



**ESCUELA DE INGENIERÍA DE FUENLABRADA**

**INGENIERÍA EN SISTEMAS AUDIOVISUALES Y  
MULTIMEDIA**

**TRABAJO FIN DE GRADO/MÁSTER**

**EDICIÓN DE ESCENAS 3D MEDIANTE VOZ**

Autor : Pablo Esteban Camuendo Carlosama

Tutor : Dr. Jesús María González Barahona

Cotutor: (si procede)

Curso académico 2024/2025





©2025 Pablo Esteban Camuendo Carlosama

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en

<https://creativecommons.org/licenses/by-sa/4.0/deed.es>

*Dedicado a  
mi familia.*

# **Agradecimientos**

Quisiera expresar mi más sincero agradecimiento a todas las personas e instituciones que han hecho posible la realización de esta tesis de grado.

En primer lugar, mi familia, durante todos estos años de estudio, por su apoyo incondicional y esfuerzo, su comprensión y aliento continuo. Su fe en mí es la mayor inspiración. Ellos siempre han dicho que la educación es la base de todo lo que puedo construir a continuación y es gracias a ellos que he podido llegar aquí. Gracias, mamá y papá

A mi hermana, que ha sido un constante apoyo y de la cual quiero ser un ejemplo a seguir, que sepa que el desarrollo de uno mismo es difícil, pero satisfactorio al final.

A mi tutor de tesis, por su guía, paciencia y conocimientos compartidos a lo largo de este proceso. Su experiencia ha sido crucial para el desarrollo de este trabajo.

Finalmente, a todos aquellos que de una u otra manera han contribuido a mi crecimiento personal y profesional durante mi etapa universitaria.

A todos, mi más profunda gratitud.

# **Resumen**

RESUMEN LO HAGO AL FINAL ...

# **Summary**

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Estructura de la memoria . . . . .	2
1.2. Objetivo general . . . . .	2
1.3. Objetivos específicos . . . . .	3
<b>2. Tecnologías utilizadas</b>	<b>4</b>
2.1. Tecnologías de reconocimiento de voz . . . . .	4
2.1.1. Whisper . . . . .	5
2.1.2. Whisper Web . . . . .	6
2.1.3. Web Speech API . . . . .	7
2.1.4. AssemblyAI . . . . .	9
2.2. A-Frame . . . . .	10
2.3. Three.js . . . . .	13
2.4. WebXR . . . . .	14
2.5. WebGL . . . . .	15
2.6. HTML . . . . .	16
2.7. JavaScript . . . . .	17
2.7.1. Cliente (front-end) . . . . .	17
2.7.2. Servidor (back-end) . . . . .	17
2.8. CSS . . . . .	18
2.9. NodeJS . . . . .	19
2.9.1. Node.js y el Manejo de Peticiones HTTP . . . . .	19
2.9.2. Certificados . . . . .	20
2.10. Aplicaciones de inspiración al proyecto . . . . .	21

2.10.1. Arkio: . . . . .	21
2.10.2. VR Sculpting: . . . . .	22
2.10.3. Bridge Crew (Star Trek) . . . . .	23
2.11. Tecnologias auxiliares . . . . .	24
2.11.1. Github . . . . .	24
2.11.2. Visual Studio Code . . . . .	24
2.11.3. Gafas Meta Quest3 . . . . .	25
2.11.4. LaTeX . . . . .	26
<b>3. Desarrollo del proyecto</b>	<b>27</b>
3.1. Arquitectura general . . . . .	28
3.2. Sprint 1 . . . . .	29
3.2.1. Objetivos . . . . .	29
3.2.2. Tareas Realizadas . . . . .	29
3.2.3. Resultados . . . . .	31
3.2.4. Lecciones aprendidas . . . . .	31
3.3. Sprint 2 . . . . .	32
3.3.1. Objetivos . . . . .	32
3.3.2. Tareas Realizadas . . . . .	32
3.3.3. Resultados . . . . .	34
3.3.4. Lecciones aprendidas . . . . .	35
3.4. Sprint 3 . . . . .	36
3.4.1. Objetivos . . . . .	36
3.4.2. Tareas Realizadas . . . . .	36
3.4.3. Resultados . . . . .	37
3.4.4. Lecciones aprendidas . . . . .	37
3.5. Sprint 4 . . . . .	38
3.5.1. Objetivos . . . . .	38
3.5.2. Tareas Realizadas . . . . .	38
3.5.3. Resultados . . . . .	40
3.5.4. Lecciones aprendidas . . . . .	40

3.6. Sprint 5 . . . . .	41
3.6.1. Objetivos . . . . .	41
3.6.2. Tareas Realizadas . . . . .	41
3.6.3. Resultados . . . . .	42
3.6.4. Lecciones aprendidas . . . . .	42
3.7. Sprint 6 . . . . .	43
3.7.1. Objetivos . . . . .	43
3.7.2. Tareas Realizadas . . . . .	43
3.7.3. Resultados . . . . .	44
3.7.4. Lecciones aprendidas . . . . .	44
<b>4. Resultados</b>	<b>45</b>
4.1. Flujo de funcionamiento . . . . .	46
4.2. Componentes . . . . .	47
4.2.1. dynamic-modifier . . . . .	47
<b>5. Experimentos y validación</b>	<b>51</b>
5.1. Diseño Experimental . . . . .	53
5.2. Resultados . . . . .	53
5.2.1. Precisión del Reconocimiento de Voz . . . . .	54
5.2.2. Tiempo Promedio por Tarea . . . . .	54
5.2.3. Opiniones de los Usuarios . . . . .	55
5.3. Conclusión de Validación . . . . .	55
<b>6. Conclusiones</b>	<b>56</b>
6.1. Consecución de objetivos . . . . .	56
6.2. Esfuerzo y recursos dedicados . . . . .	56
6.2.1. Planificación temporal . . . . .	58
6.3. Aplicación de lo aprendido . . . . .	59
6.4. Lecciones aprendidas . . . . .	59
6.5. Trabajos futuros . . . . .	59
<b>A. Manual de usuario</b>	<b>61</b>



# Índice de figuras

2.1. Modelos de Whisper . . . . .	5
2.2. Whisper Web . . . . .	6
2.3. Compatibilidad en exploradores . . . . .	8
2.4. Ejemplo basico de WebSpeechAPI . . . . .	8
2.5. Escena simple A-Frame . . . . .	11
2.6. Ejemplo aplicacion en WebGL . . . . .	15
2.7. Ejemplo de escena de Arkio . . . . .	21
2.8. Ejemplo de uso de Shapelab . . . . .	22
2.9. Ejemplo de uso de Star Trek . . . . .	23
2.10. Gafas Meta Quest3 . . . . .	25
3.1. Funcionamiento de la segunda Demo para A-Frame . . . . .	31
3.2. Flujo de creación de componentes . . . . .	33
3.3. Flujo de creación de componentes avanzados . . . . .	33
3.4. Funcionamiento del comando basico Demo 4 . . . . .	34
3.5. Flujo de creación de componentes avanzados . . . . .	36
3.6. Funcionamiento del generador de objetos con funcionalidades extra . . . . .	37
3.7. Flujo de creación de componentes avanzados . . . . .	38
3.8. Funcionamiento de eliminar objetos con ID . . . . .	39
3.9. Funcionamiento de editar objetos con ID . . . . .	39
3.10. Funcionamiento del generador de assets . . . . .	40
3.11. Funcionamiento de htmlembbed . . . . .	43
3.12. componentes finales agregados . . . . .	44

4.1. Estructura del funcionamiento del Proyecto . . . . .	46
4.2. Funcionamiento de eliminar objetos con ID . . . . .	48
4.3. Funcionamiento de editar objetos con ID . . . . .	49
4.4. Funcionamiento del generador de assets . . . . .	49

# Capítulo 1

## Introducción

Con el gran avance en tecnologías, la realidad virtual (VR) ha experimentado un gran crecimiento en los últimos años, gracias a la evolución de tecnologías web accesibles y de código abierto. A-Frame es un framework potente y versátil, basado en WebXR y Three.js, que facilita la creación y configuración de escenas aprovechando directamente el navegador usando HTML y JavaScript.

Unido a este framework tendremos el desarrollo del reconocimiento del habla que permitirá controlar la escena mediante comandos de voz. La sinergia entre A-Frame y el control por voz representa un enfoque innovador con el potencial de transformar la accesibilidad, la usabilidad y la inmersión en las experiencias de realidad virtual, ofreciendo una alternativa intuitiva a los métodos de interacción tradicionales.

Este proyecto surge de la inquietud por investigar nuevas formas de interacción en entornos 3D web. A día de hoy existen avances significativos en la creación visual de escenas virtuales, la edición y manipulación de objetos puede resultar en ocasiones compleja y poco intuitiva. La idea de un editor controlado por voz sería como una alternativa prometedora para agilizar el flujo de trabajo y mejorar la accesibilidad para usuarios con diferentes necesidades.

Actualmente, se ha avanzado en la exploración de diferentes arquitecturas y sistemas de componentes para la implementación de un editor 3D dirigido por voz. Como resultado de esta investigación, se ha desarrollado un prototipo funcional que permite la creación y edición básica de objetos en una escena 3D mediante comandos de voz. Sin embargo, es importante señalar que la aplicación aún se encuentra en una fase de desarrollo y no está completamente pulida. El trabajo futuro se centrará en refinar la funcionalidad existente, ampliar el conjunto de comandos

de voz y mejorar la estabilidad y el rendimiento general del sistema.

## 1.1. Estructura de la memoria

El resto de la memoria se estructurara de la siguiente manera:

- 2 Tecnologías utilizadas: Se muestran todas las tecnologías utilizadas durante el desarrollo del proyecto, desde las principales como A-Frame para la construcción de entornos inmersivos en el navegador, hasta herramientas auxiliares como Node.js o WebXR. Se da una breve descripción de cada tecnología, el contexto de uso y su papel dentro de cada una de las demos.
- 3 Desarrollo del Proyecto: Se detalla el proceso completo de creación, donde se implementaron dos versiones de la demo: una que usa la Web Speech API directamente en el navegador, y otra que envía audio a un servidor local para procesarlo con Assembly AI.
- 4 Resultados: Aquí se muestra el resultado final de ambas demos
  - Descripción para el usuario: Se explica qué funcionalidades incluye cada demo, cómo se activan los comandos por voz y cómo se muestra la transcripción dentro del entorno 3D.
  - Descripción del comportamiento de la interfaz: Se detalla cómo se representa la transcripción en la escena VR/AR, así como la lógica de actualización dinámica de objetos en el espacio virtual.
- 5 Experimentos y validación
- 6 Conclusiones

## 1.2. Objetivo general

El trabajo de fin de grado consiste en explorar diversas metodologías para la implementación de un editor 3D dirigido por voz, finalizando en el desarrollo de un prototipo funcional basado en un sistema de componentes que demuestre la viabilidad de la creación y edición de escenas

3D mediante comandos de voz en navegadores web compatibles. En entornos de escritorio o dispositivos VR/AR como las gafas Quest3

### 1.3. Objetivos específicos

- Explorar distintas soluciones para encontrar una que funcione bien en gafas tipo Quest3 y otra en el navegador de escritorio.
- Explorar qué opciones se tienen para Speech-to-text en un entorno como un navegador o dispositivos tipo Quest3.
- Conseguir transcripciones del usuario con la aplicación.
- Reconocimiento de órdenes concretas para interactuar con el editor de escenas.
- Explorar distintas soluciones para encontrar una que funcione bien en gafas tipo Quest3 y otra en el navegador de escritorio.
- Hacer que el prototipo soporte comandos como crear, editar, mover...
- Construir el prototipo de forma que funcione en el navegador en escritorio y en equipos VR/AR como las gafas Quest3.
- Utilizar para la construcción el framework A-Frame, porque se adapta especialmente bien al objetivo anterior.
- Realización de un prototipo modular.

# **Capítulo 2**

## **Tecnologías utilizadas**

El proyecto utiliza la interfaz webkitSpeechRecognition para implementar el reconocimiento de voz en el navegador. Esta interfaz es una versión específica para WebKit de la API de reconocimiento de voz de la Web Speech API, que permite a las aplicaciones web convertir el habla en texto.

Además, el proyecto emplea tecnologías web estándar como HTML, CSS y JavaScript para construir la interfaz de usuario y manejar la lógica del cliente. Estas tecnologías trabajan conjuntamente para proporcionar una experiencia interactiva de reconocimiento de voz directamente en el navegador.

Se creó un flujo de trabajo extra para el uso del proyecto en gafas de VR/AR, debido a la creación de un servidor privado lanzado con Node.js → para ejecutar el servidor back-end. framework HTTP → Para manejar las solicitudes del cliente. Assembly AI API → Servicio externo de reconocimiento de voz. Fetch → Para enviar los datos de audio a Assembly AI desde el servidor.

\*\* PONER ALGUNA CITA EN CADA APARTADO SPEECH \*\* W3C CONSTRUCCIÓN SOBRE LOS OBJETOS A-FRAME ¿THREE ¿WEBXR ¿WEBGL A FRAME HABLAR DE COMPONENTES, CAPTURAS, ACCIONES, LLAMADOS, CÓDIGO, ESCENA

### **2.1. Tecnologías de reconocimiento de voz**

El reconocimiento de voz en navegadores permite a los usuarios interactuar con aplicaciones web y contenido utilizando su voz en lugar de escribir. Esto abre un mundo de posibilidades para

la accesibilidad, la productividad y la creación de experiencias de usuario más interactivas.

### 2.1.1. Whisper

En primera instancia se planeaba trabajar con Whisper, un modelo de 'Speech Recognition' de OpenAI que se destaca por su capacidad para transcribir audio en múltiples idiomas con alta precisión, incluso en condiciones de ruido o con variaciones en el acento del hablante, con el objetivo de integrarlo al proyecto.

Whisper<sup>1</sup> jugaba una parte esencial. Se eligió debido a la arquitectura robusta y de código libre. Como parte de la investigación, se analizaron los requisitos técnicos para la implementación, incluyendo dependencias, consumo de recursos, y compatibilidad con otros componentes del sistema.

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 2.1: Modelos de Whisper

Whisper presenta los modelos que se pueden ver en la Figura 2.1. Se pueden observar diferentes tamaños de modelos, cada uno con distintos niveles de precisión, velocidad y requerimientos de hardware.

Los modelos más grandes (medium, large) son altamente demandantes de GPU, esto hace que los modelos tengan mucha más precisión, pero que su velocidad sea más afectada. Para una menor latencia, los modelos más pequeños (tiny, base, small) son mejores, ya que su demanda de hardware es menor.

Alguno de los problemas que presentó Whisper fue el tamaño de los modelos, ya que incluso el modelo más pequeño (tiny) pesa varios cientos de MB. Se necesita PyTorch y dependencias

---

<sup>1</sup>Whisper OpenAI github [10].

que no están disponibles en entornos web estándar (como JS/WebAssembly). Además del uso intensivo de GPU/CPU, lo cual no es viable en la mayoría de navegadores móviles o de escritorio.

### 2.1.2. Whisper Web

Actualmente, algunos proyectos han empezado a portar versiones ligeras al navegador (construcciones en WebAssembly u ONNX). Un ejemplo de esto es Whisper Web<sup>2</sup>

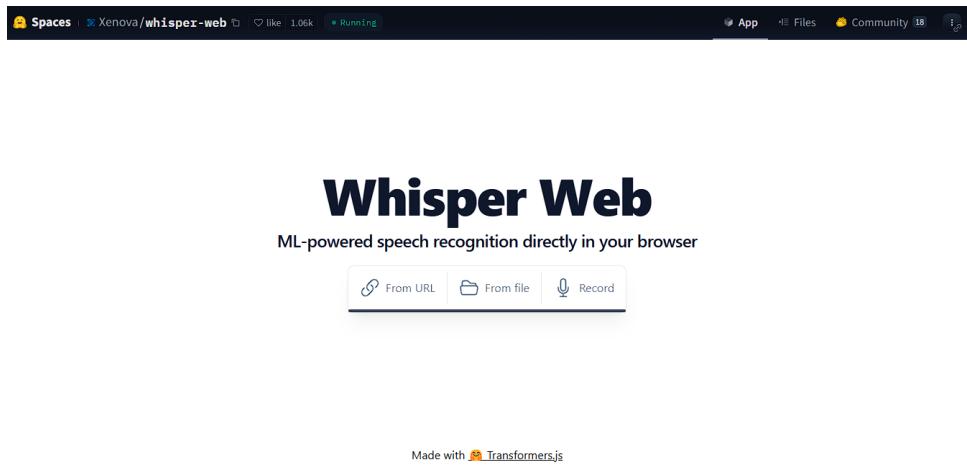


Figura 2.2: Whisper Web

En la Figura 2.2 se muestra un proyecto de la comunidad que busca llevar Whisper al navegador, sin la necesidad de un back-end. Para ello se usan tecnologías como:

- WebAssembly (WASM): para correr código pesado directamente en el navegador.
- ONNX (Open Neural Network Exchange): para convertir modelos de PyTorch a un formato más portable.
- WebGPU o WebGL: para aprovechar el hardware del usuario (si es compatible).

Para ello se da uso a la plataforma y comunidad de Hugging Face enfocada en el desarrollo, distribución y uso de modelos de inteligencia artificial.

El funcionamiento en esta aplicación web es el siguiente:

- Se carga un archivo de audio o se graba la voz.

---

<sup>2</sup>Whisper Web Hugging Face [13].

- En el navegador elige una versión comprimida del modelo Whisper y la elección del lenguaje si el usuario lo desea.
- El modelo se ejecuta localmente en el navegador (no sube tu audio a la nube).
- La app procesa el audio y muestra la transcripción en pantalla.
- El navegador puede exportar la información.

Esta idea se descartó debido a la complejidad de implementación en el navegador, ya que ejecutar modelos de reconocimiento de voz como Whisper directamente en la web implica múltiples desafíos técnicos. Entre ellos destacan el alto consumo de recursos, la falta de soporte en entornos como WebAssembly/WebGPU, y la necesidad de transcribir y optimizar los modelos para que puedan ejecutarse localmente en el cliente sin afectar la experiencia del usuario.

Además, el peso de los modelos de sus versiones más pequeñas, puede afectar significativamente los tiempos de carga y procesamiento, especialmente en dispositivos móviles o con hardware limitado. Por estas razones, se optó por explorar soluciones alternativas que pudieran integrarse de manera más eficiente dentro del proyecto.

### 2.1.3. Web Speech API

La principal tecnología nativa para reconocimiento de voz directamente en los navegadores es la Web Speech API, específicamente su interfaz SpeechRecognition.

La Web Speech API fue introducida como un borrador por el W3C (World Wide Web Consortium) en 2012. Aunque algunas partes de la API, como la síntesis de voz (SpeechSynthesis), han alcanzado un estado de recomendación, la parte de reconocimiento de voz (SpeechRecognition) aún se considera un borrador de trabajo. Sin embargo, ha sido implementada en la mayoría de los navegadores modernos, aunque con diferentes niveles de soporte y posibles requisitos de prefijos específicos del navegador (ej.: webkitSpeechRecognition en Chrome), por ejemplo el uso de esta API en navegadores específicos para gafas VR/AR como las Quest3 es directamente imposible, debido a que no hay soporte para ello, por esto lo más factible para el uso de reconocimiento de voz en casos como estos es usar un back-end que reciba las peticiones y chunks de audio del cliente y éste mismo las transcriba devolviendo así el texto transcrita.

El uso principal que le daríamos para este proyecto será el 'SpeechRecognition' para convertir la voz en texto. Es cierto que la compatibilidad de navegadores es muy grande, pero el soporte de SpeechRecognition no es estándar aún, y muchos navegadores solo lo permiten con el prefijo webkit (webkitSpeechRecognition). Funciona principalmente en Chrome y navegadores basados en Chromium, como se ve en la Figura 2.3.

	□					□						
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS
<b>SpeechRecognition</b>	✓ 33	✓ 79	✗ No	✓ 20	✓ 14.1	✓ 33	✗ No	✓ 20	✓ 14.5	✓ 2	✓ 4.4.3	✓ 14.5
	*	*	*	*	*	*	*	*	*	*	*	*

Figura 2.3: Compatibilidad en exploradores

Finalmente, una de las razones por las que se optó por esta opción es su facilidad de implementación, ya que no requiere de librerías externas, se ejecuta directamente en el navegador y es ideal para el uso de prototipos, demos o apps ligeras.

```
// Verificar compatibilidad con la API de reconocimiento de voz
if ('webkitSpeechRecognition' in window) {
    recognition = new webkitSpeechRecognition();
    recognition.continuous = true;
    recognition.interimResults = false;
    recognition.lang = 'es-ES';
} else {
    console.error('API de reconocimiento de voz no soportada en este navegador');
}
```

Figura 2.4: Ejemplo básico de WebSpeechAPI

Como se observa en la Figura 2.4 la integración se hace en pocas líneas de código. Este código muestra la activación de reconocimiento continuo '**recognition.continuous = true;**', esto quiere decir que se procesara la transcripción continuamente, pero con la condición de devolver el resultado final gracias a '**recognition.interimResults = false;**' además de seleccionar el idioma que se desea transcribir.

Por otro lado, no todas son ventajas, existen limitaciones para el uso de esta API, tales como el requerimiento de conexión a internet, ya que utiliza los servidores de Google para el reconocimiento. Además, el soporte y el comportamiento pueden variar entre diferentes navegadores. Y sobre todo, cuenta con una menor precisión en comparación con modelos como Whisper, especialmente en ambientes ruidosos o con acentos fuertes.

#### 2.1.4. AssemblyAI

Assembly AI<sup>3</sup> es una plataforma de Inteligencia Artificial que ofrece potentes APIs para el procesamiento de audio, incluyendo reconocimiento de voz (Speech-to-Text), comprensión del lenguaje natural (NLP) y otras funcionalidades de inteligencia de audio.

Cuenta con características similares a Web Speech API, salvo que tiene algunas mejoras en otros puntos.

Assembly AI se enfoca en ofrecer modelos de reconocimiento de voz de última generación entrenados en grandes cantidades de datos, lo que generalmente resulta en una mayor precisión, especialmente en entornos ruidosos o con acentos variados, con la posibilidad de identificar al hablante, detectar las emociones expresadas en el audio, incluso admite diferentes formatos de archivo y fuentes de audio/video que puedes ser en tiempo real o no.

La API es robusta y escalable, fue diseñada para desarrolladores que necesitan integrar capacidades de reconocimiento de voz en aplicaciones complejas y a gran escala. Por ello al ser una API externa, funciona de manera consistente en diferentes navegadores y plataformas, su uso en navegadores se realizará a través de una API REST y las respuestas se reciben como respuesta a dicha API por esto requiere una conexión a internet activa para enviar y recibir datos de la API. Assembly AI es un servicio comercial y tiene costos asociados en función del volumen de audio procesado, aunque a menudo ofrecen planes gratuitos o de prueba.

Finalmente, usamos esta tecnología para el proyecto en una segunda demo para el uso de gafas VR/AR, debido a que los navegadores de Meta de las gafas de pruebas no soportaban el manejo de la API anterior de Web Speech Recognition.

---

<sup>3</sup>Documentacion de Assembly [3].

## 2.2. A-Frame

Para la implementación en AR/VR se uso A-Frame<sup>4</sup>, este es un framework web de código abierto usado para el desarrollo de experiencias en realidad virtual y aumentada. Utilizando HTML como lenguaje principal para la definición de escenas. Este framework se basa en Three.js, el cual es una biblioteca de JavaScript para gráficos en 3D.

A-Frame se caracteriza por el uso simplificado de Three.js para la creación de escenas, evitando el directo y complejo gestionamiento de los detalles de renderización en 3D.

Usando una arquitectura llamada ECS (Entity Components System) que es aplicada a los videojuegos, en donde cada objeto es una entidad diferenciada, que puede o no albergar otras entidades. Debemos tener en cuenta que para acceder a la librería de este framework debemos usar la línea de código del Listing 2.1. Esta etiqueta `<script>` permite cargar la versión 1.7.0 de A-Frame directamente desde la web. Dicha versión puede variar.

```
<script src='https://aframe.io/releases/1.7.0/aframe.min.js'></script>
```

Listing 2.1: Línea de código de A-Frame

Una de las características más claras de A-Frame es la definición de entidades utilizando un etiquetado similar al de HTML. Esto permite a desarrolladores construir mundos virtuales de manera rápida y simple, como se observa en este fragmento de código del Listing 2.2:

```
<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>
```

Listing 2.2: Escena A-Frame básica

---

<sup>4</sup>Véase la página de referencia de A-Frame [1].

Cada entidad de esta escena tiene sus propiedades especificadas en la página de la documentación de A-Frame<sup>5</sup>

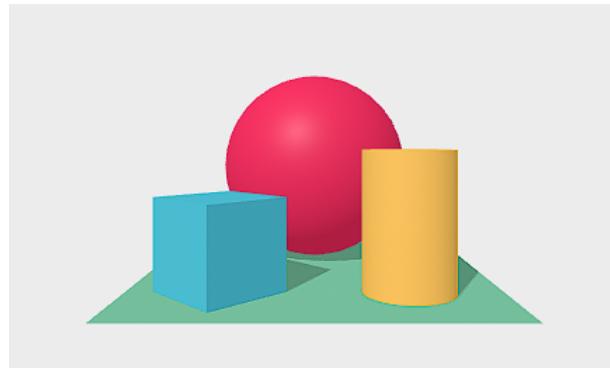


Figura 2.5: Escena simple A-Frame

El núcleo de A-Frame es su sistema de entidades-componentes. Las entidades son objetos genéricos que adquieren propiedades y comportamientos mediante componentes. Los componentes son módulos reutilizables que añaden comportamientos, propiedades visuales, interacciones y varias acciones más a las entidades. A-Frame proporciona componentes predefinidos (position, rotation, scale, material, geometry...). Se puede observar en el Listing 2.3, el componente añade una entidad con un <a-text> que dará una información al usuario.

```
AFRAME.registerComponent('text-message', {
  init: function () {
    const userMessage = document.createElement('a-text');
    userMessage.setAttribute('id', 'userMessage');
    userMessage.setAttribute('value', 'Bienvenido a VOICE VR');
    userMessage.setAttribute('align', 'center');
    userMessage.setAttribute('color', 'black');
    userMessage.setAttribute('position', '0 20 -45');
    userMessage.setAttribute('width', '25');
    userMessage.setAttribute('look-at', '[camera]');
    this.el.appendChild(userMessage);
  }
});
```

Listing 2.3: Crear Componente

---

<sup>5</sup>Véase la documentación de A-Frame [2].

Una entidad puede tener muchos componentes a la vez. Se pueden modificar los valores de un componente en tiempo real usando JavaScript. La creación de componentes depende de varias cosas como:

- **Definición del componente:** Se realiza utilizando el método `AFRAME.registerComponent`.
- **schema:** Define los datos que el componente puede recibir, como configuraciones o parámetros personalizables.
- **init():** Función que se ejecuta al iniciar el componente. Se utiliza para establecer comportamientos iniciales o escuchar eventos de la escena.
- **this.el:** Hace referencia a la entidad (`<a-entity>`) sobre la cual está aplicado el componente.

## 2.3. Three.js

Como mencionamos anteriormente, A-Frame se construye sobre la potente biblioteca de gráficos 3D Three.js<sup>6</sup> el cual es una biblioteca de JavaScript que facilita la creación de gráficos 3D. A-Frame abstrae la complejidad de Three.js, permitiendo a los desarrolladores centrarse en la creación de contenido sin tener que lidiar directamente con la configuración de WebGL, shaders, matrices, etc. Sin embargo, para necesidades más avanzadas, los desarrolladores pueden acceder directamente al objeto Three.js subyacente dentro de los componentes personalizados de A-Frame. Con Three.js, puedes crear y manipular formas básicas como cubos o esferas, importar modelos complejos (en formatos como GLTF), aplicar materiales y texturas, e iluminar escenas con luces realistas. Además, incluye cámaras y controles intuitivos para navegar, así como soporte para animaciones fluidas.

La construcción de escenas mediante Three.js utilizará componentes como:

- **Escena (Scene)**: el contenedor de todos los elementos 3D, como objetos, luces y cámaras.
- **Cámara (Camera)**: define el punto de vista desde el que se observa la escena. Las más comunes son la cámara de perspectiva y la ortográfica.
- **Renderizador (Renderer)**: convierte la escena y la cámara en una imagen visible en el lienzo HTML (<canvas>), usando WebGL.
- **Objetos (Mesh)**: representan modelos 3D, creados a partir de geometrías (formas básicas como cubos, esferas, planos, etc.) y materiales (color, textura, iluminación).
- **Luces (Light)**: iluminan la escena y permiten que los materiales reaccionen visualmente según el tipo de luz aplicada.

---

<sup>6</sup>Learning Three.js: The JavaScript 3D Library for WebGL [5].

## 2.4. WebXR

Contextualizando este apartado debemos conocer la historia y desarrollo de WebXR. Fue desarrollada bajo las especificaciones del consorcio de W3C (World Wide Web)<sup>7</sup>.

Su predecesor fue WebVR

'WebVR is an open specification that makes it possible to experience immersive virtual reality in your browser.'<sup>8</sup>

Esta era una API experimental creada únicamente para el desarrollo en realidad virtual. Este estándar fue algo fundamental para el desarrollo actual, pero presentaba varias limitaciones, ya que no abordaba los problemas relacionados con la realidad aumentada. La evolución y la integración de la realidad aumentada es la que creó el actual WebXR.

WebXR(Web Extended Reality) es un conjunto de tecnologías web que permite crear experiencias en realidad virtual (VR) y en realidad aumentada (AR) dentro de navegadores. Esto implica que cualquier usuario puede acceder a estas experiencias sin necesidad de descargar aplicaciones separadas.

WebXR presenta varias funcionalidades como:

- **Renderizado de Escenas 3D:** Facilita el renderizado de escenas 3D en estos dispositivos a las velocidades de fotogramas adecuadas, creando que la experiencia sea inmersiva.
- **Seguimiento de Movimiento:** WebXR puede rastrear el movimiento y la orientación de la cabeza y los controladores del usuario, lo que permite la interactividad.
- **Manejo de Entrada:** Admite la entrada de varios controladores y dispositivos XR, lo que permite la interacción del usuario.

Estas funcionalidades mencionadas más otras, crean las beneficiosas características que hacen que a este tipo de tecnologías web sean accesibles y compatibles en diferentes plataformas, aprovechando las diferentes tecnologías web familiares como JavaScript, HTML y WebGL para crear experiencias WebXR.

---

<sup>7</sup>WebXR según W3C [12].

<sup>8</sup>WebVR según W3C [7].

## 2.5. WebGL

WebGL surgió por la necesidad de llevar los gráficos 3D directamente a un navegador. Antes de que los gráficos 3D se limitaban a tecnologías basadas en plugins como 'Adobe Flash' o renderización puramente con JavaScript, esto resultaba en una caída abrupta de rendimiento.

Creada en 2007 por el grupo Khronos<sup>9</sup> fue desarrollada basándose en una API conocida como OpenGL ES 2.0 diseñada para dispositivos embebidos.

*'WebGL is an API that brings hardware-accelerated 3D graphics to the Web, leveraging the widely adopted OpenGL ES 2.0 standard.' [6]*

Esta tecnología fue impulsada gracias a empresas como Mozilla y Google, quienes fueron actores clave para la implementación y desarrollo del WebGL.

Una de las características más importantes de WebGL es la capacidad de aprovechamiento de la GPU de los dispositivos del usuario, permitiendo una renderización más rápida y eficiente de gráficos en 3D. Por otro lado, la integración en el DOM junto con otras tecnologías web como HTML, CSS y JavaScript ayudan al desarrollo de escenas en 3D como la Figura 2.6. La multiplataformeidad que ofrece WebGL permite que se pueda ejecutar en cualquier navegador, independientemente de su sistema operativo.

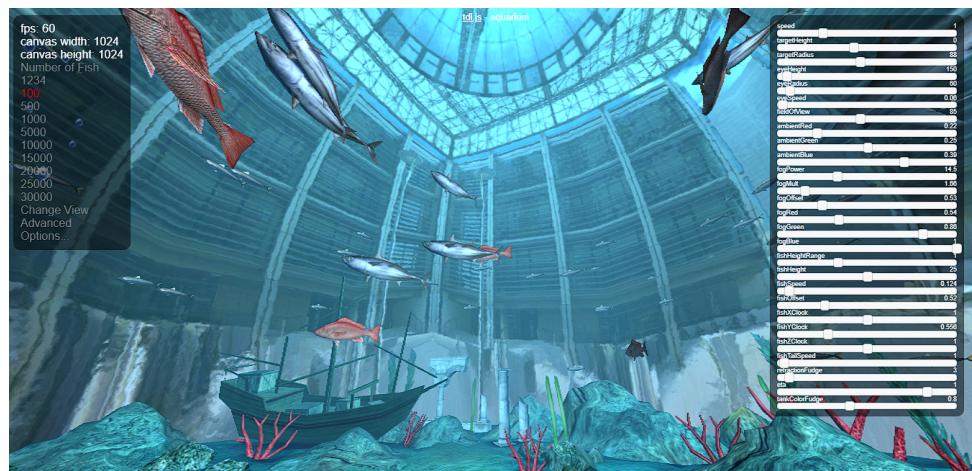


Figura 2.6: Ejemplo aplicación en WebGL

---

<sup>9</sup>Khronos Group [6].

## 2.6. HTML

HTML(HyperText Markup Language) fue inventado por Tim Berners-Lee en el CERN (Organización Europea para la Investigación Nuclear) a principios de la década de 1990. Se buscaba que los científicos pudieran compartir información fácilmente a través de una red, en donde el hipertexto de HTML<sup>10</sup> jugaba una parte esencial.

*'HTML (HyperText Markup Language) is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page. The markup tells the web browser how to display a web page's words and images for the user.'* [4]

Esta definición de W3C muestra el propósito del HTML. A medida que la web evolucionó, HTML también lo hizo creando diferentes versiones que se iban adaptando al uso de navegadores.

La versión más moderna versión, el HTML5, introdujo una gran cantidad de características, como el soporte multimedia de audio y video, además de la posibilidad de Canvas para gráficos 2D.

El HTML como tal se conoce como lenguaje de marcado, lo que significa que utiliza etiquetas (tags) para estructurar el contenido. Estas etiquetas indican el significado y la presentación de diferentes elementos (encabezados, párrafos, imágenes, enlaces, etc.). Los documentos HTML cuentan con una estructura jerárquica basada en el anidamiento de elementos; esto permite introducir elementos o etiquetas dentro de otras, definiendo una relación entre dichos objetos de 'Padre-Hijo'. HTML5 introdujo elementos semánticos como:

La mayoría de las etiquetas usadas en HTML se deberán cerrar tras su uso.

---

<sup>10</sup>HTML W3C [4].

## 2.7. JavaScript

'JavaScript es un lenguaje de programación interpretado que permite implementar funcionalidades complejas en páginas web, haciéndolas más dinámicas e interactivas.' [9]

Diseñado originalmente para ejecutarse en el navegador del cliente (front-end). Se caracteriza por la posibilidad de ser ejecutado línea a línea por el navegador, sin tener que compilarse antes de ello, además al ser un lenguaje de alto nivel facilita la escritura y comprensión de código, además también cuenta con un tipado de variables lo que significa que dichas 'variables' pueden cambiar a lo largo de la ejecución. Hoy en día también se usa en el lado del servidor (back-end) con entornos como Node.js, así como en desarrollo de aplicaciones móviles y de escritorio.

### 2.7.1. Cliente (front-end)

En el lado del cliente, JavaScript<sup>11</sup> permite manipular el DOM (Document Object Model), gestionar eventos del usuario, y hacer peticiones asincrónicas al servidor (AJAX o Fetch API), lo que lo convierte en una herramienta fundamental para crear experiencias de usuario dinámicas y reactivas. También ofrece acceso a diversas APIs del navegador que permiten funcionalidades avanzadas como geolocalización, almacenamiento local, gráficos mediante Canvas o WebGL, y comunicación en tiempo real con WebSockets o WebRTC.

### 2.7.2. Servidor (back-end)

En el entorno del servidor, JavaScript ha cobrado gran relevancia gracias a Node.js, que permite ejecutar código JavaScript fuera del navegador. Con Node.js es posible crear servidores web completos, manejar bases de datos y demás acciones. Su uso potenciado por npm (Node Package Manager), ofrece miles de módulos reutilizables para acelerar el desarrollo.

JavaScript juega un papel fundamental en el desarrollo de páginas web dinámicas, dotando al HTML antes mencionado con diferentes posibilidades de acción.

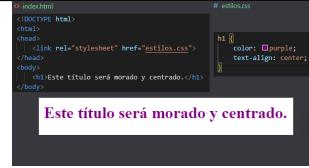
A lo largo del desarrollo del proyecto se ha usado en diferentes escenarios, desde la creación de un servidor simple, hasta la codificación de componentes para A-Frame con la finalidad de dar funcionalismo a escenas en 3D desarrolladas en dicho framework.

---

<sup>11</sup>Estructura de JavaScript [8].

## 2.8. CSS

CSS (Cascading Style Sheets) es como la cobertura y la decoración de una página web (el HTML sería el esqueleto). Se usa para decirle al navegador cómo mostrar los elementos de HTML: qué colores usar, qué fuentes, cómo se deben organizar en la pantalla, si deben tener márgenes, bordes, etc. Gracias a CSS, es posible crear interfaces web atractivas, responsivas y coherentes en distintos dispositivos. Para su uso se podrá utilizar de la siguiente forma:

Método	Descripción	Ejemplo Visual
Inline	Se aplica directamente en el atributo <code>style</code> de un elemento HTML. Útil para aplicar estilos rápidos.	 <pre>&lt;p style="color: blue; font-size: 16px;"&gt;Este texto será azul.</pre> <p>Este texto será azul.</p>
Internal	Se define dentro del bloque <code>&lt;style&gt;</code> en la sección <code>&lt;head&gt;</code> del HTML.	 <pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt;   &lt;style&gt;     p {       color: green;       font-weight: bold;     }   &lt;/style&gt; &lt;/head&gt; &lt;body&gt;   &lt;p&gt;Este texto será verde y en negrita.&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre> <p>Este texto será verde y en negrita.</p>
External	Consiste en enlazar un archivo externo con extensión <code>.css</code> mediante <code>&lt;link&gt;</code> . Es la forma más escalable.	 <pre>index.html # estilos.css &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head&gt;   &lt;link rel="stylesheet" href="estilos.css"&gt; &lt;/head&gt; &lt;body&gt;   &lt;h1&gt;Este título será morado y centrado.&lt;/h1&gt; &lt;/body&gt;</pre> <p>Este título será morado y centrado.</p>

Cuadro 2.1: Formas de aplicar CSS en un documento HTML

Esta tecnología la usamos particularmente en una sección en la que mezclamos un HTML con estilos dentro de un panel de A-Frame conocido como 'HTML Embed Component'<sup>12</sup>.

<sup>12</sup>HTML Embed Component information [11]

## 2.9. NodeJS

Node.js es un entorno de ejecución de JavaScript de código abierto y multiplataforma. Permite ejecutar código JavaScript fuera de un navegador web.

Node.js fue creado por Ryan Dahl y se lanzó inicialmente en 2009. Se estaba buscando una forma más eficiente de manejar conexiones y concurrencia en la web después de experimentar problemas con el servidor web Apache. Se inspiró en lenguajes orientados a eventos y propuso una arquitectura basada en un bucle de eventos no bloqueante. La clave de esto fue la utilización del motor V8 presente en Google Chrome, el cual compila JavaScript de manera rápida y eficiente.

Node.js se centra en una arquitectura orientada a eventos y no bloqueante. Esto significa que, en lugar de esperar a que una operación de entrada/salida (E/S) se complete (bloqueando el hilo), Node.js registra una función de callback que se ejecutará una vez que la operación finalice. Esto permite manejar muchas conexiones simultáneas de manera eficiente con un solo hilo.

Existen problemas con lo anterior mencionado; uno de los más importantes era el anidamiento de callback, ya que con un número excesivamente grande de estos, la lectura y mantenimiento de estas se vuelven difíciles. Este problema se mitigó con la llegada de las promesas y las funciones `async/await`. También el uso de tareas complejas podría sobrecargar la CPU y afectar negativamente al hilo principal.

### 2.9.1. Node.js y el Manejo de Peticiones HTTP

Node.js es excelente para manejar peticiones HTTP, tanto para crear servidores web que responden a las solicitudes de los clientes (navegadores, otras aplicaciones) como para realizar peticiones a otros servidores (APIs externas).

- **Módulo `http` y `https`:** Proporcionan las bases para crear servidores y clientes HTTP/HTTPS de bajo nivel.
- **Frameworks (Express):** Simplifican la creación de servidores web con funcionalidades para enrutamiento, middleware y gestión de solicitudes/respuestas.

- **Clientes HTTP:** Para realizar peticiones a otros servidores, se pueden usar los módulos nativos (`http`, `https`) o bibliotecas de terceros más convenientes como `axios` o `node-fetch`.
- **Manejo de Métodos HTTP:** Node.js facilita el manejo de diferentes métodos HTTP (GET, POST, PUT, DELETE, etc.) para construir APIs RESTful.
- **Streams:** Node.js utiliza *streams* para manejar grandes cantidades de datos de manera eficiente durante las peticiones y respuestas, evitando cargar todo en la memoria.

### 2.9.2. Certificados

Para asegurar las comunicaciones a través de la web (HTTPS), Node.js permite trabajar con certificados SSL/TLS.

- **Módulo https:** El módulo `https` permite crear servidores HTTPS configurando las opciones del certificado (clave privada, certificado, certificados de autoridad CA si es necesario).
- **Obtención de Certificados:** Los certificados se pueden obtener de autoridades de certificación (CA) como *Let's Encrypt* (que ofrece certificados gratuitos) o de proveedores comerciales. También se pueden generar certificados autofirmados para entornos de desarrollo o pruebas (aunque no son confiables para producción).
- **Configuración del servidor:** Al crear un servidor HTTPS en Node.js, se deben especificar las rutas a los archivos del certificado y la clave privada en las opciones de configuración.
- **Manejo de Peticiones Seguras:** Una vez configurado el servidor HTTPS, Node.js manejará automáticamente el cifrado y descifrado de las comunicaciones.
- **Clientes HTTPS:** Al realizar peticiones HTTPS a otros servidores, Node.js maneja la verificación del certificado del servidor remoto (si está configurado correctamente con una CA de confianza).

## 2.10. Aplicaciones de inspiración al proyecto

El reconocimiento de voz y la creación de objetos está abriendo muchas posibilidades interesantes en diversas aplicaciones y juegos, incluyendo experiencias en gafas de realidad virtual (VR) y realidad aumentada (AR). Aquí te presento algunos ejemplos:

### 2.10.1. Arkio:

Arkio es una herramienta de diseño espacial colaborativo que permite a las personas crear y colaborar utilizando Realidad Virtual (VR), así como en PC, tablet y móvil.

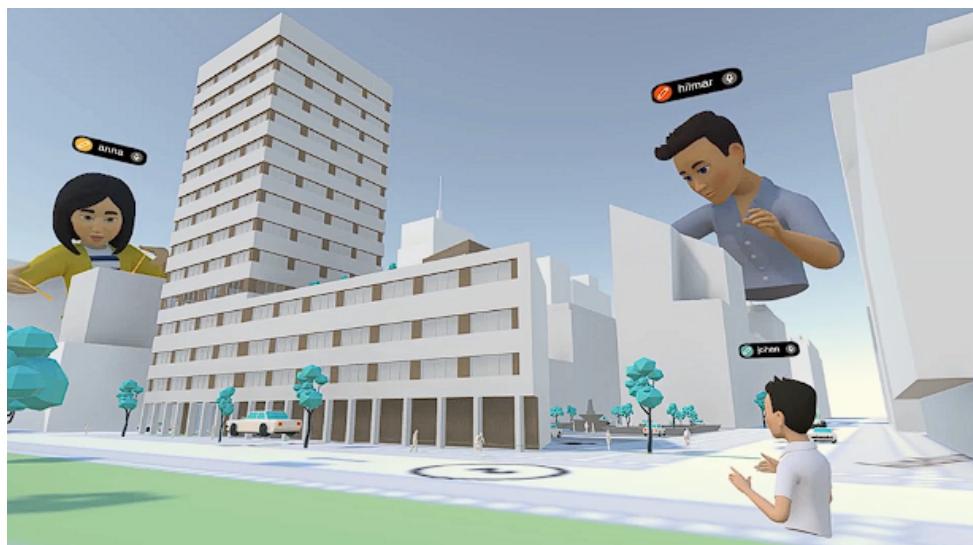


Figura 2.7: Ejemplo de escena de Arkio

- Se centra en el diseño de interiores, edificios, espacios virtuales y planificación urbana.
- Facilita la colaboración multiplataforma en tiempo real como se ve en la Figura 2.7.
- Dispone de plugins para integrar con herramientas de diseño 3D como Revit, Rhino y SketchUp, permitiendo importar modelos existentes y exportar el trabajo realizado en Arkio.
- Arkio permite a los usuarios dibujar y manipular elementos arquitectónicos de forma intuitiva en el espacio VR mediante el uso de los mandos de que traen las gafas como la Quest3.

### 2.10.2. VR Sculpting:

VR Sculpting con Comandos de Voz: En la escultura VR (como en Shapeland Figura 2.8 o Gravity Sketch), se han explorado o implementado (a menudo a través de plugins o herramientas de terceros como VoiceAttack) comandos de voz para tareas específicas durante el proceso de modelado. Estos comandos no forman parte nativa del software, pero permiten extender su funcionalidad, especialmente en contextos de accesibilidad, productividad o flujo de trabajo manos libres. Por ejemplo, podrías usar la voz para:

- Seleccionar herramientas.
- Modificar parámetros.
- Realizar acciones.

Si bien esto no es la creación del objeto desde cero con la voz, sí permite una manipulación y modificación significativa mediante comandos hablados.

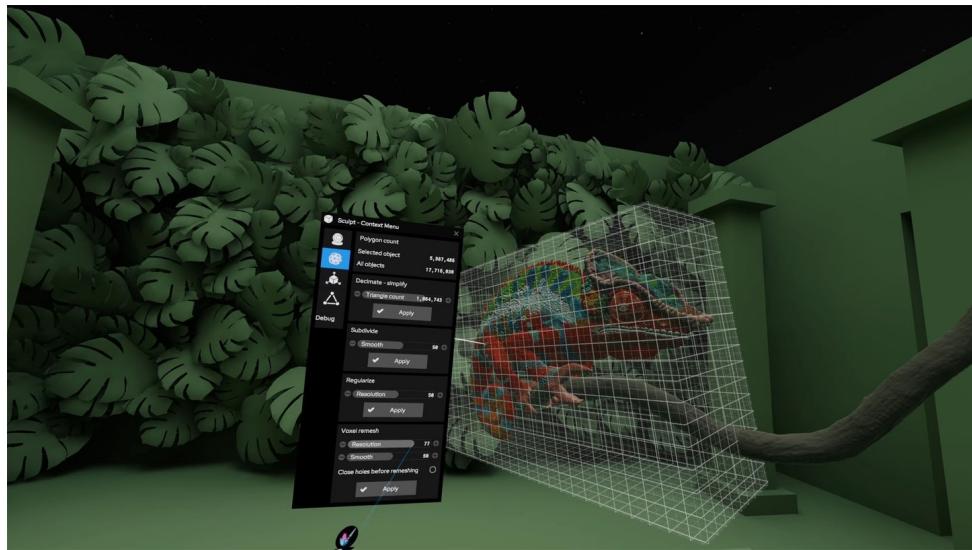


Figura 2.8: Ejemplo de uso de Shapeland

### 2.10.3. Bridge Crew (Star Trek)

Inicialmente, se añadió soporte para comandos de voz mediante la integración con IBM Watson, permitiendo a los jugadores emitir órdenes verbales a la tripulación en misiones para un solo jugador. Esta funcionalidad estaba disponible exclusivamente en inglés y requería el uso de un micrófono durante el juego, como se puede ver en la Figura 2.9. Ubisoft desactivó esta característica debido a la finalización del contrato con IBM, lo que resultó en la eliminación del soporte de comandos de voz en todas las plataformas.



Figura 2.9: Ejemplo de uso de Star Trek

Estas aplicaciones sirvieron como inspiración tanto por sus mecanismos de creación de objetos como por su capacidad para detectar y ejecutar comandos por voz. Este proyecto fue desarrollado con el objetivo de aprovechar esas funcionalidades, integrándolas en una sola herramienta accesible. El resultado en este proyecto busca combinar ambas facetas **creación y control por voz** en una plataforma accesible desde cualquier navegador web.

## 2.11. Tecnologías auxiliares

### 2.11.1. Github

GitHub se ha consolidado como la plataforma líder para el desarrollo colaborativo de software. Basándose en el sistema de control de versiones de Git, ofrece un entorno robusto para que desarrolladores de todo el mundo puedan trabajar simultáneamente en proyectos, gestionando los cambios de código de manera eficiente y organizada.

Su principal función radica en el alojamiento de repositorios, que actúan como almacenes centrales para el código fuente de un proyecto. Git permite rastrear cada modificación realizada en el código a lo largo del tiempo, facilitando la reversión a versiones anteriores, la identificación de la autoría de los cambios y la fusión del trabajo de diferentes colaboradores. Las pull requests (solicitudes de extracción) sirven como mecanismo para proponer cambios, iniciar discusiones sobre el código y, finalmente, integrarlos a la rama principal.

Gracias a estas funcionalidades se ha ido desarrollando poco a poco el proyecto, del que se está hablando en este documento. Si bien no se explotó toda la funcionalidad que Github aporta, ha sido de gran ayuda para la recuperación de versiones anteriores y guardar avances.

### 2.11.2. Visual Studio Code

Visual Studio Code es un editor de código fuente con un gran equilibrio entre ligereza y potencia. Se distingue por su rendimiento rápido y su capacidad para manejar proyectos de gran envergadura sin comprometer la fluidez. Una de sus características más destacadas es su soporte integrado para una amplia gama de lenguajes de programación, incluyendo JavaScript y Python entre otros.

Para cada lenguaje, VS Code ofrece funcionalidades específicas como el resaltado de sintaxis, que facilita la lectura del código, además de constar con un sistema de autocompletado inteligente que sugiere código, muestra información sobre funciones y parámetros, y ayuda a prevenir errores.

La depuración integrada es otra funcionalidad clave, permitiendo a los desarrolladores ejecutar y analizar su código directamente desde el editor. Se pueden establecer puntos de interrupción, inspeccionar el valor de las variables y seguir la ejecución paso a paso para identificar

y corregir errores de manera eficiente.

El Marketplace de extensiones ofrece una vasta colección de herramientas. En este proyecto se usaron extensiones como **Live Server** y **Live Preview** para visualizar los cambios en tiempo real que se hacen tanto en JavaScript como en cuerpo del archivo de A-Frame.

### 2.11.3. Gafas Meta Quest3

Las gafas de realidad virtual (VR) y realidad mixta (MR) de Meta. Ofrecen experiencias inmersivas en VR y la capacidad de superponer elementos virtuales en el mundo real (MR). Cuentan con mayor resolución, potencia y nuevas funcionalidades como passthrough a color de alta resolución y seguimiento de manos mejorado.



Figura 2.10: Gafas Meta Quest3

Se utilizaron para hacer las pruebas de funcionamiento de la aplicación en los navegadores de Meta, pero contaba con ligeros problemas de compatibilidad para el uso de APIs que se usan en navegadores normales.

#### 2.11.4. LaTeX

LaTeX, por otro lado, es un sistema de composición de textos de alta calidad, ideal para documentos técnicos. Se centra en la estructura y el contenido, dejando el formato en manos del sistema.

LaTeX automatiza la generación de elementos estructurales como la numeración de secciones, subsecciones, figuras y tablas. También se encarga de la creación automática de índices, bibliografías y referencias cruzadas, asegurando la coherencia y precisión en todo el documento. La salida final se caracteriza por su alta calidad tipográfica, gracias a los algoritmos de composición y al uso de fuentes profesionales.

Es usado para el desarrollo de la memoria del trabajo de fin de grado. Se tomó esta elección debido a que permite centrarse únicamente en el contenido textual, mientras que el sistema se encarga del diseño y formato del documento.

# **Capítulo 3**

## **Desarrollo del proyecto**

A este proceso se aplicaron estrategias basadas en metodologías ágiles. Para ser más preciso, se usó el método de Scrum para organizar el trabajo, un marco que facilita la gestión de proyectos complejos mediante ciclos iterativos conocidos como sprints. Hasta que no se consigue el objetivo de un sprint, no comienza el otro, y la consecución de los objetivos de todos los sprints coincide con la consecución del objetivo final del proyecto. Estos sprints a su vez se dividen en tareas mucho más pequeñas que persiguen objetivos más concretos y ayudan a la consecución del objetivo final del sprint.

Estructura mínima de cada sprint:

- Objetivos
- Tareas Realizadas
- Resultado
- Lecciones aprendidas

### 3.1. Arquitectura general

#### ■ Sprint 1: Investigación y Prototipado Inicial (3 semanas)

- Investigación sobre uso, posible desarrollo y función de los objetos relacionados con el proyecto.
- Desarrollo e investigación de reconocimiento de voz.
- Desarrollo de la primera demo sobre funcionamiento de A-FRAME.

#### ■ Sprint 2: Desarrollo de A-FRAME Base (2 semanas)

- Creación de objetos de A-FRAME simples.
- Creación de componentes para A-FRAME y reestructuración de código.

#### ■ Sprint 3: Funcionalidades y Componentes (2 semanas)

- Creación de funcionalidades para componentes.
- Aumento de comandos para la funcionalidad del proyecto.
- Creación de escena simple para funcionalidades escritorio

#### ■ Sprint 4: Mejora de componentes y Mejora Visual (2 semanas)

- Crear componentes para editar y eliminar objetos.
- Mejora de la estética y colocación de objetos.
- Crear funcionalidad para nuevos componentes.

#### ■ Sprint 5: integración en VR/AR (1 semana)

- Posibilidad de integración en VR/AR (Quest3).
- Creación de servidor privado.

#### ■ Sprint 6: Pasos finales (3 semanas)

- Mejora la ética y posicionamiento de objetos en escena.
- Implementación de plugins extra para la visualización y decoraciones.
- Maquetado final de la demo para escritorio.
- Maquetado y unión del back-end y front-end para demo con gafas Quest3.

## 3.2. Sprint 1

**Investigación y Prototipado Inicial** El desarrollo de este primer Sprint tuvo una duración de 3 semanas, debido a que fue la primera toma de contacto con el servicio de reconocimiento de voz y del uso complejo en A-Frame. Además, contamos con diferentes programas que se fueron desarrollando en el transcurso de este sprint. Estos programas se irán explicando a continuación.

### 3.2.1. Objetivos

Los objetivos principales eran la recolección de información, datos y ejemplos de aplicaciones que pudieran ayudar al desarrollo de este proyecto, además de familiarizarnos más con los diferentes programas que se usarán en el desarrollo.

- Investigación del posible desarrollo, uso y funcionamiento de las tecnologías implicadas en el desarrollo del proyecto.
- investigación y desarrollo de ejemplos para reconocimiento de voz.
- Desarrollo de la primera demo sobre funcionamiento de A-FRAME.

### 3.2.2. Tareas Realizadas

- Gestionar información, ver ejemplos (si existen), buscar demos similares.
- Aplicar tecnologías de reconocimiento de voz y posibles usos.
- Desarrollo de demos iniciales para validar el uso de A-Frame con contenido dinámico.

#### Aplicación de tecnologías de reconocimiento de voz

Tras la recolección y análisis de información técnica relevante. Para ello, se harán pruebas con distintos sistemas de transcripción de voz. Y finalmente, se desarrollaron 3 ejemplos de cómo debía realizarse la funcionalidad para generar una transcripción.

##### 1. Reconocimiento de Voz en Python con Whisper

Se utilizó el modelo **Whisper** de OpenAI en **local** para transcripción de audio. Este programa se llevó a cabo mediante Python, grabando la voz mediante el micrófono, y guardando el audio para luego usar Whisper en él y obtener una transcripción.

## 2. Servidor híbrido Node.js + Python

Para el segundo caso, usamos un servidor propio creado a base de Node.js que se comunicaba con un script de Python en tiempo real para realizar las transcripciones del mismo modo que en el caso anterior. Este caso se desarrolló para poder implementar la transcripción de voz en el navegador y desarrollar la idea principal del proyecto. La transcripción obtenida se transmitiría al cliente web a través de WebSockets.<sup>1</sup>

## 3. Reconocimiento en el navegador (Web Speech API)

Como caso de uso final se encontró una API dedicada a los navegadores, Web Speech API dio paso a la transcripción rápida y efectiva. Es necesario recalcar que depende mucho de la calidad de audio, y el navegador. Dio unos buenos resultados, además de que, en el contexto del proyecto, cumplía los estándares que se intentaban alcanzar con el reconocimiento y transcripción de voz.

### Contacto con A-Frame y reconocimiento de voz

También se planteó el primer contacto con el framework haciendo unas demos sencillas en las que se dará uso a un `<a-text>` para mostrar texto dinámicamente. **Demo – Reconocimiento de Voz + A-Frame**

En esta demo se trabajó con el reconocimiento de voz desde el navegador (usando ‘Web Speech API’). De la tal manera que al pulsar el botón principal que aparece en la Figura 3.1 se activa el evento de detección de voz que obtendrá permisos para el uso del micrófono y este al detectar al usuario hablar transcribe la voz, todo esto dentro del mismo navegador, gracias al componente **’input-text’**. Tras la obtención de la transcripción, esta aparece automáticamente en el elemento `<a-text>` dentro de la escena. La experiencia permite mostrar mensajes de voz directamente dentro de un entorno 3D. Si se detectan palabras como crear, editar u otro texto, el fondo cambia de color<sup>2</sup>

---

<sup>1</sup>Ejemplo disponible en GitHub: [https://github.com/R4CC00N/SpeechRecognition\\_in\\_Browser/blob/main/Sprint2/Python/prueba\\_whisper.py](https://github.com/R4CC00N/SpeechRecognition_in_Browser/blob/main/Sprint2/Python/prueba_whisper.py).

<sup>2</sup>Demo disponible en GitHub: [https://r4cc00n.github.io/SpeechRecognition\\_in\\_Browser/Sprint3/v2\\_A-Frame.html](https://r4cc00n.github.io/SpeechRecognition_in_Browser/Sprint3/v2_A-Frame.html).

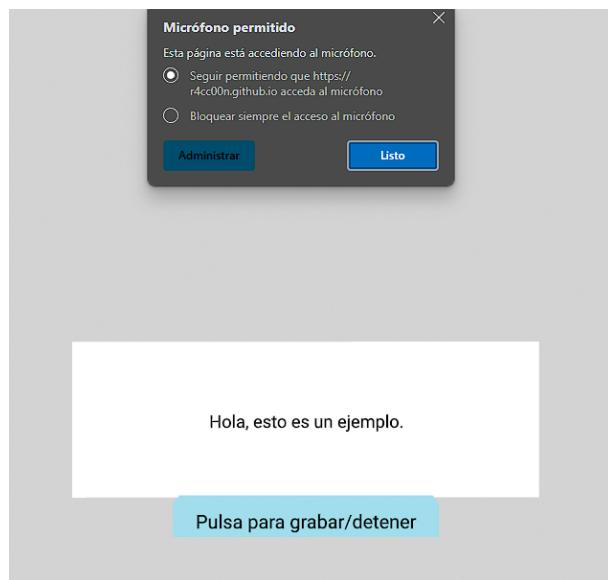


Figura 3.1: Funcionamiento de la segunda Demo para A-Frame

### 3.2.3. Resultados

**Web Speech API:** es la solución más óptima para navegadores, aunque limitada en dispositivos VR se terminará usando en las demás demos.

**Demo - Reconocimiento de Voz + A-Frame:** Una vez conocidos los elementos a interactuar, el añadir un factor como el reconocimiento de voz fue un desafío de grado medio. Se encontró que la transcripción podía ser usada directamente para el cambio de valor dentro del <a-text>.

### 3.2.4. Lecciones aprendidas

- Se concluyó que la opción **Web Speech API** fue la más eficiente y directa para pruebas en escritorio. También tuvo varias cosas negativas, la principal fue que en el uso de la detección y transcripción de voz en gafas de VR/AR, como son las Quest3, no fueron soportadas en el navegador de Meta.
- No obstante, esta incompatibilidad de **Web Speech API** con navegadores de gafas como la Quest3 llevó a plantear dos líneas de trabajo: una demo basada en navegador para escritorio, y una segunda demo soportada por back-end para su funcionamiento en gafas de realidad mixta, empleando Node.js y alguna API de reconocimiento de voz en el servidor.

### 3.3. Sprint 2

Este Sprint tuvo una duración de 2 semanas, además contiene **demos interactivas** en las que se combina **A-Frame** con **reconocimiento de voz** para crear objetos 3D dentro de una escena usando comandos hablados. Cada demo mejora la anterior, especialmente en el reconocimiento y manejo de **coordenadas y números**, permitiendo finalmente la creación precisa de objetos con posición y tamaño definidos por voz. Además del incremento y mejoras en los **componentes** de A-Frame.

#### 3.3.1. Objetivos

- Implementación básica de reconocimiento de comandos de voz en una escena de A-Frame.
- Permitir la creación de objetos primitivos mediante comandos de voz.
- Implementar la capacidad de interpretar números en cifra tanto positivos como negativos.
- creación de componentes para A-FRAME y reestructuración de código.

#### 3.3.2. Tareas Realizadas

##### Demo - Comando Básico y Reconocimiento Inicial

En esta demo se creó un componente llamado '**input-text**' que, aparte de transcribir el audio, también detectará comandos de voz. Estos comandos serán procesados de la transcripción, como se puede ver en la Figura 3.2. Trabajamos principalmente con el comando 'crear [objeto]', este permite la generación de cubos, solicitando a continuación la '**posición**' y el '**tamaño**' del objeto.

Sin embargo, presenta ciertas limitaciones, una de ellas es la fiabilidad en el reconocimiento numérico (valores numéricos que la transcripción decide automáticamente poner como letras e incapacidad para interpretar valores números negativos en las transcripciones). Otra de las limitaciones que encontramos es la captura de coordenadas como un bloque único

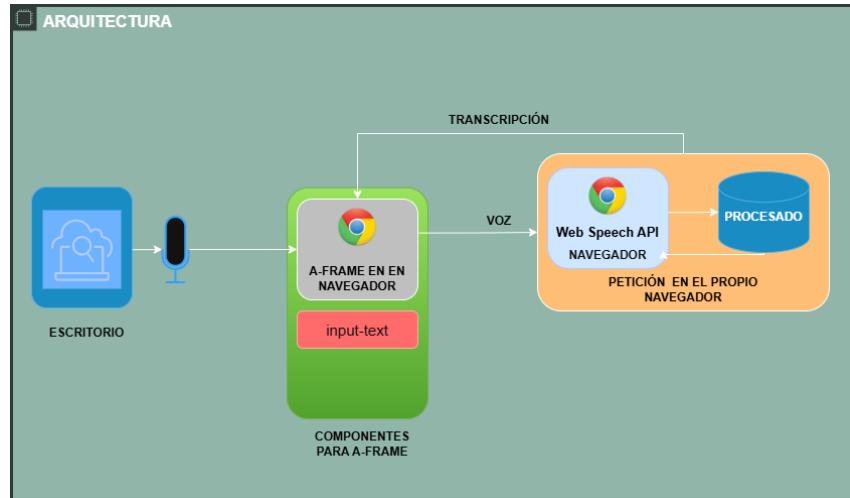


Figura 3.2: Flujo de creación de componentes

### Demo Final - Creación de Componentes en A-Frame y Reestructuración de Código

Este sprint se centró en modularizar la escena en A-Frame mediante la creación de componentes reutilizables como se ve en la Figura 3.3, en ella se puede ver:

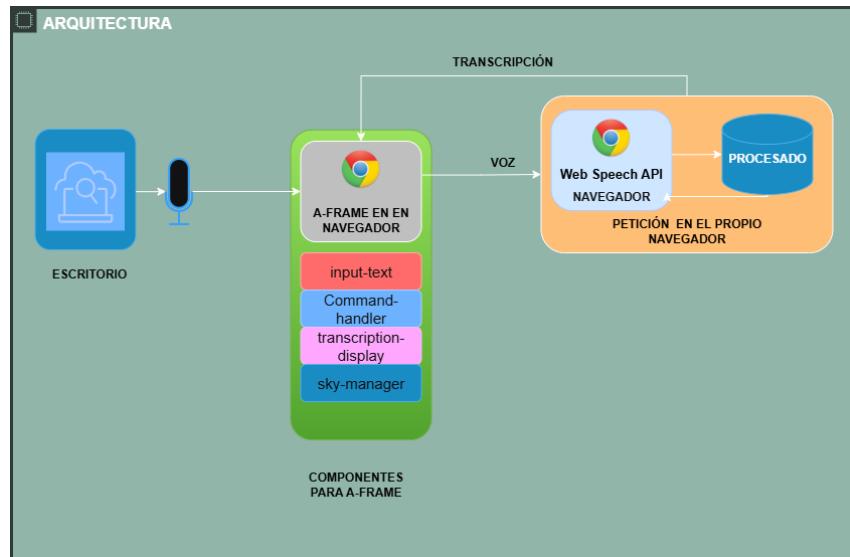


Figura 3.3: Flujo de creación de componentes avanzados

**command-handler:** Es un componente que interpreta comandos de voz para crear objetos en la escena, pero con una entrada de audio configurable. Además de aplicar una reestructuración del código con el fin de mejorar la organización, escalabilidad y mantenibilidad del proyecto. Depende de que otro componente en la escena realice el reconocimiento de voz y

emita un evento 'transcript' con el texto transcrita. Durante su funcionamiento, emite eventos a nivel de escena (`enter-create-mode`, `exit-create-mode`) para señalar el inicio y el fin del proceso de creación.

**transcription-display:** Tiene la función de mostrar en un elemento de texto 3D con un **ID** definido (`<a-text>`) la transcripción de voz proveniente de otro componente de la escena. Se puede observar su uso en la Figura 3.4 en donde aparecen los mensajes para el usuario en la parte superior de la caja de texto normal.

**sky-manager:** Se encarga de gestionar el color del fondo de la escena (`<a-sky>`) en respuesta a eventos personalizados emitidos a nivel de la escena provenientes del **command-handler**.

Añadido a estos componentes, decidimos deparar el código de **javascript** y el **HTML**, para organizar así mejor los archivos y la estructuración del proyecto. También se han solucionado los problemas con los números creando un manejador de números.

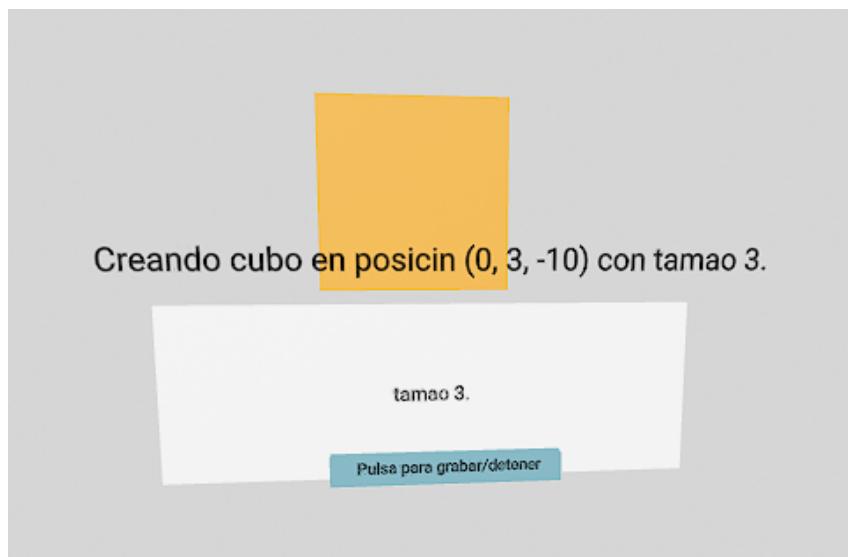


Figura 3.4: Funcionamiento del comando basico Demo 4

### 3.3.3. Resultados

- Se logró una implementación funcional del reconocimiento de voz para la creación de objetos 3D en un entorno A-Frame.
- Se demostró una mejora progresiva en la capacidad del sistema para comprender y procesar comandos de voz relacionados con la creación, posición y tamaño de objetos.

- La interacción para especificar la posición de los objetos evolucionó de una entrada de bloque (123) a una solicitud secuencial ( $x=1,y=2,z=3$ ), mejorando la usabilidad.
- En la última demo, se implementó con éxito la interpretación de números tanto en formato numérico como en palabras, incluyendo valores negativos, lo que amplía la flexibilidad de la entrada por voz.
- Se proporcionó retroalimentación visual constante al usuario sobre el estado de la grabación y el texto reconocido a través de la interfaz en la escena.
- Se mejoró la legibilidad del texto al separar en diferentes archivos el JavaScript y el HTML.

### 3.3.4. Lecciones aprendidas

- La creación de componentes en A-Frame facilita la organización y la reutilización de la lógica de la aplicación.
- La interpretación del lenguaje natural, incluso para tareas específicas, requiere una consideración cuidadosa de las posibles formas en que los usuarios pueden expresar sus mandos. El manejador para números es un ejemplo de cómo abordar esta complejidad.
- La retroalimentación visual es esencial para mantener al usuario informado y para depurar el proceso de reconocimiento e interpretación de voz.
- La iteración y la mejora progresiva, como se ve en la secuencia de los cuatro demos, es un enfoque efectivo para abordar desafíos técnicos.

## 3.4. Sprint 3

Este sprint tuvo una duración de 2 semanas, en donde se desarrolló las capacidades del proyecto, añadiendo más modificadores, creando componentes e incluso mejorando la división del código. Durante el desarrollo de este Sprint se crearon componentes para el accionar del proyecto.

### 3.4.1. Objetivos

- Creación de funcionalidades para componentes.
- Aumento de comandos para la funcionalidad del proyecto.
- Creación de escena simple para funcionalidad en escritorio.

### 3.4.2. Tareas Realizadas

#### Demo - Creador de objetos funcionalidades

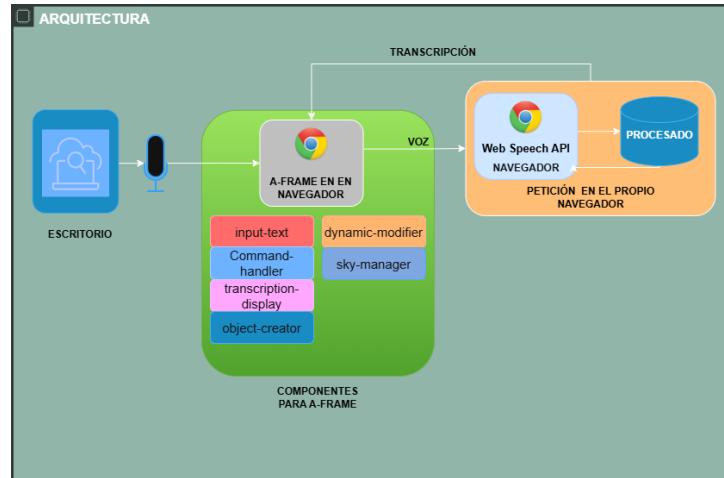


Figura 3.5: Flujo de creación de componentes avanzados

**object-creator:** Escucha el evento **start-object-creation** y, basándose en el tipo de objeto recibido, crea la entidad con una geometría predefinida, una posición y rotación fijas.

**dynamic-modifier:** Este componente edita en tiempo real el objeto creado con diferentes funcionalidades de modificación Paso a Paso (**stepsPOS**, **stepCOLOR**, **stepID**, **stepSIZE**) para manejar la modificación de cada atributo de forma conversacional.

Se introduce una variable global **step** (initializada en null). Esta variable se utiliza para gestionar un flujo de conversación paso a paso durante la modificación de los atributos (posición, tamaño, color, ID).

La palabra clave '**cambio**' ahora se utiliza para salir del modo de modificación. Este cambio envía un mensaje al usuario que se utiliza para guiarlo a través del proceso de modificación.

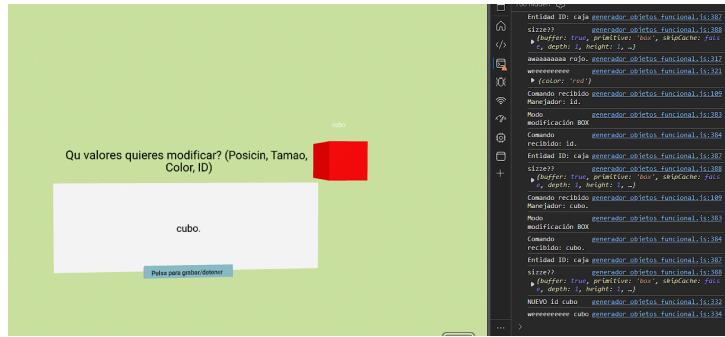


Figura 3.6: Funcionamiento del generador de objetos con funcionalidades extra

### 3.4.3. Resultados

Este sprint implementó la interacción por voz para la gestión de objetos:

Se creó el componente **object-creator** para crear objetos básicos, el cual cumple relativamente bien su función.

Además, se implementó la **modificación conversacional** paso a paso de posición, color, ID y tamaño durante la creación del objeto.

La mayoría de estos componentes se mantendrán hasta la etapa final, solventando algunos fallos que pudieran encontrarse en ellos durante el desarrollo.

### 3.4.4. Lecciones aprendidas

**Componentes para modularidad:** Organizar la lógica en componentes facilita el desarrollo y mantenimiento.

**Gestión de estados en voz:** Las interacciones conversacionales requieren un seguimiento claro del estado.

**Asincronía de la voz:** La transcripción y respuesta necesitan manejar la latencia.

**Comandos de voz intuitivos:** El lenguaje claro y la guía mejoran la experiencia del usuario.

## 3.5. Sprint 4

Este Sprint tuvo una duración de 2 semanas, en donde se dieron la mayoría de los avances en creación de componentes, como se puede ver en la Figura 3.7 .

### 3.5.1. Objetivos

- Crear componentes para editar y eliminar objetos creados.
- Modificar la escena para mejor uso del usuario.
- Crear comandos nuevos para la funcionalidad de los nuevos componentes.
- Mejora de la estética y colocación de objetos.

### 3.5.2. Tareas Realizadas

#### Demo - Nuevas funcionalidades

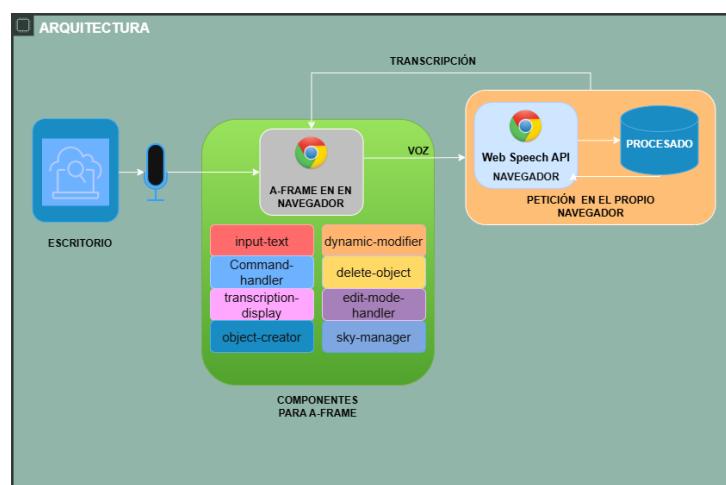


Figura 3.7: Flujo de creación de componentes avanzados

Esta demo es la más completa de todas, agrupamos la funcionalidad presentada anteriormente y añadiendo unas mejoras de las que hablaremos a continuación. Primero se añadió la lógica para crear diferentes tipos de objetos (cubo, esfera, plano, luz, y assets como contenedor y radio) al recibir comandos de voz. Se agregaron nuevos componentes con funcionalidades diversas:

- **delete-object:** Este componente maneja la lógica para eliminar objetos de la escena mediante comandos de voz, solicitando el ID del objeto a eliminar como se ve en la Figura 4.2.

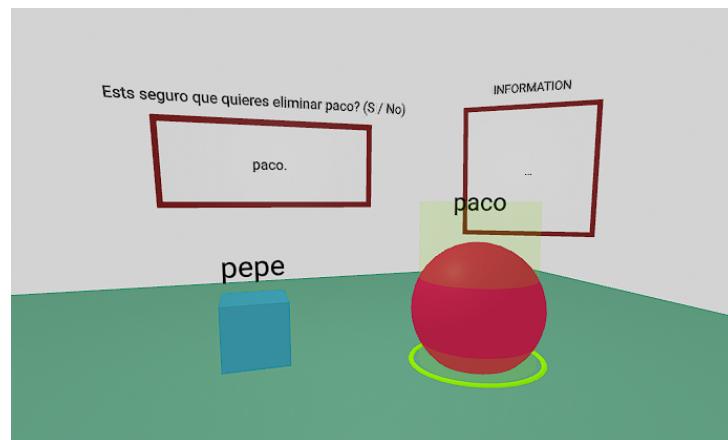


Figura 3.8: Funcionamiento de eliminar objetos con ID

- **edit-mode-handler:** Este componente gestiona el modo de edición de objetos existentes en la escena, según el ID del objeto, como se ve en la Figura 4.3.

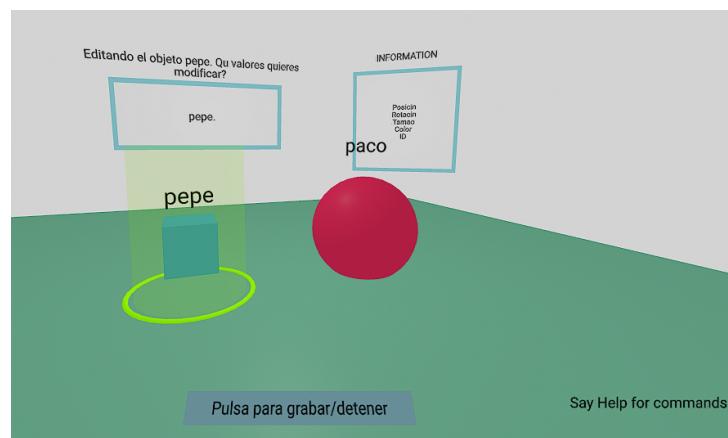


Figura 3.9: Funcionamiento de editar objetos con ID

- **object-creator:** Este componente se encarga de instanciar los diferentes tipos de objetos en la escena basándose en los eventos recibidos, como se ve en la Figura 4.4.

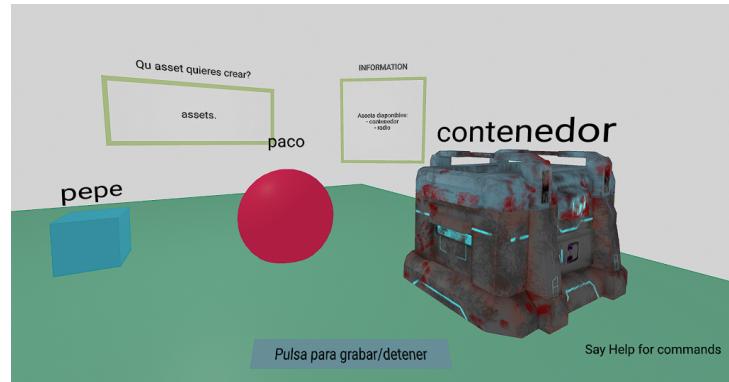


Figura 3.10: Funcionamiento del generador de assets

Además, se definen varias funciones utilitarias (**updateUserMessage**, **stepsPOS**, **stepsROT**, **stepCOLOR**, **stepID**, **stepSIZE**, **getEntityById**, **getLastCreatedEntity**, **createTorus**) para facilitar la manipulación de la escena y los objetos.

También se cambiaron y agregaron comandos nuevos al uso del **command-handler** así como **crear**, **editar**, **ayuda**, **salir**. Se añadieron paneles nuevos para añadir información extra que puede ser útil al usuario tanto escrita como visual, como se puede ver en los bordes de los paneles que cambian de color, dependiendo del estado en el que se encuentre la aplicación (verde=crear, rojo=eliminar ...).

### 3.5.3. Resultados

**Gestión de nuevos comandos:** Integración de creación, edición y eliminación por voz mediante componentes dedicados (delete-object, edit-mode-handler, object-creator) y modificación avanzada con dynamic-modifier.

**Cambio en la estética y colocación de objetos:** Se modificó el HUD para hacerlo más agradable, visualmente y de ayuda para el usuario, creando así una escena sencilla.

### 3.5.4. Lecciones aprendidas

**Identificación de objetos:** Es crucial darle un nombre a los objetos para luego poder dar uso de las funciones de edición y eliminación específicas.

**Reutilización de código:** Las funciones utilitarias evitan la repetición.

**Retroalimentación al usuario:** Informar sobre los comandos y el estado es esencial.

## 3.6. Sprint 5

La duración de este sprint fue de una semana. Durante este periodo se identificaron varios fallos críticos que llevaron a una reestructuración significativa del flujo de trabajo inicial.

### 3.6.1. Objetivos

- Explorar la posibilidad de integración del proyecto con dispositivos VR/AR, específicamente las gafas Quest 3.
- Creación de servidor propio para peticiones de transcripción.

### 3.6.2. Tareas Realizadas

Durante este proceso se encontró el fallo más grande. En primer lugar, se quería realizar este proyecto íntegro en gafas VR/AR Quest3. Pero la API de reconocimiento de voz integrada en navegadores no fue soportada por el navegador de **Meta**, entonces se volvió a replantear todo lo trabajado con anterioridad. Probamos con tecnologías diferentes de reconocimiento de voz, refactorizamos el componente que se encargaba de reconocimiento de voz dentro de nuestro proyecto de A-Frame. Pero no se encontró ninguna solución, ya que no soportaba la detección del audio.

Por ello, se creó un servidor privado que fuese capaz de manejar los audios obtenidos del cliente y luego procesarlos y mandarlos a una API de **Speech-to-text** llamada Assembly AI, que devolverá la transcripción al servidor y este finalmente al cliente por medio del servidor. Esta fue la parte desafiante del sprint. Ya se había manejado un servidor con anterioridad, pero este manejaba unos procesos simples y todo ocurría en el mismo ordenador, pero a la hora de conectarse otros dispositivos, dependía de permisos y certificaciones para hacer esta conexión posible.

La implementación de un servidor HTTPS funcional presentó un nuevo reto, ya que se necesitaba garantizar la conexión desde otros dispositivos dentro de la red local. Para ello, se abordaron los siguientes aspectos técnicos:

-**Certificado SSL:SSL/server.key**(clave privada) y **SSL/server.crt**(certificado público). En los navegadores aparecerá como “no seguro” si es autofirmado.

#### -Configurar un servidor HTTPS

**-Acceso desde otros dispositivos en red local:** Ahora la conexión deberá hacerse al servidor, entonces en la URL deberemos apuntar a la IP del servidor que esté en la misma conexión de internet y con el puerto correspondiente(ej. <https://IP:8080/>)

### 3.6.3. Resultados

Tras la creación de certificados y configuraciones, se pudo conectar el servidor con las gafas de VR/AR Quest3. El problema era la detección del micrófono, que solo se hacía si el servidor tenía los permisos para HTTPS.

Como resultado, se consiguió la interacción completa esperada: el cliente envía el audio al servidor, este lo reenvía a Assembly AI para su transcripción, y finalmente el texto transcrita es devuelto al cliente para su visualización.

También hay que recalcar que para el uso de algunos complementos nos tocó usar una versión de A-Frame pasada, para poder usar el desplazamiento por la escena.

### 3.6.4. Lecciones aprendidas

- Las limitaciones de plataformas específicas (como navegadores embebidos en dispositivos VR) pueden requerir rediseños profundos en la arquitectura del sistema.
- Contar con un servidor propio proporciona flexibilidad y control, pero también exige cumplir con requisitos de seguridad como HTTPS para trabajar con APIs modernas o acceder a funciones sensibles como el micrófono.
- La comunicación segura entre dispositivos en red local es posible incluso con certificados autofirmados, siempre y cuando se configuren correctamente.

## 3.7. Sprint 6

Este Sprint duró 3 semanas, en donde se llevaron a cabo los últimos ajustes para la implementación final tanto en escritorio como para gafas VR/AR Quest3.

### 3.7.1. Objetivos

- Mejora de la estética y colocación de objetos.
- Implementación de plugins extra para la visualización y decoraciones.
- Maquetado final de la demo para escritorio.
- Maquetado y unión del back-end y front-end para demo con gafas Quest3.

### 3.7.2. Tareas Realizadas

Se mantuvo la posición de los objetos que dan información, además de agregar un plugin para que el panel completo siempre mirase a la cámara. Esto se realizó con un **look-at**.

Durante este sprint se decidió modificar un poco la visualización de los paneles y optamos por usar un plugin llamado **html embed** que permite incrustar HTML en los paneles de A-Frame, quedando de la forma que se ve en la Figura 3.11



Figura 3.11: Funcionamiento de htmdembed

Durante este sprint se añadieron también nuevos componentes al funcionamiento del proyecto, unos para la creación de objetos que llevasen otros componentes incrustados, de tal manera que con un solo componente se pudiese estructurar todo. Por otro lado, también se agregaron componentes para nuevos comandos tales como **Guardar** y **Cargar** escenas Figura 3.12.

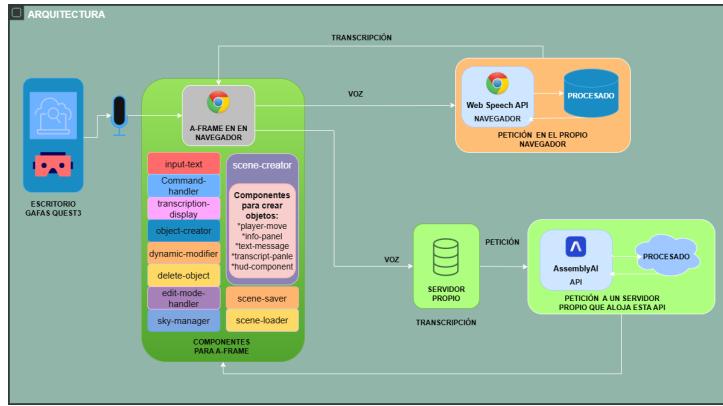


Figura 3.12: componentes finales agregados

Finalmente, se hizo la inclusión de las demos para escritorio y gafas Quest3 en donde el cambio más significativo entre ellas es el componente '**input-text**', el cual en la versión de escritorio usa **WebSpeechAPI** y en la versión de AR/VR usamos un servidor conectado a **AssemblyAI**.

### 3.7.3. Resultados

El desarrollo de componentes reutilizables mejoró la modularidad del sistema, permitiendo construir objetos más complejos con menos código y mayor coherencia visual.

También se logró una integración efectiva de entrada por voz, adaptada según la demo, lo que permitió mantener una experiencia coherente en ambos entornos.

### 3.7.4. Lecciones aprendidas

Una de las principales lecciones fue la importancia de diseñar con enfoque multiplataforma desde el inicio, considerando las capacidades y limitaciones propias de cada entorno.

Adaptar interfaces y métodos de interacción según el dispositivo resultó crucial para garantizar una experiencia fluida y natural.

También se valoró el uso de herramientas externas y plugins, que ampliaron las posibilidades de diseño y funcionalidad sin comprometer el rendimiento.

Finalmente, se reafirmó la importancia de mantener una arquitectura limpia y desacoplada entre el front-end y el back-end, lo que facilitó la integración final de las demos.

# **Capítulo 4**

## **Resultados**

**COMO SE USA AL USUARIO:** con capturas de pantalla. . . **DESCRIPCION TECNICA PARA MANTENERLO A UN IGENIERO**

- Que componentes que hay y como estan hechos
- como se relacionan los componentes
- que apis usan y como se utilizando
- el diagrama de uso..
- lo coge el component lo pasa a donde sea y lo devuelve. . .

**USO DE COMPONENTES PARA CONSTRUIR DISTINTAS APLICACIONES**

- COMPONENTES QUE SIRVEN PARA X cosas
- SON reutilizables
- MODULAR Y COMO USARLA.
- COMO COMPONER LOS COMPONENTES.

## 4.1. Flujo de funcionamiento

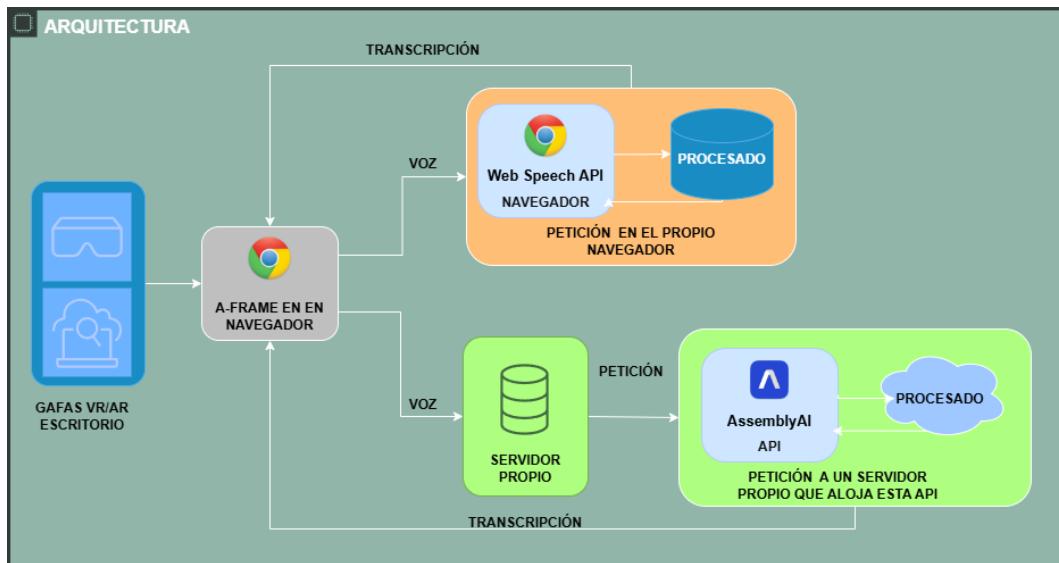


Figura 4.1: Estructura del funcionamiento del Proyecto

El diagrama de la Figura 4.1 muestra dos arquitecturas distintas para transcribir voz en una aplicación web VR/AR construida con A-Frame. En la primera demo, la transcripción se realiza directamente en el navegador usando la Web Speech API. Esto permite capturar la voz y procesarla sin necesidad de servidores externos, lo cual es rápido pero depende del soporte del navegador. En la segunda demo, la voz se envía a un servidor propio, que luego la reenvía a la API de AssemblyAI para su transcripción. Esta opción ofrece mayor precisión y capacidades avanzadas, aunque requiere una infraestructura adicional y puede tener más latencia.

se necesita el uso del 'NOMBRE DEL-.js' para el uso en html con la llamada a el scrpit y poner el componente en una entidad de la scena... ademas de un componente que almacene los objetos llamado 'xxx'

\*\* SEGUIR CON ESTO \*\*

Descripción para usuario (manual de usuario) Descripción de la implementación

## 4.2. Componentes

### 4.2.1. dynamic-modifier

- **stepsPOS(posKey, entity, transcript):** Permite modificar las coordenadas X, Y, Z de la posición en pasos. Al decir 'posición', se pregunta por la 'x', luego 'y', luego 'z'.
- **stepCOLOR(entity, transcript):** Intenta encontrar un color válido en el transcript utilizando un mapa (colorsMap) y actualiza el color del material del objeto.
- **stepID(entity, transcript):** Permite cambiar el id de la entidad y también actualiza el texto que se muestra sobre el objeto.
- **stepSIZE(entity, transcript):** Intenta extraer un número del transcript y lo aplica como un nuevo tamaño a todos los parámetros de la geometría del objeto (excepto primitive y otros parámetros específicos).

MEDICIONES DE TIEMPO PARA LAS RESPUESTAS, Y LOS CAMBIOS EN LA ESCENA ... **command-handler:** Es un componente que interpreta comandos de voz para crear objetos en la escena, pero con una entrada de audio configurable. además de aplicar una reestructuración del código con el fin de mejorar la organización, escalabilidad y mantenibilidad del proyecto. Depende de que otro componente en la escena realice el reconocimiento de voz y emita un evento transcription con el texto transcritto. Durante su funcionamiento emite eventos a nivel de escena (enter-create-mode, exit-create-mode) para señalar el inicio y el fin del proceso de creación. **transcription-display:** Tiene la función de mostrar en un elemento de texto 3D con un **ID** definido (<a-text>) la transcripción de voz proveniente de otro componente de la escena. Se puede observar su uso en la Figura 3.4 en donde aparece los mensajes para el ususario en la parte superior de la caja de texto normal.

**sky-manager:** Se encarga de gestionar el color del fondo de la escena (*a-sky*) en respuesta a eventos personalizados emitidos a nivel de la escena provenientes del **command-handler**.

- **delete-object:**Este componente maneja la lógica para eliminar objetos de la escena mediante comandos de voz.

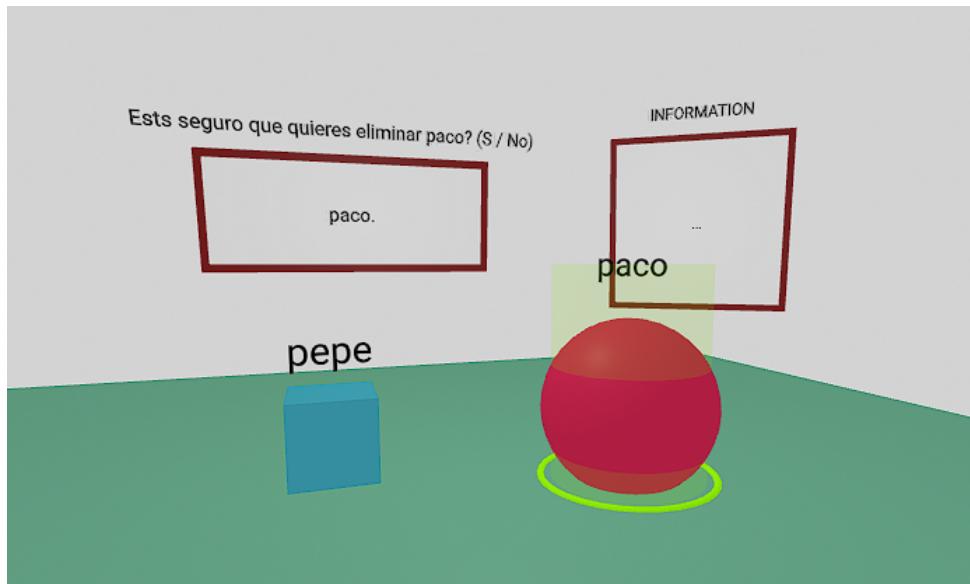


Figura 4.2: Funcionamiento de eliminar objetos con ID

Escucha el evento transcription. Pide al usuario el ID del objeto a eliminar. Se visualiza un torus alrededor del objeto seleccionado para confirmar la eliminación. Permite confirmar o cancelar la eliminación mediante los comandos 'sí' o 'no'. Emite un mensaje de confirmación o cancelación al usuario. Permite salir del modo de eliminación con los comandos 'salir', 'atrás' o 'cambio'.

- **edit-mode-handler:** Este componente gestiona el modo de edición de objetos existentes en la escena.

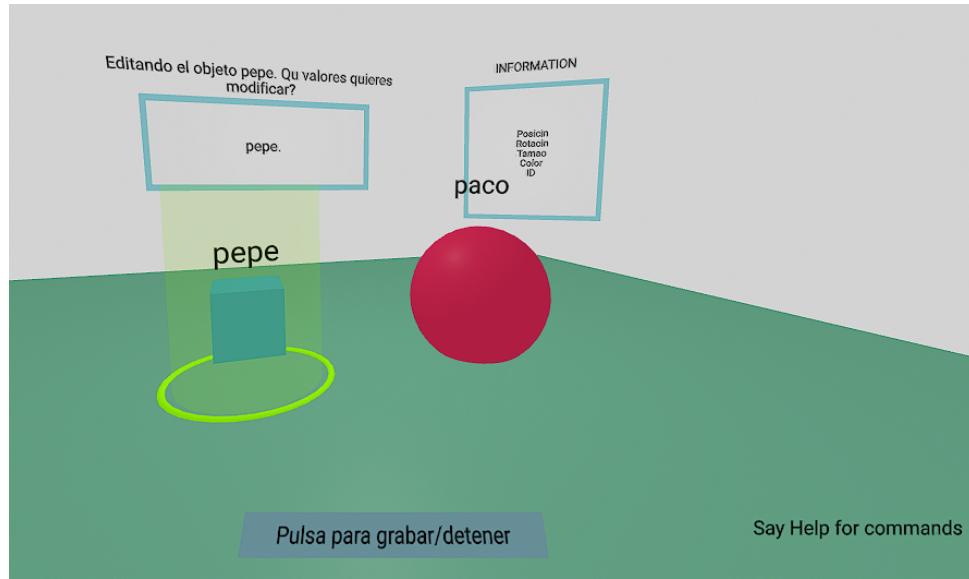


Figura 4.3: Funcionamiento de editar objetos con ID

Escucha el evento transcription. Pide al usuario el ID del objeto a editar. Busca el objeto por su ID y visualiza un torus alrededor para indicar que está en modo de edición. Utiliza la misma lógica de pasos y palabras clave que dynamic-modifier para permitir la modificación de las propiedades del objeto seleccionado. Elimina el torus al salir del modo de edición (“cambio”).

- **object-creator:** Este componente se encarga de instanciar los diferentes tipos de objetos en la escena basándose en los eventos recibidos.

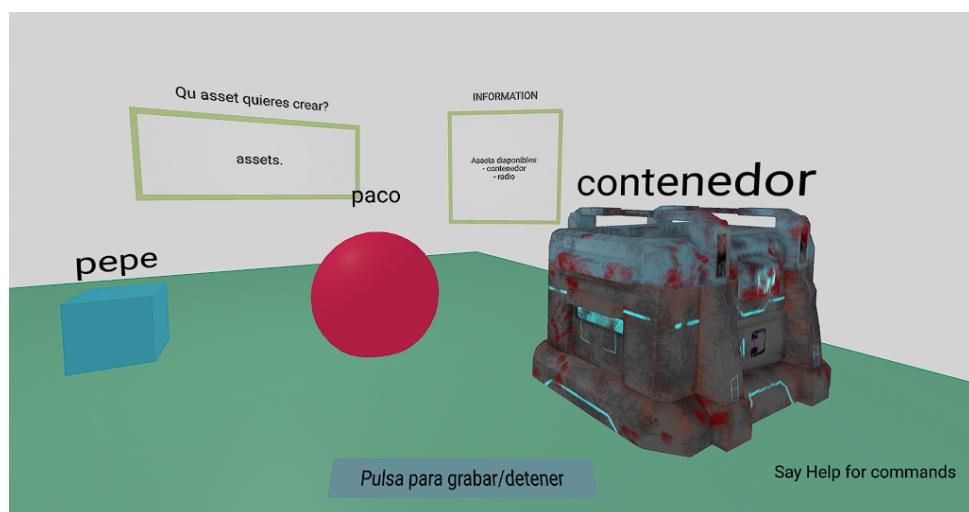


Figura 4.4: Funcionamiento del generador de assets

Escucha el evento start-object-creation. Define los datos de creación para cada tipo de objeto (cubo, esfera, plano, luz, contenedor, radio) en formato JSON. Crea los elementos a-entity correspondientes con los atributos definidos (geometría, posición, material, etc.). Añade una clase 'dynamic-object' a los objetos creados. Crea una etiqueta de texto (aunque la función createText parece estar vacía en este fragmento). Establece las variables booleanas (modifyingBox, modifyingSphere, etc.) para indicar qué tipo de objeto se está modificando.

# Capítulo 5

## Experimentos y validación

Con el fin de validar la funcionalidad del sistema desarrollado '**una herramienta para la creación de objetos en entornos 3D mediante comandos de voz en A-Frame**' se realizó una serie de experimentos centrados en evaluar tanto el correcto reconocimiento de comandos como la experiencia de usuario.

Las pruebas se realizaron en un entorno controlado dentro de un habitacion que estaba equipada con una mesa de trabajo despejada, una silla ergonómica y la infraestructura tecnológica necesaria para ejecutar la aplicación.

Se utilizaron dos versiones del sistema para comparar el rendimiento entre plataformas, pero hay 3 posibilidades de uso:

- Versión de escritorio con un monitor de 24 pulgadas
- Versión de móvil de gama media
- Versión en gafas Quest3

Participaron cinco individuos con perfiles técnicos diversos, que no habian probado la aplicacion con anterioridad y ademas usuarios como el 1 y el 5 no habian experimentado el uso de gafas tipo Quest3. Esta variedad de perfiles se buscó intencionadamente para obtener una retroalimentación rica y representativa de diferentes niveles de familiaridad con la tecnología.

Las pruebas se llevaron a cabo utilizando la versión final de la demostración de la aplicación que estan alojadas en los siguientes repositorios.

**Versión de escritorio:** [https://github.com/R4CC00N/SpeechRecognition\\_in\\_Browser/tree/main/DemoEscritorio](https://github.com/R4CC00N/SpeechRecognition_in_Browser/tree/main/DemoEscritorio)

**Versión móvil:** [https://github.com/R4CC00N/SpeechRecognition\\_in\\_Browser/tree/main/DemoEscritorio](https://github.com/R4CC00N/SpeechRecognition_in_Browser/tree/main/DemoEscritorio)

**Versión en gafas Quest 3:** [https://github.com/R4CC00N/SpeechRecognition\\_in\\_Browser/tree/main/DemoVR](https://github.com/R4CC00N/SpeechRecognition_in_Browser/tree/main/DemoVR)

Para cada una de las pruebas, un único experimentador guio a cada participante a través de la sesión, asegurándose de que las instrucciones se proporcionaran de manera estandarizada y consistente. Antes de comenzar cada tarea, el experimentador se dirigía al participante utilizando las siguientes palabras textuales:

“A continuación, te guiaré a través de una serie de tareas sencillas para evaluar cómo interactúas con el sistema. Por favor, sigue mis indicaciones y los mensajes informativos que aparecerán en la pantalla o dentro del entorno virtual.”

Una vez que el participante estaba preparado, el experimentador procedía a indicar cada tarea de forma secuencial, apoyándose también en los carteles de información integrados en la aplicación que proporcionaban recordatorios visuales de los comandos esperados. La secuencia de tareas que cada participante debía completar fue la siguiente:

**Creación de un cubo:** El experimentador indicaba verbalmente: “Por favor, intenta crear un cubo utilizando el comando de voz ‘**crear**’ y luego ‘**cubo**’.” El participante debía accionar el sistema de reconocimiento de voz (pulsando el botón) y pronunciar el comando. Después de la acción del usuario, se observaba si el cubo se creaba correctamente en la escena y avisara al experimentador.

**Cambio de color del objeto:** Una vez creado el cubo, el experimentador dirá: “Ahora, por favor, intenta cambiar el color del cubo a rojo utilizando el comando ‘**color**’ y acto seguido ‘**rojo**’.” El participante debía activar el reconocimiento de voz si se desactivo previamente y pronunciar el comando. En el momento en el que el color del cubo cambiaba a rojo según lo esperado se avisaba al experimentador.

**Posicionamiento del objeto:** Tras el cambio de color, el experimentador instruía: “Ahora, intenta mover el cubo a otra ubicación dentro de la escena. Puedes usar el comando ‘**posicion**’ y seguir las instrucciones en la información de la aplicación para especificar coordenadas.” Se observaba si el participante lograba desplazar el objeto utilizando los comandos de movimiento disponibles y una vez conseguido se avisaba al experimentador.

**Eliminación del objeto:** Finalmente, el experimentador pedía: “Por favor, elimina el cubo

de la escena utilizando el comando '**'eliminar'**'.“ Se verificaba si el objeto desaparecía de la escena tras la ejecución del comando por parte del usuario.

**Manejo libre de la aplicación:** Despues de completar las tareas guiadas, el experimentador indicaba: “Ahora tienes unos minutos para explorar libremente la aplicación. Intenta crear otros objetos, cambiar sus propiedades o realizar cualquier acción que te parezca interesante utilizando los comandos de voz que recuerdes o que veas en la información.“ Durante este tiempo, se observaba cómo el participante interactuaba con el sistema sin indicaciones directas, los comandos que intentaba utilizar y cualquier dificultad que pudiera encontrar.

## 5.1. Diseño Experimental

El experimento se enfocó en medir tres aspectos principales del sistema:

### Objetivos

- Precisión del reconocimiento de voz.
- Facilidad de uso e intuición de los comandos.
- Tiempo medio desde la grabación de voz hasta la creación de objetos.
- Conseguir realimentacion de los usuarios sobre la experiencia.

## 5.2. Resultados

De 20 comandos de reconocimiento de voz se han obtenido los siguientes resultados en cada versión, añadiendo las observaciones más destacadas de cada una en el Cuadro 5.2

### 5.2.1. Precisión del Reconocimiento de Voz

Usuario	Plataforma	Comandos exitosos	Observaciones
Usuario 1	Quest3	19	Reconocimiento lento, modalidad de grabación de audio molesta.
Usuario 2	Escritorio	18	Reconocimiento rápido, errores con las transcripciones en algunos casos.
Usuario 3	Móvil	17	Problemas con la interfaz para grabar audio, una vez obtenido el manejo, bastante fluido gracias a los mensajes de información
Usuario 4	Escritorio	17	Sin demoras en respuesta, no tiene comandos encadenados
Usuario 5	Quest3	18	Alta precisión en acento marcado, ligeras demoras en respuesta.

Cuadro 5.1: Porcentaje de reconocimiento de comandos por usuario y plataforma

### 5.2.2. Tiempo Promedio por Tarea

Usuario	Crear objeto	Cambiar color	Mover objeto
Usuario 1	Quest3	95 %	Reconocimiento lento, modalidad de grabación de audio molesta.
Usuario 2	Escritorio	90 %	Reconocimiento rápido, errores con las transcripciones en algunos casos.
Usuario 3	Móvil	85 %	Problemas con la interfaz para grabar audio, una vez obtenido el manejo, bastante fluido gracias a los mensajes de información
Usuario 4	Escritorio	85 %	Sin demoras en respuesta, no tiene comandos encadenados
Usuario 5	Quest3	90 %	Alta precisión en acento marcado, ligeras demoras en respuesta.

Cuadro 5.2: Porcentaje de reconocimiento de comandos por usuario y plataforma

- **Crear objeto:** 3–5 segundos.
- **Cambiar color:** 2–3 segundos.
- **Mover objeto:** 4–6 segundos.

- **Eliminar objeto:** 2 segundos promedio.

Se observó que los tiempos en la versión Quest3 fueron ligeramente más altos debido al retardo en la conexión al servidor externo.

### 5.2.3. Opiniones de los Usuarios

Las opiniones recogidas al final de las pruebas fueron mayoritariamente positivas. A continuación se resumen algunos comentarios:

- **Usuario 1:** "La experiencia en realidad virtual es muy inmersiva. Me gustaría poder usar frases más naturales y mejorar el tiempo de espera. Además, el manejo de grabar y parar de grabar por cada comando me pareció molesto."
- **Usuario 2:** "Sorprendido por la precisión del reconocimiento. Muy útil para crear escenas rápidamente y además muy vistosas."
- **Usuario 5:** "Solo bastan unos minutos para entender cómo manejar la aplicación, la parte de ejecución libre fue la mejor."

## 5.3. Conclusión de Validación

Los resultados obtenidos muestran que el proyecto cumple con los objetivos planteados: permite la creación y manipulación de objetos 3D mediante comandos de voz de forma eficiente y accesible, en entornos de escritorio y VR. La precisión en el reconocimiento fue alta en ambos entornos, y los usuarios consideraron la experiencia intuitiva y útil.

Es cierto que aún existen limitaciones como manejo de envío de audios para las transcripciones y latencia alta en una versión del proyecto.

# Capítulo 6

## Conclusiones

### 6.1. Consecución de objetivos

DEBEMOS REVISAR SI LOS OBJETIVOS PLANTEADOS ESTAN BIEN PARA SEGUIR POR AQUI.

### 6.2. Esfuerzo y recursos dedicados

Se ha dividido el proyecto por sprint y se ha hablado en cada sprint de su duracion en semanas, pero aun no destaca el tiempo que se le ha dedicado.

Durante el primer sprint **Investigación y Prototipado Inicial**, la parte mas larga a mi parecer fue leer documentacion y buscar informacion de diferentes tecnologias de reconocimiento de voz,

- La primera semana se busco informacion cada dia en torno a 2 horas excepto los fines de semana.
- La segunda semana se realizaron diferentes experimentos de cual seria la posible implementacion para la deteccion de voz y cual se elegiria para ello fue en torno a 3 horas durante viernes a domingo.
- La tercera semana se realizo una incrustacion del reconocimiento de voz y A-Frame en la que se trabajo durante 2 horas diarias de lunes a miercoles.

Dandonos un tiempo de trabajo en el primer sprint de 31 horas.

Para el segundo sprint **Desarrollo de A-FRAME Base**, la parte mas compleja fue el manejo y creación de componentes para crear objetos primitivos.

- La primera semana para la creación de objetos primitivos se trabajo durante 2 horas diarias durante 5 dias a la semana.
- La segunda semana se reestructuro el codigo y se creo manejadores para el componente de cracion de objetos este sprint se trabajo 4 horas solo sabado y domingo.

Dandonos un tiempo de trabajo en el segundo sprint de 18 horas.

Para el tercer sprint **Funcionalidades y Componentes**, la parte mas larga fue la creacion de componentes nuevos por el hecho de que se debian tener conexion entre componentes para que funcionasen.

- La primera semana para el aumento de comandos se trabajo por 3 horas miercoles, jueves y viernes.
- La segunda semana para la creacion de escena simple, solucion de problemas y funciones para componentes se trabajo 3 horas durante viernes a lunes.

Dandonos un tiempo de trabajo en el tercer sprint de 21 horas.

Para el cuarto sprint **Mejora de componentes y Mejora Visual**.

- La primera semana para creacion de componentes extra y modificacion de escena se trabajo 3 horas solo sabado y domingo.
- La segunda semana para crear comandos nuevos se trabajo 2 horas diarias durante la semana.

Dandonos un tiempo de trabajo en el cuarto sprint de 16 horas.

Para el quinto sprint **integración en VR/AR**.

- La semana para crear el server y hacer pruebas en las gafas topmo entorno a 6 horas de un solo dia.

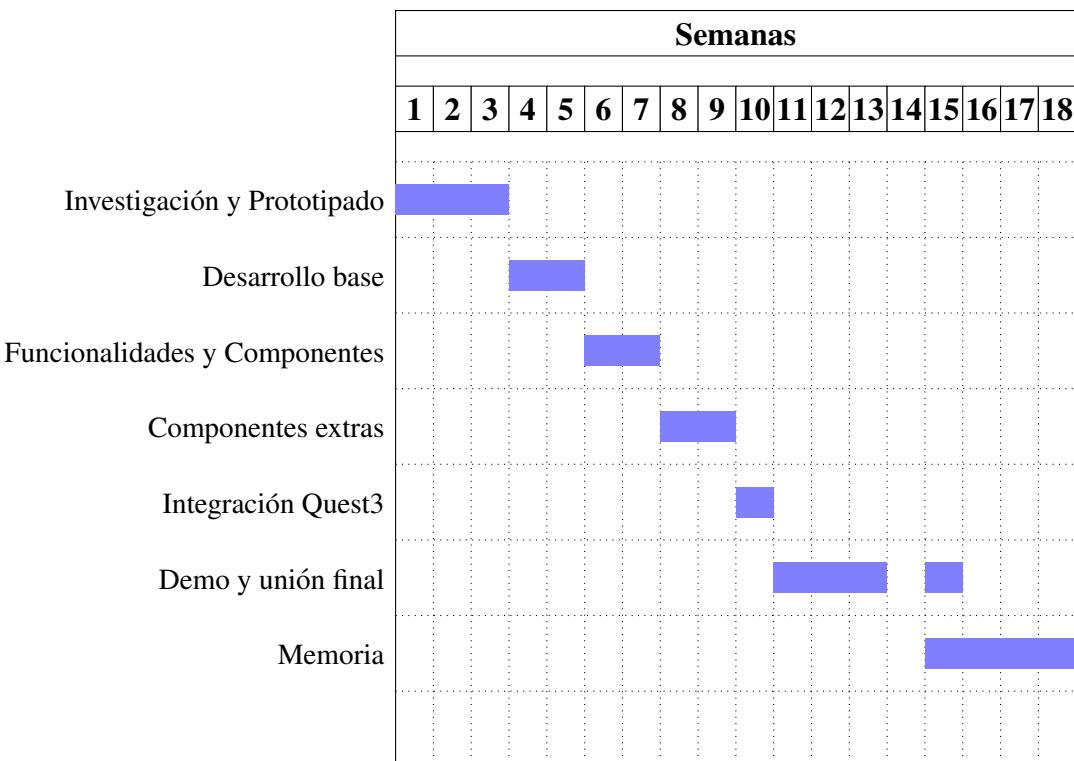
Para el sexto sprint **Pasos finales**.

- La primera semana para nuevos componentes se trabajo en ella durante 2 horas diarias toda la semana.
  - La segunda semana en la implementacion de plugins y pruebas se trabajo entorno a 4 horas diarias durante el finde semana
  - La tercera semana en la maquetacion y union de las demos se trabajo en torno a 3 horas diarias durante la semana.

Dandonos un tiempo de trabajo en el cuarto sprint de 36 horas.

Cabe recalcar que el proyecto se empezó a inicios de diciembre y se ha ido compaginando con el trabajo y las semanas de vacaciones, por ello es que se ha trabajado en la parte del código durante 4 meses aproximadamente y 1 mes más en la parte de la memoria

## Diagrama de Gantt - Sprints



### **6.2.1. Planificación temporal**

Con el diagrama de GANTT sirve?;

### 6.3. Aplicación de lo aprendido

...EN CONSTRUCCIÓN AÚN...

### 6.4. Lecciones aprendidas

A lo largo del desarrollo de este proyecto, se ha adquirido un conocimiento variado y profundo en tecnologías involucradas en la creación de la aplicación.

- Se ha profundizado en el uso de A-Frame principalmente, necesario para la creación de experiencias en realidad virtual.
- Se indaga profundamente en la construcción de componentes para A-Frame, con diferentes funcionalidades.
- Se profundizó en diferentes programas de detección de voz y sus aplicaciones.
- Creación de un servidor privado para conexiones mediante HTTPS.
- Trabajar con dispositivos de VR/AR como las Quest3.
- Se mejoró el manejo de JavaScript, aunque aplicado a el entorno de A-Frame.
- Se ha aprendido a utilizar LaTex para la realización de documentos de investigación y técnicos como es la memoria.

### 6.5. Trabajos futuros

En este apartado se van a mencionar posibles mejoras e ideas que se pueden implementar para mejorar esta aplicación:

- Se pueden mejorar la detección de comandos encadenados, de tal manera que si detectase en una frase todos los atributos necesarios no haría falta la respuesta del componente para pedir cada dato. Esto ayudaría al dinamismo para creación de escenas, además de evitar tiempos de espera entre comandos.

- Se pueden añadir comando para poder editar todos los parametros de cada objeto primitivo en la libreria de A-Frame. Por si es necesario la super edicion de objetos.
- Se podria integrar un modelo de lenguaje para que el programa entienda diminutivos, sinonimos, etc.
- En el tema visual, se podria añadir una interfaz que muestre al usuario que propiedades puede modificar dentro de una entidad, de esta manera ayudaria a saber como acceder y que valores se pueden dar.
- Tambien en el tema de retroalimentacion del programa al usuario, se podria añadir una funcion de voz (text-to-speech) para que devuelva los mensajes que ahora son de ayuda en texto, mediante la voz.
- Finalmente se podria crear un juego de plataformas simple en donde la dinamica principal para pasar el mapa sea la creacion de objetos con colisiones mediante comandos de voz e incluso crear objetos por voz para poder enfrentar enemigos, esta idea de juego aprovecharia al maximo la creacion de objetos y trancripcion de comandos de tal manera que la experiencia de juego se veria enriquecida.

# **Apéndice A**

## **Manual de usuario**

Esto es un apéndice. Si has creado una aplicación, siempre viene bien tener un manual de usuario. Pues ponlo aquí.

# Bibliografía

- [1] A-Frame. A web framework for building virtual reality experiences. <https://aframe.io/>, 2025. Consultado el 6 de abril de 2025.
- [2] A-Frame. A web framework for building virtual reality experiences. <https://aframe.io/docs/1.7.0/introduction/>, 2025. Consultado el 6 de abril de 2025.
- [3] AssemblyAI. Assemblyai - ai models to transcribe and understand speech. Official website of AssemblyAI.
- [4] W. W. W. Consortium. What is html? Introduction to HTML by the W3C.
- [5] J. Dirksen et al. *Learning Three.js: the JavaScript 3D library for WebGL*, volume 3. Packt Publishing Livery Place, UK, 2013.
- [6] K. Group. Webgl overview. Official website of the WebGL standard by Khronos Group.
- [7] W. I. W. C. Group. Webvr 1.1 specification, 2017. Superseded by WebXR Device API.
- [8] M. D. Network. Estructuras de datos y tipos en javascript. Documentación sobre tipos de datos primitivos, objetos y arrays en JavaScript por MDN en español.
- [9] M. D. Network. ¿qué es javascript? Introducción a JavaScript en MDN en español.
- [10] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever. Robust speech recognition via large-scale weak supervision. <https://github.com/openai/whisper>, 2022.
- [11] Supereggbert. aframe-htmlembbed-component, 2025. Accessed: 2025-05-03.

- [12] W3C. WebXR Device API. <https://www.w3.org/TR/webxr/>, 2024. [Accedido: 7 de abril de 2025].
- [13] Xenova. Whisper web - robust speech recognition in the browser. <https://huggingface.co/spaces/Xenova/whisper-web>, 2022. Hugging Face Space.