caso, se actualiza el estado y se emite el evento `save-scene-command`.
- **Comandos de edición**: cualquier otro comando válido se considera una instrucción destinada a crear o modificar la escena, por lo que se emite el evento `scene-edit-command`.

- **llm-client**: Este componente es el encargado de transformar instrucciones en lenguaje natural en una representación estructurada de la escena en formato YAML mediante el uso de un modelo de lenguaje (LLM).

  Durante la inicialización, configura la conexión con OpenRouter, obtiene referencias al estado de la escena y registra un manejador para recibir comandos de edición. Cuando se recibe una instrucción, el texto es procesado aplicando un mecanismo de *debounce* para evitar peticiones redundantes y se descartan posibles comandos duplicados en un intervalo corto de tiempo.

  Antes de realizar la consulta al modelo, el componente obtiene el estado actual de la escena desde `scene-orchestrator`. Este estado se incorpora al prompt con el objetivo de permitir modificaciones incrementales sin reconstruir la escena completa.

  El prompt se construye dinámicamente mediante un conjunto de reglas que definen el comportamiento del sistema, incluyendo restricciones espaciales, catálogo de objetos disponibles, primitivas permitidas y formato de salida requerido en YAML. Estas reglas pueden adaptarse automáticamente en función de las capacidades activas del sistema.

Una vez construido el prompt, el componente realiza una petición HTTP de tipo `POST` al endpoint configurado de OpenRouter. Esta petición se realiza mediante `fetch` y contiene dos partes principales: las cabeceras de autenticación y el cuerpo de la petición.

En las cabeceras se incluye el token de acceso mediante el campo `Authorization`, junto con el tipo de contenido `application/json`. En el cuerpo de la petición se envía un objeto JSON que contiene el modelo seleccionado y el mensaje que se quiere procesar:

- `model`: identifica el modelo de lenguaje que se va a utilizar.
- `messages`: contiene el prompt generado, enviado como mensaje de usuario.

De forma simplificada, la petición sigue la siguiente estructura:

```
fetch(https://openrouter.ai/api/v1/chat/completions, {
  method: "POST",
  headers: {
    Authorization: "Bearer <API_KEY>",
    "Content-Type": "application/json"
  },
  body: JSON.stringify({
    model: MODEL,
    messages: [
      { role: "user", content: prompt }
    ]
  })
})
```

La respuesta del modelo es procesada y normalizada para garantizar su correcto formato en YAML. En caso de error en la interpretación, se notifica enviando un mensaje de error. Si la respuesta es válida, se parsea y se combina con el estado anterior para preservar información no modificada.

Finalmente, el componente emite el evento `yaml-generated` con la nueva representación estructurada de la escena, permitiendo que otros componentes actualicen el estado global y desencadenen el renderizado correspondiente.

- **scene-orchestrator**: Durante la inicialización, el componente obtiene una referencia al elemento encargado del renderizado mediante el parámetro `rendererSelector`. En la configuración principal del proyecto, este selector apunta al elemento `#roomScene`, que contiene el componente `room-renderer`.

  A continuación, el componente inicializa la variable `currentRoom`, que representa el estado actual de la escena. Si se proporciona un valor en `initialRoom`, se utiliza como estado inicial. En caso contrario, se crea una habitación por defecto con dimensiones, paredes, texturas, punto de entrada, entorno, luces y una lista inicial de objetos vacía. Estado inicial de la escena:

```
room: {
    width: 200,
    depth: 300,
    height: 100,
    ceiling: true,
    walls: {
      north: "barrier",
      east: "glass",
      south: "wall",
      west: "wall"
    },
    textures: {
      floor: "../assets/floor-texture.jpg",
      wall: "../assets/jeroglifico.jpg",
      ceiling: "../assets/jeroglifico.jpg"
    },
    entryPoint: { x: -10, y: 2, z: 20 },
    environment: { skyColor: "#000000", stars: true },
    lights: [
      { type: "ambient", color: "#ffffff", intensity: 0.85 },
      { type: "ambient", color: "#ebd9e9", intensity: 0.3 }
    ],
    objects: []
  }
```

  El componente registra un manejador asociado al evento definido en `inputEvent`.

Cuando se recibe este evento, se extrae el nuevo estado de `e.detail.room`. Si el evento no contiene una habitación válida, no se realiza ninguna acción.

Cuando el estado recibido es válido, se actualiza `currentRoom` utilizando `structuredClone()`. Esta función permite crear una copia profunda del objeto recibido, evitando referencias compartidas entre componentes y reduciendo posibles efectos secundarios durante la manipulación del estado.

Si el parámetro `autoRender` está activado, el componente invoca el método `renderCurrentRoom()`. Este método localiza el componente `room-renderer` asociado al elemento definido por `rendererSelector` y ejecuta su método `renderRoom()`, pasando como argumento el estado actual de la escena.

De este modo, `scene-orchestrator` no se encarga directamente de crear entidades 3D, sino que delega esta responsabilidad en `room-renderer`. Esta separación permite mantener diferenciadas la gestión del estado y la representación visual.

Además, durante la inicialización, si `autoRender` está activo, el componente renderiza automáticamente la habitación inicial. Esto permite que la escena tenga un estado visible desde el primer momento, incluso antes de recibir comandos del usuario.

- **room-renderer**: El componente registra un manejador asociado al evento `yaml-generated`. Cuando recibe este evento, extrae el objeto `room` contenido en `e.detail.room` y llama al método `renderRoom()`, que es el encargado de reconstruir la escena. El método `renderRoom()` realiza primero varias comprobaciones de validez. Si no existe información de habitación, o si faltan propiedades obligatorias como `width`, `depth`, `height` o `walls`, el renderizado se cancela. Esto evita crear escenas incompletas o inconsistentes.

  Antes de construir la nueva escena, el componente elimina todo el contenido previo del elemento raíz mediante `this.root.innerHTML = `. De esta forma, cada nueva versión de la escena se renderiza desde cero a partir del estado actualizado. Posteriormente, se ejecutan las siguientes fases:

  - Renderizado del entorno visual.
  - Renderizado de luces.

  - Construcción de la habitación principal.
  - Sincronización de la posición del usuario.
  - Renderizado de objetos dentro de la habitación.

- **primitives-manager**: El sistema define un conjunto cerrado de primitivas válidas mediante la constante `ALLOWED_PRIMITIVES`, que incluye elementos como `box`, `sphere`, `cylinder`, `cone`, `plane`, `circle` y `torus`. Esto permite controlar qué tipos de geometría pueden ser generados por el sistema.

- **objects-catalog**: Cada entrada del catálogo está identificada por una clave única (como `chair_basic` o `statue_venus`) e incluye la información necesaria para su instanciación, como la ruta al archivo en formato GLB (`src`), el tipo de objeto, un desplazamiento vertical respecto al suelo (`floorOffset`) y una escala base (`scale`). Este catálogo es utilizado por el componente `room-renderer` para crear los objetos dentro de la escena y por el componente `llm-client` para limitar el conjunto de modelos que el sistema puede generar, garantizando así la coherencia entre la descripción estructurada y los recursos disponibles, además de facilitar la extensibilidad del sistema mediante la incorporación de nuevos modelos sin necesidad de modificar la lógica principal.

- **aframe-lounge**: Se apoya en varios subcomponentes especializados:

  - lounge-floor: genera el suelo
  - lounge-ceiling: genera el techo
  - lounge-wall: construye las paredes
  - lounge-entry-point: define el punto de entrada del usuario
  - lounge-staydown: ajusta la posición de los objetos al suelo
  - lounge-plinth: crea pedestales
  - lounge-collider: gestiona colisiones entre objetos

- **push-to-github**: El componente escucha el evento de guardado `save.scene-command` y, al recibirlo, obtiene el estado actual de la escena desde `scene-orchestrator`. Este estado es transformado a formato YAML y se realiza una normalización de rutas para garantizar la compatibilidad de los recursos en el entorno del repositorio.

Una vez preparado el contenido, se construye una petición HTTP contra la API REST de GitHub. El contenido YAML se codifica en Base64 mediante `btoa()`, ya que la API requiere que los archivos se envíen en este formato, y se envía junto con la información del repositorio, la ruta del archivo y un mensaje de commit. La petición se realiza sobre el endpoint:

`https://api.github.com/repos/{owner}/{repo}/contents/{filePath}`

En las cabeceras se incluye el token de autenticación en formato `Bearer`, mientras que en el cuerpo de la petición se envía un objeto JSON que contiene:

```
fetch("https://api.github.com/repos/OWNER/REPO/contents/room.yaml?ref=BRANCH",
  method: "GET",
  headers: {
    "Authorization": "Bearer <TOKEN>",
    "Accept": "application/vnd.github+json",
    "User-Agent": "XR-Room-Saver"
  }
})
})
```

Antes de realizar la subida, el sistema comprueba si el archivo ya existe mediante una petición `GET`. En caso afirmativo, se obtiene su identificador `sha`, necesario para actualizar el contenido existente. Si no existe, el archivo se crea directamente mediante una petición `PUT`.

```
fetch("https://api.github.com/repos/OWNER/REPO/contents/room.yaml", {
  method: "PUT",
  headers: {
    "Authorization": "Bearer <TOKEN>",
    "Content-Type": "application/json",
    "User-Agent": "XR-Room-Saver"
  },
  body: JSON.stringify({
    message: "Push automático: Guardar escenario XR (${new Date().toISOString()
    content: btoa(yamlContent),
    branch: "main",
    sha: existingSha // solo si el archivo ya existe
```

```
    })
  })
  .then(response => response.json())
```

- `message`: mensaje del commit que describe la operación realizada.
- `content`: contenido del archivo codificado en Base64.
- `branch`: rama del repositorio sobre la que se realiza la operación.
- `sha`: identificador del archivo, únicamente en caso de actualización.

Si la operación se completa correctamente, el componente notifica el éxito al usuario y, en caso contrario, gestiona los errores proporcionando retroalimentación visual.

Además, el guardado de la escena activa un flujo de integración continua mediante GitHub Actions, que despliega automáticamente el contenido utilizando GitHub Pages. De este modo, cada escena almacenada queda publicada y accesible a través de una URL.

4.1.4. Entorno de ejecución

Los entornos de ejecución disponibles para la aplicación son:

1. **Navegador de escritorio (PC)**

   La aplicación presenta una interfaz híbrida que combina un entorno tridimensional inmersivo con una capa de interfaz gráfica tradicional superpuesta (overlay). Esta interfaz se muestra sobre la escena 3D y permite al usuario interactuar con el sistema mediante elementos convencionales como botones, campos de texto y el micrófono del ordenador.

   La interfaz inicial al acceder a la escena incluye controles básicos para iniciar y detener la captura de voz, así como un campo de entrada de texto que permite introducir comandos manualmente. Además, se muestra información de estado en tiempo real, indicando al usuario si el sistema se encuentra escuchando, procesando o en reposo. Como complemento, se incluye un área de visualización donde se presenta la representación intermedia generada por el sistema, lo que facilita la comprensión del resultado de las instrucciones proporcionadas.

La interacción en este entorno se realiza principalmente mediante el uso de ratón y teclado. El usuario puede activar el reconocimiento de voz mediante los botones Hablar y Parar, o introducir directamente comandos en formato textual, lo que resulta especialmente útil durante las fases de prueba y depuración del sistema.

2. **Dispositivos de realidad virtual (VR)**

   En el entorno de realidad virtual, la aplicación se ejecuta directamente desde el navegador del dispositivo, permitiendo al usuario acceder a una experiencia completamente inmersiva en tres dimensiones. A diferencia del modo de escritorio, la interacción se adapta a las capacidades propias de este tipo de dispositivos, priorizando el uso de comandos de voz y controladores.

   Para iniciar el reconocimiento de voz, el usuario debe mantener pulsado el botón trasero del mando derecho. Mientras dicho botón permanece presionado, el sistema activa la captura de audio mediante los componentes de entrada de voz.

   Al soltar el botón, se detiene la captura y el audio se procesa para obtener una transcripción en texto.

   Este mecanismo de interacción, basado en el paradigma push-to-talk, permite controlar de forma precisa cuándo se inicia y finaliza la captura de voz, evitando activaciones no deseadas y mejorando la experiencia de uso en entornos inmersivos.

   Además, la aplicación proporciona retroalimentación visual dentro del propio entorno virtual mediante un panel informativo (HUD), que indica el estado del sistema (escuchando, procesando, etc.). Esto resulta fundamental en ausencia de interfaces tradicionales dado que permite al usuario conocer en todo momento el estado de la interacción.

3. **Dispositivos móviles**

   La aplicación también puede ejecutarse en dispositivos móviles, aprovechando las capacidades del navegador para ofrecer una versión accesible del sistema. Aunque la experiencia puede estar limitada por el tamaño de pantalla y los recursos del dispositivo, este entorno permite una gran portabilidad y facilita el acceso a la aplicación sin necesidad de hardware especializado.

4.2. Ejemplo de uso: creación de un bosque

Como demostración de las capacidades de la caja de herramientas desarrollada, se ha implementado un sistema que integra todos los componentes del sistema.

Vamos a realizar un ejemplo concreto de uso en el cual vamos a crear un bosque con piedras y un río. Para ello, el usuario realiza los siguientes pasos:

1. El usuario entra en la escena

Figura 4.9: Escena inicial

2. Mantiene pulsado el botón trasero del mando derecho y pronuncia mediante voz “Crea un bosque con varios árboles distribuidos por la escena, añade rocas y un rio”. Vemos el resultado generado:

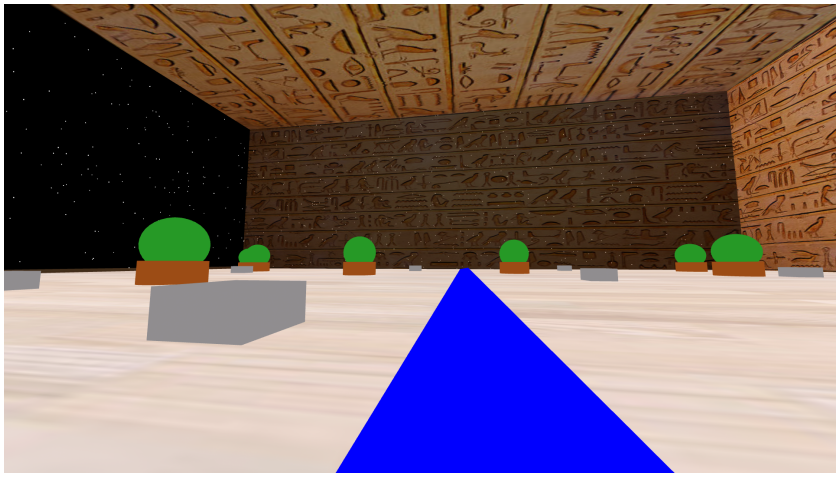


Figura 4.10: Escena final

En este ejemplo se puede observar la manera en la que podemos construir escenas mediante comandos de voz.

4.3. Ejemplo de reutilización: asistente conversacional basado en LLM

Un ejemplo de reutilización de los componentes desarrollados sería la creación de un asistente conversacional basado en texto. Únicamente se necesitaría el componente `text-input` y `llm-client`.

El flujo de funcionamiento sería el siguiente:

1. El usuario introduce una pregunta o instrucción mediante un campo de texto.
2. El componente de entrada genera un evento con el texto introducido.
3. El componente encargado de consultar al modelo de lenguaje recibe la entrada y realiza la petición al LLM.
4. El sistema procesa la respuesta obtenida y la muestra al usuario en la interfaz.

En este caso, el componente `llm-client` podría reutilizarse modificando el prompt y el formato de salida esperado. En lugar de solicitar una representación estructurada en YAML para generar una escena, el modelo recibiría instrucciones orientadas a responder preguntas, resumir información o mantener una conversación con el usuario.

Este ejemplo demuestra que la arquitectura desarrollada no está limitada a la generación de escenas tridimensionales, sino que permite reutilizar componentes concretos para construir aplicaciones diferentes, como asistentes conversacionales, sistemas de ayuda contextual o interfaces de consulta basadas en lenguaje natural.

Capítulo 5

Experimentos y validación

Se realizaron una serie de experimentos para validar la funcionalidad del sistema desarrollado. Para ello, se tuvieron en cuenta tanto la experiencia del usuario como el correcto funcionamiento del sistema mediante los comandos de voz. Estas pruebas se han realizado en un espacio cerrado, sin ruido ambiente y con todo lo necesario para poder ejecutar la aplicación.

5.1. Objetivos del experimento

El objetivo principal de este experimento es evaluar los siguientes aspectos:

- Precisión del reconocimiento de voz
- Evaluar el tiempo de respuesta del sistema desde que el usuario emite una instrucción por voz hasta que la escena es generada o modificada
- Experiencia del usuario
- Retroalimentación de los usuarios
- Identificar posibles problemas o limitaciones del sistema, tanto a nivel técnico como de diseño de interacción

Este conjunto de objetivos permite analizar el sistema desde una perspectiva tanto cuantitativa (tiempos, precisión) como cualitativa (experiencia de usuario), proporcionando una visión completa de su comportamiento.

5.2. Diseño del experimento

El experimento se ha diseñado con el objetivo de evaluar el sistema en condiciones cercanas a su uso real. Para ello, se ha optado por realizar las pruebas en un entorno de realidad virtual utilizando gafas Meta Quest 2, permitiendo así una interacción completamente inmersiva.

Se ha seleccionado esta configuración debido a que representa el escenario más exigente desde el punto de vista técnico y el más relevante en términos de experiencia de usuario.

Además, se ha utilizado el sistema de reconocimiento de voz basado en *vosk-browser*, ya que permite ejecutar el reconocimiento de forma local en el navegador, evitando dependencias de servicios externos y garantizando un comportamiento más estable y reproducible durante las pruebas.

5.2.1. Condiciones del experimento

Las pruebas se han realizado bajo las siguientes condiciones:

- Ejecución del sistema en entorno de realidad virtual (Meta Quest 2).
- Uso exclusivo de interacción por voz mediante el mecanismo *push-to-talk*.
- Todos los componentes del sistema activos (reconocimiento de voz, LLM, renderizado y persistencia).

5.2.2. Metodología

Cada usuario ha realizado una serie de tareas predefinidas con el objetivo de evaluar de forma controlada las funcionalidades principales del sistema. Para ello, previamente se le ha explicado de manera sencilla el funcionamiento del sistema y los objetos disponibles en la escena. Las tareas que se han propuesto a cada uno de los usuarios son:

- **Crear habitación**: El usuario genera una habitación de dimensiones 30x30x30.
- **Insertar objetos**: Debe añadir dos objetos en la escena.
- **Modificar propiedades**: Cambia dos características de los objetos que ha introducido en la escena.

- **Modificar elementos**: Ajusta el color del cielo, activa o no estrellas y cambia el tipo de paredes.

- **Guardar escenario**: Ejecuta el comando de guardado y verifica la escena final publicada en GitHub Pages.

Una vez completadas estas tareas guiadas, se ha permitido a los usuarios interactuar libremente con el sistema, con el objetivo de observar su comportamiento en un contexto no dirigido y evaluar la usabilidad de la herramienta en escenarios reales de uso.

Durante la realización de las pruebas se han recogido:

- Tiempos de respuesta del sistema.

- Errores de reconocimiento de voz.

- Observaciones sobre la interacción del usuario.

- Comentarios y valoraciones subjetivas.

Además, se ha realizado un seguimiento directo de los usuarios durante la prueba, registrando sus impresiones y dificultades desde el momento en que comienzan a utilizar el sistema hasta la finalización de la sesión.

5.3. Usuarios para realizar la prueba

Usuario	Perfil técnico	Experiencia en VR	Uso habitual de dispositivos digitales
Usuario 1	Sí	Sí	Sí
Usuario 2	No	No	Sí
Usuario 3	Sí	No	Sí
Usuario 4	No	No	Sí
Usuario 5	No	No	No

Cuadro 5.1: Carcaterísticas de los usuarios

5.4. Resultados

- **Precisión del reconocimiento de voz**: El sistema presenta una alta precisión en condiciones controladas, especialmente en usuarios con mayor claridad en la emisión de comandos. Sin embargo, se detectan errores en usuarios con menor experiencia o en casos donde las instrucciones son más complejas o menos estructuradas. Estos resultados indican que el sistema es robusto en escenarios habituales, aunque puede verse afectado por factores como la pronunciación o la complejidad del lenguaje utilizado.

Usuario	Comandos con éxito	Precisión (%)	Observaciones
Usuario 1	5/5	100 %	Sin errores, comandos claros
Usuario 2	4/5	80 %	Fallo en frase larga
Usuario 3	5/5	100 %	Buen reconocimiento general
Usuario 4	4/5	80 %	Problemas de pronunciación
Usuario 5	3/5	60 %	Dificultades iniciales con comandos

Cuadro 5.2: Número de comandos con éxito por usuario

- **Tiempo por tarea**:

Usuario	Crear habitación	Insertar objetos	Modificar propiedades	Modificar elementos	Guardar escenario
Usuario 1	10 s	17 s	7 s	13 s	2 s
Usuario 2	13 s	15 s	8 s	12 s	2 s
Usuario 3	11 s	13 s	6 s	16 s	3 s
Usuario 4	12 s	16 s	7 s	14 s	2 s
Usuario 5	15 s	20 s	9 s	17 s	3 s

Cuadro 5.3: Tiempo de respuesta de cada tarea por usuario

- **Grado de satisfacción de los usuarios**:

Usuario	Facilidad de uso	Precisión percibida	Rapidez	Interfaz visual	Satisfacción global
Usuario 1	4	5	4	4	4
Usuario 2	4	3	3	4	4
Usuario 3	5	5	4	5	5
Usuario 4	4	4	4	4	4
Usuario 5	3	3	3	4	3

Cuadro 5.4: Evaluación de la satisfacción de los usuarios (escala de 1 a 5)

- **Opiniones de los usuarios**:

Usuario	Observaciones
Usuario 1	Funcionamiento sencillo para modificar las escenas mediante voz y sistema muy flexible
Usuario 2	Sistema intuitivo aunque tuvo que repetir varias veces ciertos comandos
Usuario 3	Destaca la expetiencia inmersiva y la rapidez de la generación de escenas
Usuario 4	Facilidad de uso del sistema y sugiere la mejora de la precisión de los comandos de voz
Usuario 5	Muestra dificultades para hacer uso del sistema, aunque tras varios intentos logró generar las escenas

Cuadro 5.5: Opiniones de los usuarios

- **Problemas detectados**:

  - Errores puntuales en el reconocimiento de voz, especialmente en frases largas o poco estructuradas.
  - Necesidad de repetir algunos comandos debido a ambigüedades en la interpretación, como por ejemplo, a la hora de intentar guardar el escenario.
  - Tiempo de respuesta ligeramente elevado en tareas complejas.
  - Problemas menores en la visualización de algunos paneles en el entorno VR, que fueron corregidos posteriormente.

5.5. Conclusión del experimento

Los resultados obtenidos en el experimento permiten concluir que el sistema desarrollado cumple de manera satisfactoria los objetivos planteados, demostrando su viabilidad como herramienta para la creación y modificación de entornos tridimensionales mediante lenguaje natural.

En términos de precisión, el sistema ha mostrado un comportamiento robusto en el reconocimiento de voz en condiciones controladas, alcanzando valores elevados en la mayoría de los casos. No obstante, se han identificado limitaciones en situaciones donde las instrucciones son más complejas o en usuarios con menor experiencia, lo que sugiere la necesidad de mejorar la tolerancia a variaciones en el lenguaje natural.

En cuanto al tiempo de respuesta, los resultados indican que el sistema ofrece una interacción suficientemente ágil para su uso en tiempo casi real. Aunque las tareas más complejas presentan tiempos ligeramente superiores debido al procesamiento del modelo de lenguaje, estos se mantienen dentro de rangos aceptables para una experiencia inmersiva.

Desde la perspectiva de la experiencia de usuario, los participantes han valorado positivamente la facilidad de uso del sistema y su carácter intuitivo, destacando especialmente la capacidad de generar contenido sin necesidad de conocimientos técnicos. La integración en un entorno de realidad virtual ha contribuido significativamente a mejorar la percepción de inmersión y naturalidad en la interacción.

Por otro lado, el experimento ha permitido identificar diversas áreas de mejora, entre las que destacan la optimización del reconocimiento de voz en casos ambiguos, la reducción del tiempo de respuesta en determinadas operaciones y la ampliación del catálogo de elementos disponibles. Asimismo, se han detectado pequeños problemas en la interfaz que han sido corregidos durante el desarrollo.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

El objetivo general de este Trabajo Fin de Grado consistía en investigar nuevas formas de interacción con entornos tridimensionales mediante lenguaje natural, con especial énfasis en la voz, dentro del contexto de la realidad extendida.

Este objetivo se considera cumplido, ya que se ha desarrollado un sistema completo que permite al usuario interactuar con una escena 3D mediante comandos en lenguaje natural a través de una caja de herramientas basada en componentes desacoplados que se comunican mediante eventos.

Con el fin de alcanzar el objetivo principal, se han cumplido los siguientes objetivos específicos:

- **Experimentar con distintas tecnologías web orientadas al desarrollo de experiencias inmersivas**: Se ha usado el framework A-Frame, que permite la creación de escenas 3D inmersivas. Durante el desarrollo se han realizado pruebas tanto en navegador de escritorio como en gafas Meta Quest 2, evaluando sus limitaciones y compatibilidades.

- **Diseñar y desarrollar un conjunto de herramientas que permita la creación de escenas en realidad extendida.**: Se ha desarrollado un conjunto de herramientas a través de componentes de A-Frame en el que a través de un archivo estructurado podemos ver la escena.

- **Analizar e integrar distintas soluciones de reconocimiento de voz**: Se han implementado tres soluciones diferentes:
  - Web Speech API, utilizando el objeto `SpeechRecognition` para navegadores compatibles.
  - Vosk en navegador, ejecutando el reconocimiento de voz de forma local mediante WebAssembly y procesamiento de audio en tiempo real.
  - Whisper mediante la API de Groq, utilizando el objeto `MediaRecorder` para capturar audio y enviarlo a un servicio externo para su transcripción.
- **Investigar el uso de APIs externas a través de encaminadores**: Se ha integrado OpenRouter como intermediario para el acceso a modelos de lenguaje (LLM). Esta abstracción permite elegir el proveedor concreto del modelo, facilitando el uso de diferentes LLMs a través de una única interfaz.
- **Explorar el uso de inteligencia artificial generativa para interpretar lenguaje natural**: Se ha explorado el uso de modelos de lenguaje (LLM) que reciben instrucciones del usuario y generan una representación estructurada con la descripción de la escena. El sistema no se limita a interpretar comandos simples, sino que incorpora una serie de reglas y restricciones, permitiendo generar escenas completas o modificar elementos existentes.
- **Diseñar una arquitectura modular basada en componentes e implementar un flujo completo desde la entrada del usuario hasta la visualización de la escena**: Se han desarrollado y personalizado múltiples componentes en A-Frame que encapsulan funcionalidades específicas del sistema, como la captura de voz, dirigir comandos a otro componente, la interacción con modelos de lenguaje, la gestión del estado o el renderizado de la escena. Estos componentes permiten construir el flujo completo de la aplicación de forma desacoplada, facilitando la reutilización y la extensibilidad del sistema. Gracias a esta arquitectura, es posible combinar distintos componentes para crear sistemas más complejos o más sencillos.
- **Desarrollar mecanismos de interacción mediante comandos de voz**: Se ha implementado un sistema de comandos basado en lenguaje natural, donde el usuario puede crear, modificar y guardar la escena con el uso de frases clave.

- **Adaptar la aplicación a escritorio y realidad virtual**: El sistema se ha diseñado para funcionar tanto en navegador como en VR. En escritorio se utilizan botones para iniciar y detener la captura de voz, mientras que en VR se emplea un mecanismo de tipo *push-to-talk* con los controladores.

- **Incorporar almacenamiento y despliegue automático**: Se ha implementado un mecanismo de integración continua que permite almacenar la escena en un repositorio remoto a través de la API de GitHub.

En conjunto, todos los objetivos planteados han sido alcanzados mediante la implementación de una caja de herramientas que valida la viabilidad de la interacción mediante lenguaje natural en entornos tridimensionales. Asimismo, el trabajo demuestra que la integración de realidad virtual e inteligencia artificial permite desarrollar sistemas versátiles para la creación de entornos 3D, donde la voz se consolida como un punto de entrada que posibilita la construcción y modificación de escenas de forma intuitiva y directa.

6.2. Esfuerzo y recursos dedicados

El desarrollo de este proyecto ha requerido una dedicación considerable tanto en la parte técnica como en la elaboración de la memoria. El trabajo se ha organizado en diferentes sprints, lo que ha permitido estructurar el desarrollo de forma progresiva y abordar cada fase de manera incremental.

En cuanto a los recursos utilizados, destacan los siguientes:

- Ordenador personal para el desarrollo, pruebas y redacción de la memoria.

- Gafas de realidad virtual Meta Quest 2 para la validación en entornos inmersivos.

En el Cuadro 6.1 podemos ver las horas dedicadas al desarrollo y aprendizaje por Sprint.

Tiempo	**Sprint 1**	**Sprint 2**	**Sprint 3**	**Sprint 4**	**Sprint 5**	**Memoria**
Horas	35	15	20	25	30	40
Semanas	4	2	3	3	4	5

Cuadro 6.1: Tiempo dedicado a cada Sprint

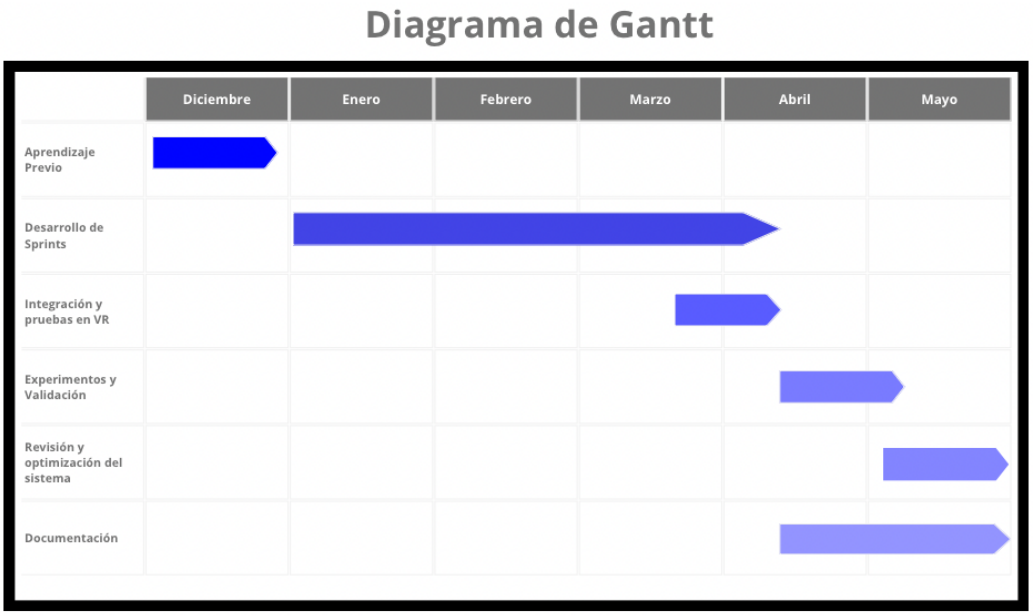


Figura 6.1: Diagrama de Gantt

6.3. Aplicación de lo aprendido

- **Fundamentos de la Programación**: Esta asignatura proporcionó las bases de la lógica de programación, incluyendo el uso de variables, estructuras de control y organización del código. Estos conceptos han sido esenciales para desarrollar la lógica del sistema en JavaScript y estructurar el flujo de la aplicación.

- **Servicios Telemáticos**: Los conocimientos adquiridos en esta asignatura han sido fundamentales para comprender el funcionamiento de los servicios web y la comunicación entre sistemas distribuidos. En el proyecto se han aplicado conceptos como la arquitectura cliente-servidor, el uso de APIs REST y la integración de servicios externos, aspectos clave en la interacción con modelos de lenguaje y sistemas de almacenamiento.

- **Aplicaciones Telemáticas**: Durante esta asignatura se adquirieron los conocimientos necesarios para el desarrollo de aplicaciones web, especialmente en HTML y JavaScript. Estos lenguajes constituyen la base del sistema desarrollado..

6.4. Conocimientos que se han tenido que adquirir

- **Desarrollo en realidad virtual**: El uso de A-Frame y la creación de entornos inmersivos en navegadores web no había sido abordado previamente en el Grado, por lo que fue necesario adquirir conocimientos específicos sobre la construcción de escenas 3D, gestión de entidades y renderizado en entornos virtuales.

- **Reconocimiento de voz e integración con modelos de lenguaje**: Se han aprendido técnicas para la captura y transcripción de voz en navegador, así como la integración con modelos de lenguaje mediante APIs externas. Esto incluye el uso de sistemas como Vosk o Whisper, así como el diseño de prompts y el procesamiento de las respuestas generadas.

- **Interacción en tiempo real en entornos 3D**: Se han adquirido conocimientos relacionados con la manipulación dinámica de objetos dentro de una escena tridimensional, incluyendo propiedades de los objetos, así como la sincronización de eventos de usuario con la actualización visual en tiempo real.

- **Integración continua y automatización**: Se ha incorporado el concepto de integración continua mediante el uso de repositorios remotos, permitiendo almacenar automáticamente los escenarios generados.

6.5. Lecciones aprendidas

A lo largo del desarrollo de este proyecto se han adquirido diversos aprendizajes tanto a nivel técnico como metodológico:

- Desarrollo de mecanismos de interacción mediante lenguaje natural a través de comandos de voz.

- Uso de A-Frame para la creación de entornos en realidad virtual.

- Diseño de sistemas modulares basados en componentes desacoplados.

- Uso de APIs externas para reconocimiento de voz y modelos de lenguaje.

- Diseño de prompts para la interpretación de instrucciones en lenguaje natural.

- Gestión de eventos para coordinar la comunicación entre componentes.
- Adaptación de la aplicación a distintos entornos (escritorio y realidad virtual).
- Importancia de la experiencia de usuario en sistemas interactivos.
- Automatización de procesos.
- Trabajar con dispositivos de realidad virtual.
- Implementar metodologías ágiles.
- Uso de LaTeX para la redacción de la documentación.

6.6. Trabajos futuros

En este proyecto se han integrado tecnologías emergentes que se encuentran en constante evolución. Por este motivo, el sistema desarrollado debe entenderse como un prototipo inicial que abre múltiples líneas de mejora y ampliación. A continuación, se presentan diversas propuestas que podrían explorarse en trabajos futuros con el objetivo de enriquecer y extender las capacidades del sistema:

- Ampliar el catálogo de objetos y texturas disponibles para aumentar la variedad de escenas generables.
- Mejorar la precisión del reconocimiento de voz, especialmente en entornos con ruido, usuarios con diferentes pronunciaciones y diferentes idiomas.
- Evolucionar hacia un sistema de interacción completamente natural, combinando voz y gestos, eliminando la dependencia de botones y permitiendo la manipulación directa de objetos en la escena.
- Optimizar el uso de modelos de lenguaje, mejorando el diseño de prompts y reduciendo la latencia de respuesta para conseguir una interacción más fluida.
- Implementar soporte multiusuario, permitiendo la colaboración en tiempo real dentro de una misma escena.

- Consolidar el sistema como un producto completo, mejorando su usabilidad, estabilidad y experiencia de usuario.
- Integrar el sistema en motores gráficos como Unity, con el objetivo de ampliar sus capacidades y explorar su uso en entornos más complejos o profesionales.
- Eliminar el uso de paneles informativos en la escena y implementar un chat de voz en la escena para ayudar al usuario en tiempo real.

Bibliografía

[1] Aldin Dynamics. Waltz of the Wizard. https://www.waltzvirtual.com/, 2026. Experiencia de realidad virtual con soporte de interacción mediante comandos de voz.

[2] Alpha Cephei Inc. Vosk speech recognition toolkit. https://alphacephei.com/vosk/, 2020. Herramienta de código abierto para reconocimiento automático del habla offline con soporte multiplataforma.

[3] Federico Alonso Aliste. Arquitectura transformers de los LLM. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://secoir.org/wp-content/uploads/2025/05/10.2-Monografia-SECOIR-2025-V1.pdf, 2025.

[4] Fielding, Roy T. et al. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. https://datatracker.ietf.org/doc/html/rfc7231, 2014. Especificación del protocolo HTTP utilizado para la comunicación entre cliente y servidor en la web.

[5] GitHub, Inc. Github actions documentation. https://docs.github.com/es/actions/get-started/understand-github-actions, 2026. Documentación oficial de GitHub Actions, plataforma utilizada para automatizar workflows.

[6] GitHub, Inc. Github repositories documentation. https://docs.github.com/es/repositories/creating-and-managing-repositories/quickstart-for-repositories, 2026. Documentación oficial sobre repositorios GitHub, utilizados para almacenar y versionar el código del proyecto.

[7] Groq Inc. Documentación API Groq. https://console.groq.com/docs/overview, 2016. Plataforma que proporciona acceso a modelos de inteligencia artificial mediante una API compatible con OpenAI y optimizada para baja latencia.

[8] Jonathan Gamble. Vosk browser: Reconocimiento de voz en el navegador. https://github.com/jonbgamble/vosk-browser, 2025. Biblioteca de reconocimiento de voz para el navegador que utiliza una compilación de Vosk mediante WebAssembly.

[9] Jong Wook, Christian Clauss, Ryan Heise, Vicki Anand. Whisper. modelos y lenguajes disponibles. https://github.com/openai/whisper, 2023. Fuente de la tabla de modelos Whisper utilizada para comparar tamaños, rendimiento y requisitos de hardware.

[10] Jeff Sutherland Ken Schwaber. *La guía de Scrum*. Ken Schwaber, Jeff Sutherland, 2013.

[11] Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland y Dave Thomas Kent Beck, Mike Beedle. Principios del manifiesto Ágil. https://agilemanifesto.org/iso/es/principles.html, 2001. Consultado el día 30 de marzo de 2026.

[12] Khronos Group. WebGL Specification. https://www.khronos.org/webgl/, 2007. Especificación oficial de WebGL para el renderizado de gráficos 2D y 3D acelerados por hardware en navegadores web.

[13] LaTeX Project. LaTeX: A document preparation system. https://www.latex-project.org/, 2024. Sistema de composición de documentos de alta calidad utilizado especialmente en entornos científicos y técnicos.

[14] Luma AI. Luma AI. https://lumalabs.ai/, 2026. Plataforma de reconstrucción y generación de escenas 3D mediante inteligencia artificial.

[15] Meshy AI. Meshy AI. https://www.meshy.ai/, 2026. Herramienta de generación de modelos 3D a partir de texto o imágenes mediante inteligencia artificial.

[16] Meta Platforms, Inc. Meta quest: Dispositivos de realidad virtual. https://www.meta.com/es/quest/, 2026. Página oficial de las gafas de realidad virtual Meta Quest, utilizadas para experiencias inmersivas en entornos VR.

[17] Microsoft. Visual studio code documentation. `https://code.visualstudio.com/docs`, 2026. Documentación oficial de Visual Studio Code, entorno de desarrollo utilizado para la edición, depuración y gestión de código en el proyecto.

[18] Mozilla Developer Network. Web Speech API: Speechrecognition. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API#api.SpeechRecognition`, 2013. Documentación oficial de la Web Speech API que permite el reconocimiento de voz en navegadores web.

[19] Mozilla Developer Network. Javascript guide. `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, 2026. Documentación oficial de JavaScript que describe sus características, funcionamiento y uso en aplicaciones web.

[20] Mozilla Developer Network. REST APIs. `https://developer.mozilla.org/en-US/docs/Glossary/REST`, 2026. Descripción del estilo arquitectónico REST y su uso en servicios web.

[21] Open AI. Whisper. `https://openai.com/es-ES/index/whisper/`, 2022. Documentación oficial de Whisper.

[22] OpenRouter. Openrouter documentation. `https://openrouter.ai/docs`, 2026. Plataforma que permite acceder a múltiples modelos de lenguaje mediante una API unificada.

[23] Supermedium and A-Frame Community. A-frame documentation. `https://aframe.io/`, 2026. Framework de código abierto para construir experiencias de realidad virtual en el navegador mediante HTML y un sistema entidad-componente.

[24] Three.js Contributors. Three.js. `https://threejs.org/`, 2026. Biblioteca de JavaScript que proporciona una abstracción de alto nivel sobre WebGL para la creación de gráficos 3D en el navegador.

[25] vTime Holdings Ltd. vTime XR. `https://vtime.net/`, 2026. Plataforma social en realidad virtual basada en entornos tipo habitaciones.

[26] World Wide Web Consortium (W3C). WebXR Device API. `https://immersive-web.github.io/webxr/`, 2022. Especificación oficial de la API WebXR que permite desarrollar experiencias de realidad virtual y aumentada en navegadores web.

[27] World Wide Web Consortium (W3C). Especificación HTML5. `https://html.spec.whatwg.org/`, 2026. Especificación oficial del lenguaje HTML5 que define la estructura y funcionamiento de las páginas web modernas.

[28] YAML. YAML specification. `https://yaml.org/`, 2026. Especificación oficial del formato YAML para la serialización de datos.