UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA INFORMÁTICA

Aplicación de realidad virtual para la visualización de escenarios tridimensionales

Virtual reality application to represent 3D environments

Realizado por
Pablo Díaz García

Tutorizado por
Rafael Marcos Luque Baena

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE
2021

Fecha defensa: septiembre 2021

Resumen

En este Trabajo de Fin de Grado se puede distinguir una parte de desarrollo y otra de investigación. Se desarrolla una aplicación web para visualizar entornos mediante el uso de gafas de realidad virtual. Estos entornos podrán ser interactivos y ofrecer una experiencia completa de inmersión.

Por otro lado, se investigan los diferentes marcos de desarrollo VR existentes y se selecciona el apropiado para nuestro desarrollo. Tras esta elección, se explicará paso a paso el método de aprendizaje usado para incorporar los conocimientos necesarios para este proyecto.

La aplicación web se realizará en el entorno de realidad virtual *A-Frame*, con base de datos *MongoDB* y el dispositivo que se usa para visualizar las escenas son las gafas *Oculus Go*. Además se crea una aplicación Android a partir de nuestra aplicación web mediante *AppsGeyser*.

Finalmente, se exponen las distintas funcionalidades que ofrece nuestro hardware en cuestión y se contrastan las capacidades de este frente a los dispositivos móviles, tanto a nivel funcional como a nivel de comodidad.

Palabras clave: Aplicación web, Realidad Virtual, *A-Frame*, *MongoDB*, *Oculus Go*.

Abstract

In this thesis we can distinguish between a development and a research part. It develops a web app in order to represent 3D environments which can be watched through specific hardware. Those environments will interact with the user to allow the participant to achieve complete inmersion.

On the other hand, it research among virtual reality frameworks for the purpose of choosing the appropriate for the development. Therefore, it will explain step by step the learning method.

The web app is developed in the virtual reality framework A-Frame. Its database belongs to MongoDB and the capability of watching those environments is granted by Oculus Go glasses. In addition, it creates an Android App through *AppsGeyser*.

At last, it shows the hardware functionality. It also contrasts the capabilities between virtual reality hardware and smartphones at different levels.

Keywords: Web Application, Virtual Reality, *A-Frame, MongoDB, Oculus Go.*

Índice

1 Introducción

1.1. Historia

Actualmente la computación gráfica es un elemento que aplicamos con frecuencia en nuestra vida cotidiana. Embarcados ya en el siglo XXI es inimaginable plantear la posibilidad de realizar ciertas labores profesionales sin la presencia de una computadora. Los avances técnicos han repercutido en una mayor accesibilidad de componentes, como son las gráficas y procesadores. Esto permite ofrecer el mundo de la realidad virtual al usuario de a pie, abriendo las puertas al interés de grandes compañías en el desarrollo de este sector apenas explorado. Un universo sin fronteras que nos permite experimentar escenarios de ensueño de forma cotidiana.

A partir de este crecimiento la industria de la realidad virtual ha trabajado en su popularización. La creación de varios *frameworks* de desarrollo de código libre ha fomentado enormemente su expansión. Podemos decir que la realidad virtual tiene presente, y su futuro se vislumbra esperanzador[1].

1.2. Motivación

Este proyecto nace a partir de la propuesta del tutor de indagar en las tecnologías de la realidad virtual. Principalmente planteamos la idea de orientar el trabajo a visualizar escenas relacionadas con habitaciones de viviendas, pero finalmente ampliamos nuestra visión a la posibilidad de observar cualquier escena existente. La principal motivación fue la posibilidad de indagar en las tecnologías de realidad virtual de forma amplia y profunda. Además de un interés personal sobre su funcionamiento.

La idea de crear una aplicación web que diera soporte y almacenamiento en una base de

datos a los modelos y la posibilidad de visualizarlos en ella entusiasmaba por su completitud. Además, el uso de herramientas y lenguajes de programación actuales daban a la labor de investigación aún más utilidad desde el punto de vista del empleo y la formación.

1.3. Objetivos

El principal objetivo de este proyecto es la creación de una aplicación web que de soporte a la visualización de entornos de realidad virtual mediante las gafas *Oculus Go*. También consta de objetivos secundarios que se centran tanto en el área de investigación como en el propio desarrollo de aplicaciones:

1. Investigación sobre los entornos web de desarrollo de realidad virtual y aprendizaje del elegido para realizar la implementación.

2. Investigar la utilidad y funcionalidades del entorno teniendo como objetivo la interacción con el mismo.

3. Entregar un producto funcional que incorpore los requisitos establecidos en un principio.

4. Creación de una aplicación móvil que use nuestra aplicación web.

5. Contrastar las posibilidades que se nos ofrecen mediante el uso de las gafas de realidad virtual *Oculus Go* frente al dispositivo móvil, a diferentes niveles de entendimiento.

1.4. Metodología

Se emplea una metodología iterativa. Para ello, se hace una distinción en bloques (iteraciones) de la construcción de la aplicación y se realizará un desarrollo por fases. Desarrollando en primera instancia las funcionalidades básicas, continuando por las funcionalidades menos necesarias para la aplicación, pero importantes a la hora de presentar un producto.

En todo este proceso tendremos en cuenta los requisitos iniciales del trabajo y, si es necesario, se añadirán a lo largo del transcurso del mismo.

Se dividirá el proyecto en iteraciones, comprendiendo cada una de ellas diferentes funciones, su sucesora se construirá con base en la anterior, de forma sucesiva. Aquí se muestra lo que abarcará cada una.

- **Iteración 0**: En esta fase se plantean los documentos necesarios para sentar las bases sobre las que se construirá el proyecto. Especificaremos los siguientes documentos:
  - **Requisitos**: Se especificará qué puede y qué no puede hacer nuestra aplicación a nivel funcional a nivel de usuario.
  - **Diagramas UML**: En estos documento se describe la interacción entre las pantallas y el usuario.
  - **Modelo de la base de datos**: Aunque la estructura utilizada es simple, es necesario especificar en forma de diagrama los objetos usados. En este caso debido a que se usa una base de datos *NoSQL* no habrá relaciones entre las entidades.
- **Iteración 1**: En este apartado se aborda la integración de las gafas de realidad virtual *Oculus Go* en la aplicación web. Además, se creará la base funcional de la aplicación, se centra sobretodo en el desarrollo de las principales vistas de la web, dejando a un lado ciertas funcionalidades con el objetivo de ser agregadas en posteriores iteraciones. Se gestionará la creación de la base de datos en la plataforma *MongoDB Atlas*, agregando los distintos tipos de esquemas necesarios e investigando la integración del mismo junto a la aplicación web en *Heroku*
- **Iteración 2**: En esta segunda fase se implementa la función de administrador, lo cual requiere de la inserción de nuevas características en nuestra base de datos y modificaciones en las vistas para la comprobación del rol de nuestro usuario actual. Además, se implementa la función exclusiva del administrador, registrar modelos.
- **Iteración 3**: Finalmente, agregaremos funciones de menor importancia de cara al funcionamiento clave, pero esenciales desde un punto de vista realista de una aplicación en producción. Estas son principalmente:
  - **Mensajes de información**: se implementan mensajes que informan al usuario del estado de los procesos solicitados.

- **Seguridad**: se añade seguridad entre vistas, con el objetivo de incapacitar al usuario a moverse entre vistas mediante la modificación de la dirección web y garantizando el acceso a nuestra aplicación exclusivamente a usuarios registrados de forma satisfactoria.
- **Aplicación móvil**: se crea una aplicación *Android* simple, mediante *AppsGeyser*, una aplicación asistente para trasladar una web existente a una aplicación móvil.

1.5. Tecnologías usadas

1.5.1. Base de datos

La aplicación web necesita capacidad para poder almacenar datos y extraerlos, por lo tanto es esencial utilizar una base de datos. Se selecciona *MongoDB*, que es una base de datos *NoSQL* muy popular.

Debido a que la complejidad de datos era baja y la relación entre ellas no tomaba un rol a tener en cuenta, sólo se necesita una base de datos con una estructura sencilla. Para esto las bases de datos de tipo *NoSQL* son apropiadas ya que no hay relaciones entre las entidades y es una especie de sistema de almacenamiento de ficheros, en el que cada fichero es una instancia de un esquema de la aplicación.

Por otro lado, es necesario para la aplicación que la base de datos funcione en la nube, y *MongoDB* cuenta con una plataforma propia llamada *MongoDB Atlas* que cumple esta función.

Figura 1: *MongoDB.*

1.5.2. Servidor

En una primera instancia se plantea la gestión de la aplicación web mediante un servidor local. Para ello, se comienza usando la aplicación *XAMPP* que permite integrar la base de datos y la aplicación web en el mismo.

Figura 2: *XAMPP.*

Sin embargo, tras indagar, se encuentran problemas de compatibilidad entre las gafas y la posibilidad de acceder al servidor local del computador. Esta funcionalidad se permite para otras gafas de realidad virtual del mercado como *Oculus Quest* mediante aplicaciones propias, sin embargo, en este caso se insta a modificar la decisión inicial y comenzar la búsqueda de un sistema de almacenamiento de la aplicación web en la nube.

Tras investigar sobre diversas herramientas de almacenamiento de aplicaciones web como servidor, se seleccionó *Heroku.* Esta es una plataforma que ejecuta aplicaciones del cliente en la nube. Esto implica que los desarrolladores no tienen que preocuparse por la infraestructura del servidor, simplemente deben centrarse en la funcionalidad de la aplicación. Se eligió esta herramienta debido a que es simple de utilizar, gratuita y además integra un sistema de gestión de versiones parecido a *GitHub.* Esto implica que se puede llevar en una misma aplicación la integración de ambas y facilita el desarrollo.

Figura 3: *Heroku.*

1.5.3. Gestión de versiones

Para el mantenimiento y control de versiones se hace uso de una herramienta muy popularizada actualmente como *Git*. Este programa es un software cuya utilidad se basa en el almacenamiento de código en la nube, y además, la posibilidad de gestionar diferentes ramas de un mismo código o en este caso de la aplicación web.

Se decide hacer uso de ella debido a su capacidad de guardar versionado, esta facilitaba cualquier labor a la hora de introducir nuevas funcionalidades y proporcionaba la posibilidad de volver a la versión anterior en cualquier momento.

Además su integración con *Heroku* era la combinación perfecta para poder llevar el versionado y el servidor en una misma aplicación y así no tener las funcionalidades tan descentralizadas.

Figura 4: *Git*.

1.5.4. Entorno de Desarrollo Web

Una parte de este trabajo comprende el estudio y la búsqueda de un *framework* de desarrollo web adecuado para las características requeridas. A este estudio se le dedica una sección, por lo que aquí simplemente se introduce de forma ligera el entorno elegido, *A-Frame*. *A-Frame* es un *framework* de desarrollo web para construir experiencias en realidad virtual. Se basa en un modelo entidad-componente que se estructura sobre la librería *three.js* del cual hablaremos en posteriores apartados 4.5. Este entorno es de los más extendidos y aunque la realidad virtual no tiene todavía tantos devotos como otras áreas, este es uno de los que más información encontraremos en Internet.

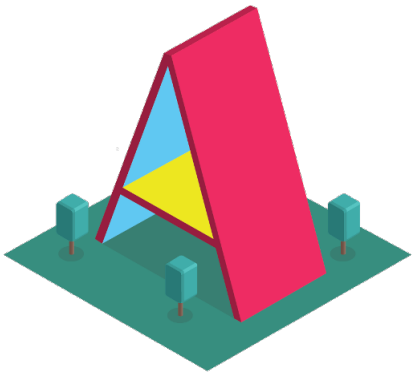


Figura 5: *A-Frame.*

1.5.5. Herramientas Adicionales

- **Aplicaciones *Oculus***

  Ha sido necesario el aprendizaje e investigación de herramientas y aplicaciones proporcionadas por *Oculus* con objetivo de realizar la integración y compatibilidad con la aplicación web. La posibilidad de visualizar el desarrollo de la realidad virtual fue frustrante, pues tanto la misma información de *Oculus* como la encontrada en Internet era mínima en cuanto al desarrollo. Se intentó mediante *Oculus Desktop*, *MozillaWebVR* o incluso el mismo almacenamiento de ficheros de las gafas. Finalmente el servidor en la nube fue la solución a este problema.

Figura 6: Tecnologías *Oculus.*

- **Entorno de desarrollo integrado**

  Inicialmente se comienza usando *Brackets*, principalmente debido a los beneficios que proporciona para la programación en *frontend*. Ofrece la capacidad de ver en tiempo

real como afectan tus modificaciones de código sobre la página web, facilitando así la compresión e identificación sobre la misma.

Figura 7: *Brackets.*

Debido a problemas de acceso desde las gafas *Oculus Go*(Apartado desarrollo) hacia el servidor local de mi máquina, nos vemos obligados a cambiar de entorno a *Glitch* provisionalmente, pues este es un entorno de desarrollo integrado *online* que permite acceder al código desde la navegación en las gafas. Esto permite probar cómo afectan modificaciones a los controles, movimiento y cámara.

Finalmente, el servidor de Heroku permite no depender la necesidad de mantener código *online*, y además surge la necesidad de crear un *back-end* para la aplicación. En este punto se decide usar finalmente *Visual Studio Code* debido a la gran ayuda que ofrece sus *plugins* y su sistema de control de ficheros dentro de la misma aplicación.

Figura 8: *Visual Studio Code.*

- **Front-end** Para el desarrollo del *front-end* se hace uso de *HTML* y *CSS*. Partiendo de entornos ayuda como son *Bootsrap* y librerías que ayudan al desarrollo de iconos y fuentes como *FontAwesome* y *GoogleFonts*.

Figura 9: *Bootsrap.*

- **Back-end** Esta parte de la aplicación se basa en *Node.js*, un entorno de *JavaScript*. Sobre este, se hace uso de diversas librerías facilitando así la implementación de diversas funcionalidades para la aplicación. Merece la pena destacar las siguientes:

  - *Three.js*: Permite crear y mostrar gráficos animados por computadora en 3D en un navegador Web. *A-Frame* está construido sobre él, por lo tanto será esencial su entendimiento[2].
  - *Mongoose*: Permite escribir consultas para una base de datos de *MongoDB*, con características como validaciones, construcción de *queries*, *middlewares*, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos [3].
  - *Express*: Es un marco de desarrollo minimalista para *Node.js* que permite estructurar una aplicación de una manera ágil, nos proporciona funcionalidades como el enrutamiento, opciones para gestionar sesiones y *cookies*, etc[4].
  - *Passport*: Es un *middleware* para *Express* que nos permite implementar estrategias de autenticación de una manera rápida y simple[5].
  - *EJS*: Se encuentra entre los motores de visualización de plantillas más populares para *Node.js* y *Express*. Su cualidad principal es la posibilidad de crear código *HTML* mediante *JavaScript*. Facilita la generación de contenido dinámico[6].

- **Appsgeyser** Esta aplicación web es la herramienta utilizada para crear la aplicación móvil en *Android*. Realmente esta crea una aplicación web cubierta por una capa que es el icono de aplicación. Simplemente al pinchar en él, redirige a la aplicación web que ya se tiene alojada en la nube.

Figura 10: *Appsgeyser.*

1.5.6. Estructura de la memoria

La memoria se divide en tres partes principales. En cada uno de estos capítulos se abordan diferentes ramas de este proyecto. Son de diversa índole pues unos formarán una parte más teórica y de investigación, y otros abordan la parte práctica y experimental del proyecto. Todos los capítulos se dividiren a su vez en diferentes apartados para darle claridad a las etapas en el proyecto y ofrecer una mejor perspectiva.

En una primera parte se aborda la investigación sobre los *frameworks* de desarrollo web de realidad virtual existentes, el análisis y la justificación la elección final. Además se da la perspectiva paso a paso del aprendizaje seguido del mismo.

Seguidamente se detalla un capítulo de investigación en el cual se ponen en contraposición las características y funcionalidades de las gafas de realidad virtual (*Oculus Go*) y los dispositivos móviles, a la hora de visualizar entornos 3D.

En tercer lugar, se muestra una descripción del proceso de desarrollo, desde la creación de los requisitos y diagramas en los que se basará la aplicación, hasta las especificaciones finales del producto.

Finalmente se agregan anexos como el manual de instalación, donde se detalla cómo hacer funcionar paso a paso y qué es necesario para hacer funcionar la aplicación, el manual de usuario y la bibliografía dónde se refieren todas las fuentes consultadas.

2

Entornos de desarrollo de realidad virtual

En este apartado se realiza una labor de investigación para indagar sobre los *frameworks* de desarrollo web realidad virtual más usados actualmente. Se pone el punto de mira en el análisis sobre qué ofrece cada *framework* en contraste al resto, y finalmente se expondrá un veredicto donde seleccionaremos uno de forma justificada.

2.1. ¿Qué es *WebVR*?

Es fundamental conocer qué hace que nuestros marcos de desarrollo sean posibles antes de investigarlos, y se lo debemos todo a *WebVR*. Hasta hace poco, las experiencias *VR* en web eran pobres debido a la potencia de banda ancha para poder transmitir los datos, sin embargo actualmente ha cambiado. Proyectos como *Google Stadia* o *Steam VR* simulan correr en tu dispositivo cuando se ejecutan íntegramente desde un servidor en un computador remoto.

WebVR está compuesta de librerías de *JavaScript* que realizan la compatibilidad con los navegadores. Además cada navegador puede implementar modificaciones para dar soporte a sus particularidades[7].

- **Código abierto**: Todo el código de sus librerías es de libre uso, lo cual hace que desarrollar en el sea accesible.
- **Rendimiento**: Lleva registro de tu rendimiento pues la realidad virtual puede ocasionar malestar si el índice de fotogramas es bajo. En dispositivos móviles su frecuencia de refresco es 60Hz y 60fps, mientras que en computador es 90Hz.

- **Accesibilidad**: Normalmente este tipo de herramientas requieren de habilidades técnicas para su manejo, no en este caso.No son necesarios conocimientos previos, teniendo un ordenador, navegador compatible y soporte *hardware* (gafas *VR*), un simple *click* te ofrece una experiencia *VR* auténtica.

- **Compatibilidad**: Actualmente es compatible con todos los navegadores más actuales (*Chrome*, *Firefox*, *Safari*,*Edge*) y, además, también con realidad aumentada y mixta.

- ***Cloud***: Se almacena en la nube, una ventaja que actualmente se está pidiendo cada vez más.

2.2. Frameworks a investigar

Todos los entornos de desarrollo sobre los que se profundizará trabajan sobre la mencionada *WebVR*. Entornos existentes como *Primrose* o *Argon.js* han sido pre-candidatos iniciales que no han pasado a la etapa final debido a su escasa documentación.

2.2.1. *React360*

Este *framework* es propiedad de *Oculus* y está basado en *React* y *React Native*, por lo que si dominas estos debería ser tu *framework* ideal, aunque su comunidad actualmente es pobre.

El principal objetivo de *React360* es permitir a los desarrolladores un punto de vista declarativo sobre *React*. *React360* se apoya en *three.js* para poder trabajar con las *APIs WebVR* y *WebGL(Web Graphics Library)* que le permite renderizar entornos 3D sin necesidad de usar *plug-ins* en el navegador.

Se caracteriza por ser un *framework* accesible y por su aceptable curva de aprendizaje. Es importante conocer que este entorno es únicamente compatible con modelos *.obj* y *.gltf2*.

React360 divide el programa en dos hilos de ejecución independientes. Uno de ellos se encarga de soportar la aplicación, y el otro procesa el modelo virtual. El objetivo de este mecanismo es ser más eficiente en el procesamiento de datos en tiempo real y, por lo tanto, requerir menos tiempo de computación.

```javascript
import React from "react";
import { AppRegistry, asset, StyleSheet, Pano, Text, View } from "react-

class EarthMoonVR extends React.Component {
  render() {
    return (
      <View>
        <Pano source={asset("chess-world.jpg")} />
        <Text
          style={{
            backgroundColor: "blue",
            padding: 0.02,
            textAlign: "center",
            textAlignVertical: "center",
            fontSize: 0.8,
            layoutOrigin: [0.5, 0.5],
            transform: [{ translate: [0, 0, -3] }]
          }}
        >
          hello
        </Text>
      </View>
    );
  }
}

AppRegistry.registerComponent("EarthMoonVR", () => EarthMoonVR);
```

Figura 11: Código ejemplo de *React360*.

En este ejemplo podemos ver como su sintaxis es similar a la de *JavaScript*. Usa componentes como *View*, *Pano* y *Text* a los cuales se les puede adherir modificaciones como podemos ver en el tipo *Text*. Se crea una clase de tipo componente que será nuestra experiencia 3D. Todos los elementos deben ir dentro de la clase *View*, que es la clase principal. En este caso se usa un modelo importado en el componente *<Pano>* y se le agrega una etiqueta *<Text>* con el valor *hello*. A este texto se le adhieren ciertas modificaciones como el color de fondo, el tipo de texto, la posición.

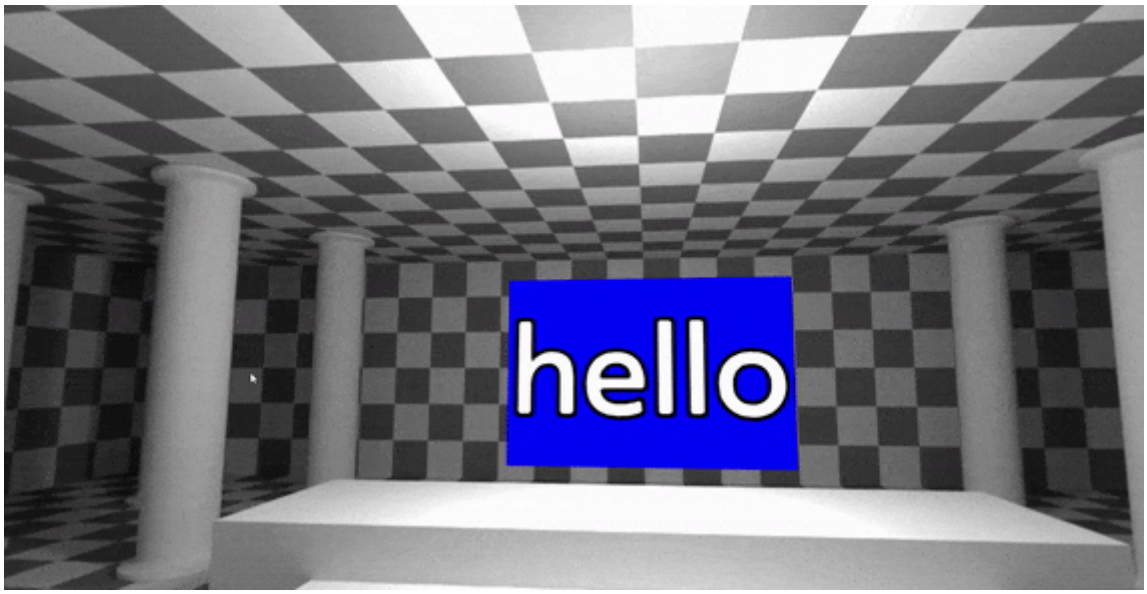


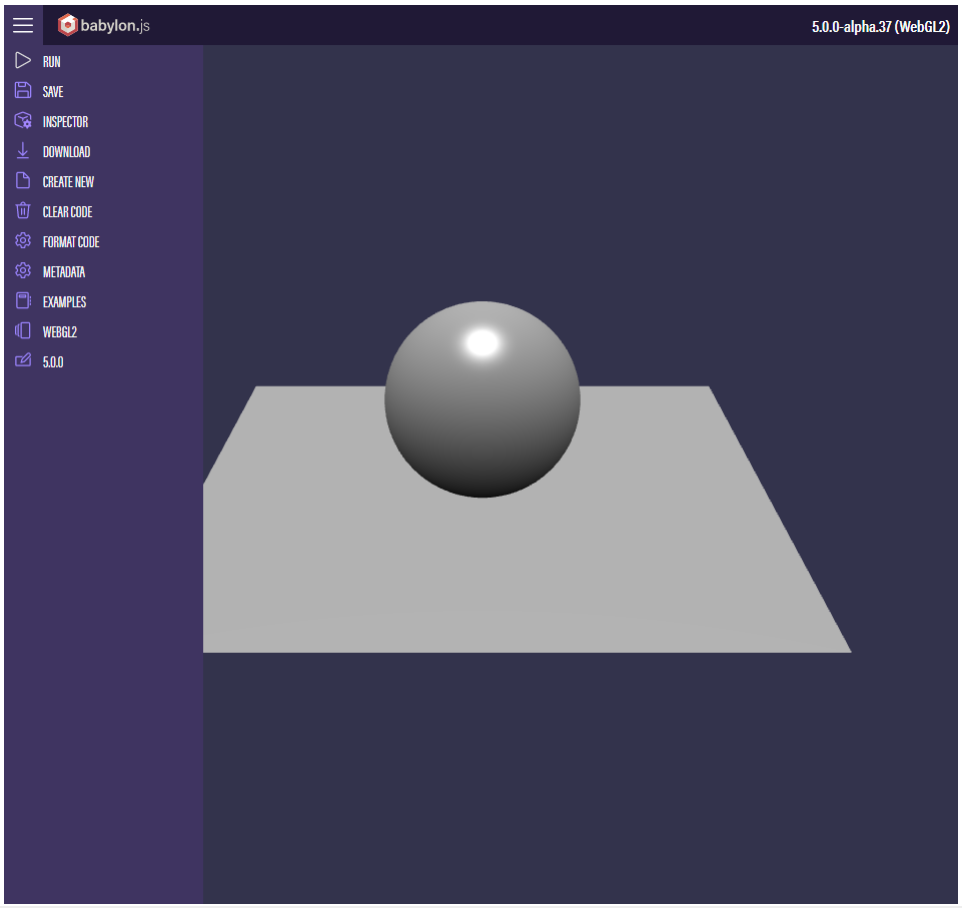
Figura 12: Representación del código *React360*.

Vemos como se muestra el componente texto dentro del modelo cargado (entorno con columnas) con las modificaciones realizadas.

2.2.2. *Babylon.js*

Este entorno, cuyo código fuente está escrito en *TypeScript* y compilado en *JavaScript*, se usa para mostrar entornos 3D en el navegador mediante *HTML5*. Aunque el entorno está evolucionando, su comunidad aún escasa y es complicado encontrar código en la red más allá de su propia documentación.

Una característica distintiva respecto a los demás entornos es la existencia del *PlayGround*. Una aplicación web del propio *Babylon.js* que nos permite crear código *online* sin necesidad de instalar ninguna dependencia. Además también nos permite visualizarlo y diferentes opciones de diseño.




```
var createScene = function () {
    // This creates a basic Babylon Scene object (non-mesh)
    var scene = new BABYLON.Scene(engine);

    // This creates and positions a free camera (non-mesh)
    var camera = new BABYLON.FreeCamera("camera1", new BABYLON.Vector3(0, 5, -10), scene);

    // This targets the camera to scene origin
    camera.setTarget(BABYLON.Vector3.Zero());

    // This attaches the camera to the canvas
    camera.attachControl(canvas, true);

    // This creates a light, aiming 0,1,0 - to the sky (non-mesh)
    var light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0, 1, 0), scene);

    // Default intensity is 1. Let's dim the light a small amount
    light.intensity = 0.7;

    // Our built-in 'sphere' shape.
    var sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 2, segments: 32}, scene);

    // Move the sphere upward 1/2 its height
    sphere.position.y = 1;

    // Our built-in 'ground' shape.
    var ground = BABYLON.MeshBuilder.CreateGround("ground", {width: 6, height: 6}, scene);

    return scene;
};
```

Figura 13: *Babylon.js PlayGround.*

Vemos que su sintaxis es similar a la de *three.js*. El objeto principal de *Babylon.js* donde irán incluidos todos los elementos de nuestra escena se crea con el método *new BABYLON.Scene()*. Podemos dividir nuestra escena en tres partes:

1. Creación de una cámara que represente el punto de vista actual sobre la escena y la agregación de controles a esta cámara para posibilitar el movimiento

2. Un tipo *CreateSphere* el cual se llama mediante el método *MeshBuilder*. Este *mesh* nos vuelve a recordar a similitudes con *three.js*. A esta esfera se le agrega una iluminación mediante el tipo *HemisphericLight*.

3. Un tipo *CreateGround* que crea el suelo sobre el que reposa la esfera.

2.2.3. *A-frame*

Es un proyecto de código abierto desarrollado por *Mozilla* que ha conseguido crear una de las mayores comunidades de realidad virtual. La principal característica de *A-Frame* es ser un lenguaje declarativo y del tipo entidad-componente con el objetivo de adherir cualidades a *three.js*.

A-Frame tiene cualidades que lo hace ser un gran candidato a la hora de elegir un *framework* para desarrollar. Su sintaxis es de alto nivel por lo tanto incluso una persona de fuera del mundo de la programación podría comprender el código. Tiene compatibilidad plena y específica para cada uno de los controladores hardware 3D actuales (entre ellas *Oculus Go*).

Sus principales características son:

- **Simplicidad**: Su complejidad sintáctica es muy baja, una simple etiqueta *<a-scene>* en tu código *HTML* será suficiente para hacer aparecer en pantalla el botón VR que te permite adentrarte en la experiencia. *A-Frame* maneja la plantilla 3D, la configuración VR y los controles predeterminados, sin necesidad de instalar dependencias.

- **Es declarativo**: su código es fácil de leer y entender, accesible para desarrolladores, profesionales, o gente inexperta.

- **Comunidad**: es el *framework* con mayor cantidad de información en la web, tanto tutoriales como código. Se siente una tecnología con una comunidad viva y con futuro.
- **Es multiplataforma**: permite crear aplicaciones funcionales para todos los controladores del mercado, Vive, *Rift*, *Daydream*, *GearVR* e incluso si careces de un hardware de este tipo para su visualización, *A-Frame* también es compatible con dispositivos móviles.
- **Arquitectura Entidad-Componente**: *A-Frame* es un potente *framework* de *three.js*, que proporciona una estructura declarativa, modular y reutilizable de componentes de entidad. Los desarrolladores tienen acceso ilimitado a *JavaScript*, *APIs DOM*, *three.js*, *WebVR* y *WebGL*.
- **Rendimiento**: está optimizado para funcionar en *WebVR*. Una de las cualidades que le distingue del resto es que *A-Frame* hace uso del *DOM*, sus elementos no tocan el motor del diseño del navegador. Cuando se actualizan los objetos se hace en memoria con baja sobrecarga.
- **Inspector Visual**: esta herramienta práctica integrada le aporta más accesibilidad aún a *A-Frame*. Se trata de inspector visual que permite agarrar, rotar o escalar entidades de nuestra escena en tiempo real. Se puede acceder a este inspector desde cualquier escena presionando *<ctrl>+ <alt>+ i*[8].

Figura 14: Ejemplo de *A-Frame*.

```
<!doctype html>
<html>
    <head>
        <title>Hola mundo</title>
        <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
    </head>

    <body>
        <a-scene>
            <a-box color="#689F38"
                   position="-2 1 -7"
                   scale="2 2 1">
            </a-box>

            <a-box color="red"
                   position="4 1 -7"
                   scale="2 2 1">
            </a-box>
        </a-scene>
    </body>
</html>

```

Figura 15: Código de la representación de *A-Frame*.

En este ejemplo podemos distinguir fácilmente los elementos representados. El objeto principal contiene la escena es *<a-scene>* y en él vemos dentro dos objetos del tipo *<a-box>* cada uno de ellos relacionado con una caja. Vemos que su sintaxis es parecida a *HTML*.

2.3. Veredicto

Es necesario destacar que la labor de investigación ha estado trabada por la escasa información existente en la web sobre realidad virtual. Finalmente el *framework* elegido para la realización del trabajo será *A-Frame*.

Ciertas cualidades en concreto son las que han hecho decantarse la balanza. Su simplicidad hacía que el acercamiento a esta nueva tecnología fuera visual y relativamente liviano. La integración con *JavaScript* permitía no partir desde cero pues se tenía cierto conocimiento previo sobre él, además de su compatibilidad con los modelos *glb* y *gltf* relacionados con los modelos. Pero la cualidad que realmente ha sido decisiva es la comunidad. La diferencia de información que podemos encontrar de *A-Frame* con respecto a los demás es abrumadora,

no sólo en la documentación donde esta se explaya de manera amplia y cercana, sino en el código existente en la red. Era fundamental la existencia de material donde poder aprender este lenguaje y esto era algo prácticamente inexistente en el resto de entornos.

3

Actualidad de la realidad virtual

3.1. Actualidad del mercado de la realidad virtual

En esta sección vamos a introducir la actualidad del mercado en cuanto a *hardware* de realidad virtual. Se analizan los diferentes visores a destacar del mercado y mostraremos sus ventajas y desventajas respecto a otros pudiendo así plantear un mapa conceptual orientativo[9].

- *Google CardBoard* (10€): Es el más económico, y totalmente dependiente de *hardware* externo para su funcionamiento. Esencialmente es un visor cuyo material base es el cartón, y su objetivo es sostener un dispositivo móvil que hace la función de pantalla del visor. Sus agarres no son cómodos y es bastante frágil. Ampliaremos su información en la siguiente sección.

Figura 16: *Google CardBoard.*

- *Oculus Go* (220€): Cuenta paneles *LCD* con una resolución conjunta de 2.560 x 1.440 puntos y una frecuencia de actualización de 72 Hz. Tiene procesador propio, por lo tanto,

puede funcionar sin necesidad de conectarse a un computador. Su potencia gráfica no es elevada, se puede catalogar entre la *PlayStation 2* y la 3. Es un visor con bajo punto dulce, es probable sufrir mareos si se han sufrido en otras expreciencias *VR*[10].

Figura 17: *Oculus Go*.

- *Oculus Quest 2* (350€): Este visor se considera uno de los mejores, si no el mejor, calidad-precio del mercado actualmente. No necesita de ningún *hardware* adicional para funcionar aunque también ofrece la posibilidad de conectarse a un computador y funcionar con sus requisitos técnicos. Cuenta con su propio procesador, *RAM* y *GPU*. Un panel *OLED* 1832x1920 por ojo y una tasa de refresco de 90*Hz*. En general, podemos catalogarlo como uno de los mejores visores para entrar en el mundo de la realidad virtual. Importante a destacar un par de ámbitos de mejora, la batería oscila las dos horas de duración, todavía es pobre, además el sistema de correas y sujeciones resulta ligeramente incómodo debido al peso del visor, 500 gramos.

Figura 18: *Oculus Quest 2*.

- *Oculus Rift S* (450€): Uno de sus mayores defectos es no ser inalámbrico. Su pantalla cuenta con una resolución 1280 x 1440 píxeles por ojo, con 80 Hz de tasa de refresco, las inferiores que se muestran en este análisis. Sin embargo, esta bajada no es apreciable debido a una reducción en la persistencia, cuyo objetivo de forma resumida es la disminución de tiempo al imprimir un *frame*. Destacar la mejora de un nuevo concepto, el cual es significante para este producto, el punto dulce. Este se define como el punto de colocación del visor donde la integración de las gafas en tu visión es completa, sin efectos borrosos o distorsiones al desplazarla. Son una opción completamente válida que nos proporciona una experiencia auténtica sin llegar a ser unos visores *premium*.

Figura 19: Visor *Oculus Rift S*.

- *HTC Vive Pro* (800€): Estas gafas permiten experimentar la mejor experiencia inmersiva actualmente debido a su tecnología de doble rastreo y mapeo. Establecen con máxima precisión la posición del usuario lo cual se asemeja más a una experiencia real. Sus características son resolución total de 2880 x 1600, 615 PPI y 90 Hz de tasa de refresco, con 110 grados de ángulo de visión. Además de esto cuentan con una tecnología de ergonomía cuyo objetivo es compensar el peso de la cabeza del usuario, lo cual es destacable ya que un uso de tiempo prolongado podría causar daños en nuestro cuello. Es interesante contrastar con el anterior visor, en este caso el punto dulce se ha reducido al aumentar las resoluciones.

Figura 20: Visor *HTC Vive Pro*.

- *Valve Index* (1000€): Este visor se encuentra en la cumbre del panorama de realidad virtual, su resolución es de 1600x1440 y su tasa de refresco 120Hz. Es un *hardware* donde confluyen las mejores características expuestas anteriormente. Gran tasa de refresco con baja persistencia. Mejora del punto dulce, comodidad y potencia superior. Recomendado para profesionales del sector.

Figura 21: Visor *Valve Index*.

En este proyecto nos centramos en la comparación entre las *Oculus Go* y las *Google Cardboard*. Estas últimas seleccionadas debido al aumento de ventas hace relativamente poco, en contraposición a *Oculus Go*, visor entregado por el tutor para la realización de este proyecto.

3.2. Comparativa

Se ubica en 2010 el nacimiento de las primeras gafas de realidad virtual, obra de *Palmer Luckey*. La escasa accesibilidad y el elevado coste de estas, hicieron a *Google* lanzar *Google CardBoard* en 2014, con el objetivo de acercar la realidad virtual a los usuarios habituales. En este apartado vamos a comparar, desde el punto de vista técnico y desde la experiencia de uso, qué ofrece *Google CardBoard*, frente al hardware específico de realidad virtual.

No sería justo contrastar las cualidades técnicas de cada herramienta sin poner en contexto lo que rodea a cada tecnología. Es importante conocer la gran diferencia de precio que existe entre estas dos opciones. El coste *Google CardBoard* ronda los 10€, mientras que un hardware de realidad virtual actualmente no baja de los 200€. Obviamente esta diferencia de precio se ve reflejada en el producto tanto en su comodidad como en sus capacidades técnicas[10]. En mi experiencia a nivel de usuario tengo que destacar diferentes puntos:

- Las gafas de cartón no se adecúan ni se adaptan correctamente a tu cara. Constan de una banda que pretende ajustar el visor a tu cabeza que cuando lo comparas con las *Oculus Go* se queda bastante atrás. Este desajuste origina que se filtre luz dentro de las gafas lo cual puede derivar en malestar visual y mareos.

- Para los usuarios con gafas graduadas es incómodo compatibilizar estas con las *Google CardBoard*, esto básicamente impide el uso de estas a usuarios con problemas de visión. Por otro lado, *Oculus Go* viene con un adaptador para utilizar las gafas graduadas a la vez que el visor.

- La sensación al llevar puesto *Oculus Go* es firme y sólida, sin embargo las gafas de cartón son endebles debido a su material y se deben cuidar encarecidamente.

Dejando a un lado la experiencia de usuario, centrémonos en las diferencias técnicas y qué pueden suponer estas para nuestro disfrute de una experiencia de realidad virtual auténtica. En este caso se comparará con un dispositivo móvil gama media actual, (*Realme 7 pro*).

	Realme 7 Pro	***Oculus Go***
Resolución	2400x1080	2560x1440
Pantalla	*AMOLED*	*LCD*
Tasa de refresco	60Hz	72Hz

Podemos ver en esta tabla que las*Oculus Go* aportan mayor cantidad de píxeles por pantalla (resolución) y mayor tasa de refresco, esta última influye en la fluidez con la que vemos las transiciones de *frames*.

Se podría pensar que en el tipo de pantalla ganan los dispositivos móviles ya que *AMOLED* está mejor valorada que *LCD*, pero la realidad es que las pantallas de tipo *LCD* muestran colores más realistas mientras que las tipo *AMOLED* muestran colores menos apagados. Por lo tanto, a la hora de realizar una inmersión a una experiencia virtual nos interesa que nuestra pantalla refleje colores cercanos a la realidad.

Además de la calidad de imagen, también encontramos diferencias notorias en las posibilidades que nos ofrecen una y otra en la experiencia virtual. El dispositivo móvil solo reconoce 3 de los 6 grados de libertad existentes, esto implica que no te permitirá desplazarte por la escena. Además tampoco se integran las funcionalidades que tienen los mandos por lo tanto mediante móvil, no podrás interactuar con el entorno.

Estas limitaciones son notorias a la hora de adentrarse en una experiencia de realidad virtual. No podrás visualizar entornos de cierta complejidad que requiera de tu movimiento, y, al no poder interactuar con mandos, no podrás navegar por los entornos que ofrece nuestra aplicación ni interactuar con ellos. Como se explica posteriormente en el proyecto, estas limitaciones son realmente visibles a nivel de desarrollador.

Entramos finalmente en el ámbito de la configuración del dispositivo. En el caso del *smartphone*, simplemente debemos introducir la dirección de nuestra página web deseada e insertar el mismo en *Google CardBoard*. Por otro, lado bajo mi punto de vista, la configuración de *Oculus Go* es ligeramente más laboriosa simplemente debido a la dificultad de escribir direcciones de web mediante el mando controlador.

4

Aprendizaje de A-Frame

En este apartado se expone paso a paso el método que se ha utilizado para el aprendizaje de A-Frame. Desgranaremos este proceso en diferentes partes explicando desde los elementos más básicos, hasta las interacciones que podemos obtener de sus funcionalidades.

4.1. Primitivos

Estos elementos son la unidad básica en *A-Frame*, su principal objetivo es ser la envoltura de otras entidades y así disminuir la dificultad de su compresión, básicamente, azúcar sintáctico.

Es importante conocer que todos los primitivos tienen como base el objeto *<a-entity>*, al cual le adhieren de forma predeterminada ciertos componentes. Expongamos un ejemplo para visualizar esto.

```
<a-box color="red" width="2"></a-box>

<a-entity geometry="primitive:box; width:2" material="color:red"></a-entity>
```

Figura 22: Comparación de entidades.

Estos dos objetos representan al mismo elemento visualmente. Podemos ver como el elemento *<a-box>* establece como predeterminado que su atributo *geometry* será del tipo *box*. Además cumple una función de enlazar atributos con la entidad reduciendo así la cantidad de código que se escribe, mientras que en la etiqueta *<a-box>* para establecer su color a rojo

basta con escribir *color="red"*, en *<a-entity>* es necesario acceder primero a la propiedad de esta manera, material="color: red". Esta función de emparejamiento de atributos la realizan los elementos primitivos[11].

Esta relación entre primitivos y *<a-entity>* también nos ofrece la capacidad de adherirle componentes. Estos componentes pueden modificar valores visuales del objeto como el color o la forma, pero también modifican su posición, rotación o escala.

A-Frame nos permite registrar nuestros propios elementos primitivos con el objetivo de agilizar la codificación y eliminar la duplicidad de código. Veamos cómo se registraría un primitivo *<a-box>*.

```
51    <script>
52        AFRAME.registerPrimitive('a-box', extendDeep({}, meshMixin, {
53
54            defaultComponents: {
55                geometry: {primitive: 'box'}
56            },
57
58            mappings: {
59                depth: 'geometry.depth',
60                height: 'geometry.height',
61                width: 'geometry.width'
62            }
63            }));
64    </script>
```

Figura 23: Cómo registrar un primitivo.

Mediante el método *AFRAME.registerPrimitive*(nombrePrimitivo,objetoJavaScript) se realiza el registro, pasándole como primer parámetro el nombre que lo representará y como segundo su definición en *JavaScript*. En el objeto *defaultComponents* se agregan las cualidades que nuestro primitivo personalizado obtendrá de forma predeterminada, en este caso su *geometry* se estable al valor *box*. En el segundo objeto se especifica como se enlaza el atributo *HTML* con la propiedad de componente. En el ejemplo se visualiza de forma intuitiva como los nuevos atributos *HTML depth*, *height* y *width* enlazan con las propiedades en cuestión.

4.2. Entidad-Componente-Sistema

Como comentamos anteriormente, esta es una de las cualidades especiales de *A-Frame* y en este apartado profundizaremos en su significado y la capacidad que le aporta.

Veamos que significa realmente este concepto que se basa en tres elementos:

- **Entidades**: son la base todos los objetos en la escena. Podríamos imaginarlos como esqueletos vacíos a los que adherimos componentes ya que, por sí mismos, no renderizan nada en la escena. Son representados por la etiqueta *<a-entity>*.

- **Componentes**: son propiedades reutilizables que pueden ser agregadas a las entidades, modificando así su apariencia, comportamiento o funcionalidad. Son representados por los atributos dentro de cada entidad. Podemos ver a las entidades como vehículos, avión, coche, barco... y a los componentes como cualidades, por ejemplo tener ruedas, en este caso, esta cualidad se la agregaríamos al coche y al avión, si estas propiedades fueran intrínsecas no podríamos manejarlas a nuestra conveniencia, por lo tanto, tendríamos un barco con ruedas. Es interesante añadir que podemos crear componentes personalizados como hicimos previamente con los primitivos.

- **Sistemas**: el objetivo de estos es manejar la lógica de nuestras entidades. Son parte opcional y son representados mediante los atributos del elemento *<a-scene>*.

4.3. Sintaxis

Conociendo estas características del lenguaje introduciremos su sintaxis. Podemos encontrar similitudes entre las entidades y los * * de *HTML*, así como las podemos encontrar entre los componentes y los estilos *css*. La sintaxis de estos componentes se realiza a través de los símbolos (:) y (;). Mediante el símbolo (:) separamos los nombres de las propiedades de sus valores, y mediante (;) separamos diferentes propiedades.

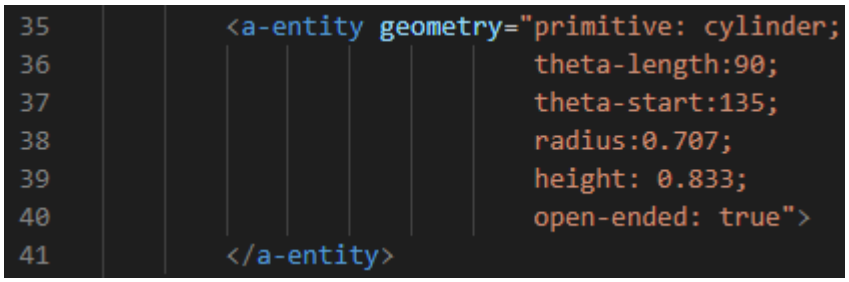


Figura 24: Sintaxis de un elemento entidad.

En este ejemplo, vemos como el nombre de la propiedad *geometry*, se separa de su instanciación mediante (:), y dentro de esta se separan las diferentes propiedades (*primitive*,*theta-lenght*,*theta-start*,*radius*,*height*,*open-ended*, mediante (;).

4.4. Manejo del *DOM* y eventos

A-Frame nos permite incorporar código *JavaScript* en él, ampliando cuantitativamente sus capacidades. Admite funciones que acceden al documento *HTML* y habilita diferentes opciones, aquí muestramos algunas básicas:

- *querySelector()*/*querySelectorAll()*: devuelve el primer elemento del tipo especificado. Agregando *a-box'* como argumento, devuelve el primer elemento que encuentre en el documento *HTML* de tipo *<a-box>*. Su variante *querySelectorAll()* devuelve un array con todos los elementos de ese tipo.

- *getAttribute()*/*setAttribute()*/*removeAttribute()*: devuelve el valor del atributo especificado, en el caso de un elemento *<a-box color=red»*, esta función con el atributo 'color' devuelve el valor 'red'. Se diferencia de su homólogo *setAttribute()* en que este último permite modificar el valor de atributo solicitado y afectará a la escena. *removeAttribute()* eliminará el atributo de esa entidad.

- *createElement()*/*appendChild()*/*removeChild()*: estos tres funciones juntas te permiten manejar la presencia de elementos en nuestra escena. Podemos crear un elemento con las características deseadas con *createElement()* y agregarlo a la escena o a otra entidad con *.appendChild()*. La última función nos permite eliminarlo.

Las funciones vistas en este apartado anterior adquieren mucho potencial al combinarla con los eventos. Los eventos son funciones que se activan ante ciertos sucesos. Algunos de estos eventos los lanza autónomamente el navegador como la interacción *click* (cuando pinchamos en un elemento con el ratón, *mouseover*/*mouseout* cuando el cursor pasa por encima o deja de pasar por un elemento, o, *submit* cuando se envía un formulario).

Existen dos funciones básicas para el manejo de eventos:

1. *emit(nombreDelEvento,detalles,bubbles)*: esta función hace que la entidad sobre la que se llame emita el evento especificado, con la información pasada en detalles. El último parámetro *bubbles* es un booleano que al activarse hace que este evento lo emitan también las entidades padres a esta.

2. *addEventListener(nombreEvento,función)*/*removeEventListener(nombreEvento,función)*: permite registrar y eliminar eventos asignándole un nombre y una función que realizarán cuando ocurra un *emit()*.

Figura 25: Codificación del evento cursor.

```
<body>
    <script>
        AFRAME.registerComponent('cambia-color', {
          schema: {
            color: {default: 'blue'}
          },

          init: function () {
            var el = this.el;
            var data = this.data;
            var defaultColor = el.getAttribute('material').color;

            el.addEventListener('mouseenter', function () {
              el.setAttribute('color', data.color);
            });

            el.addEventListener('mouseleave', function () {
              el.setAttribute('color', defaultColor);
            });
          }
        });
    </script>


 <a-scene>
<a-box color="#33FF68" position="0 1 -4" cambia-color="color: yellow"></a-box>

    <a-camera>
      <a-cursor></a-cursor>
    </a-camera>

 </a-scene>
<script>
</script>
</body>
```

Figura 26: Ejemplo de interacción mediante cursor.

Para este caso, he añadido el objeto cursor que nos indica en qué dirección estamos mirando. Se registra un componente que realiza la acción de cambiar de color la entidad cuando el cursor pasa por encima y cuando sale. Esto se realiza mediante las funciones explicadas en el apartado anterior.

4.5. Interacción con *three.js*

Debido a que *A-Frame* funciona por encima de *three.js* es necesario saber que debajo de cada entidad de la primera existe una enlazada a ella de la segunda. Esto es fundamental para entender *A-Frame* a bajo nivel. Vamos a introducir de manera liviana esta similitud, entendiendo que es escalable a todos los objetos de *A-Frame*.

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( {color: 0xffffff} );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

Figura 27: Entidad *<a-box>* en *three.js*.

La similitud de este código *three.js* en *A-Frame* sería un elemento *<a-box>*. En *three.js* para crear un objeto se realiza mediante el método *THREE.Mesh()*, pasando por parámetro la forma geométrica deseado y el tipo de material. Como se puede apreciar, es significativa la reducción de código en *A-Frame*. Importante conocer que mediante *three.js* podemos acceder y modificar los elementos de *A-Frame* a bajo nivel lo cual añade mayor manejo y posibilidades de nuestro código[12].

4.6. Manejo de controladores

Para integrar los controladores *hardware* (gafas, mandos, botones) *A-Frame* nos ofrece funcionalidades. Existe un componente llamado *tracked-controls* que es la base de todos los diferentes controladores del mercado. Cada controlador existente se enlaza con el objeto de forma diferente dependiendo de sus peculiaridades. Así por ejemplo si un controlador difiere de otro en un botón, esta funcionalidad se añade de forma dinámica.

La forma en la que podemos manejar el uso de los controladores es mediante los eventos que se les añaden. Algunos de ellos son *buttonup* (botón sin pulsar), *buttondown* (botón pulsado) o *axischanged* (modificar la posición del mando).

4.7. Integración de modelos

Finalmente la parte esencial de la aplicación, la visualización de modelos *gltf* o *glb* (formatos para archivo de escenas y modelos 3D). Estos serán los modelos que será capaz de representar la aplicación mediante un elemento propio de *A-Frame* para su integración.

```
<a-assets>
    <a-asset-item id="escena" src="./directorio/escena.gltf"></a-asset-item>
</a-assets>
```

Figura 28: Pre-carga de modelo.

Mediante estas instrucciones se carga el modelo en caché desde el inicio para reducir coste computacional. Se le asigna un identificador para poder asignarlo posteriormente en nuestra escena.

```
<a-entity gltf-model="#escena"></a-entity>
```

Figura 29: Cómo agregar un modelo a una entidad.

Envolviendo estas instrucciones en el objeto principal de *A-Frame*, *<a-scene>*, e integrando este archivo en un servidor local podemos visualizar sin interacción alguna nuestra escena.

Figura 30: Escena sin interacción.

Posteriormente, hay que añadir a este elemento la capacidad de interaccionar con nuestro controlador. Para ello obtenemos un modelo *gltf* animado de la red, en este caso se han seleccionado de la web *https://sketchfab.com/feed*.

En este caso, para obtener esta interacción crearemos un componente y se lo agregamos a la entidad del modelo. Con el objetivo además, de su reutilización para otros modelos. Este nuevo componente agrega dos nuevos eventos a la entidad, por lo tanto, esta quedará esperando a estos sucesos. En primer lugar, agregaremos un evento enlazado al *click* que activará la transición a la pose de nuestra animación. Esto visualmente se reflejará como la transición de una pose a otra. En nuestro caso tenemos dos poses asignadas al modelo de la puerta, una cerrada y otra abierta. Sin embargo, aisladamente este método solo servirá para que nuestro modelo realice la animación una única vez debido a que en el próximo *click* ya tendrá la pose final agregada, debemos eliminar nuestro componente. Realizamos esto incluyendo un nuevo evento el cual espera la finalización de la animación, y en ese momento elimina nuestra componente.

```
AFRAME.registerComponent('animar', {
    schema: {
      take: {default: '*'}
    },

    init: function () {
      var data = this.data;
      var el = this.el;

      el.addEventListener('click', function () {
        el.setAttribute('animation-mixer', {
          clip:  data.take,
          loop: 'once'
        });

      });

      el.addEventListener('animation-finished', function () {
        el.removeAttribute('animation-mixer');
      });

    }
  });
```

Figura 31: Escena con interacción.

Una vez se logra su activación mediante un evento cualquiera, su traducción al funcionamiento del hardware de realidad virtual es simple. Como ya se comentó en la sección anterior 4.6, existen diversos eventos asignados a nuestro *hardware*, por lo tanto solo tendremos que modificar el evento que activa la animación por el deseado como se muestra en el desarrollo.

5

Desarrollo del proyecto

5.1. Iteración 0

Esta es la iteración inicial en la que se documenta las capacidades sostenidas por la aplicación. En esta fase nacen las propuestas y los requerimientos del cliente, en este caso, estos fueron debatidos y decididos junto al tutor. En el encuentro se discuten cuáles eran los requerimientos que debe tener nuestra aplicación, roles, experiencia de usuario y necesidades de la misma.

En primer lugar se definen los diferentes perfiles de usuario que tendrán acceso a la aplicación. Además de especificar qué funcionalidades los distinguen. Se pueden diferenciar dos roles y, aunque sus capacidades sean parecidas, lo importante de esta implementación es la posibilidad de aumentar en un futuro las funciones de ambos teniendo establecida esta base.

- **Usuario**: tiene permiso para registrarse en la aplicación, ingresar en la misma, tiene acceso libre a la aplicación para visualizar modelos ya existentes y además, interactuar con los modelos que lo permitan.

- **Administrador**: además de las capacidades de un usuario común, este rol permite la funcionalidad de añadir modelos modelos a la base de datos. Esta funcionalidad es primordial ya que asigna un código al mismo, con el que ambos roles son capaces de visualizar el modelo mediante la aplicación.

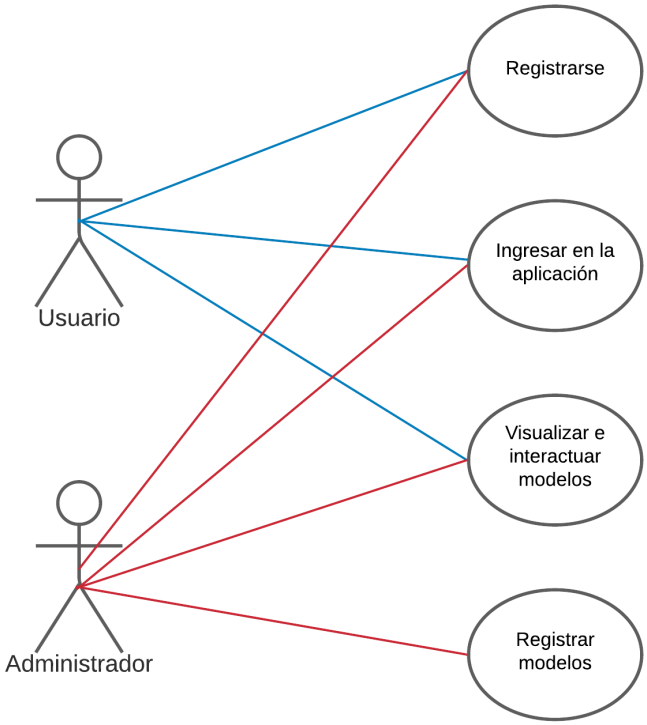


Figura 32: Casos de uso.

En segundo lugar, se realiza el análisis de requisitos. De este modo se crea la base sobre donde se construirán los cimientos de la aplicación. A continuación, se muestran especificados ordenados por iteraciones junto a su correspondiente prioridad. Esta prioridad irá disminuyendo a medida que las iteraciones avancen, pues en un estado avanzado de la aplicación, las funciones añadidas son meros ajustes o retoques.

Iteración 1	Prioridad
RF 1 - El sistema implementa el registro de usuarios en la aplicación basado en usuario y contraseña.	ALTA
RF 1.1 - El sistema implementa el ingreso de usuarios ya registrados en la aplicación.	ALTA
RF 2 - El sistema permite al usuario acceder a la visualización de cualquier modelo existente en la base de datos.	ALTA
RF 2.1 - Se habilita la capacidad de moverse por el entorno 3D con el controlador *Oculus Go*	ALTA

Tras esta iteración tenemos acceso a las capacidades básicas de nuestra aplicación, a continuación, se muestran las añadidas en la iteración 2, cuyo objetivo es agregar el rol de administrador.

Iteración 2	Prioridad
RF 3 - La base de datos permite distinguir entre usuarios y administradores.	ALTA
RF 4 - El sistema implementa el registro de modelos asignados a un código de referencia	ALTA
RF 4.1 - Un código sólo puede ser registrado una vez.	ALTA

Finalmente, en la iteración 3 se realizan ajustes sobre la aplicación para añadir pequeñas funcionalidades. El objetivo de esta es acercar la aplicación a las necesidades actuales.

Iteración 3	Prioridad
RF 1.2 - Sólo los usuarios registrados en la aplicación pueden acceder a ella.	ALTA
RF 1.3 - No se puede registrar el mismo nombre de usuario más de una vez	MEDIA
RF 1.4 - No se puede acceder a la aplicación si la contraseña especificada es errónea	MEDIA
RF 4.2 - No se permite acceder a un código de modelo no registrado.	ALTA
RF 5 - La navegación entre vistas mediante URL no se permite, se redirigirá a la pantalla de ingreso	BAJA

A continuación, se muestran los modelos de objetos que se almacenan en la base de datos de nuestra aplicación. En este caso al ser una base de datos *NoSQL* no hay relaciones entre los objetos, simplemente especificamos sus atributos y claves primarias.

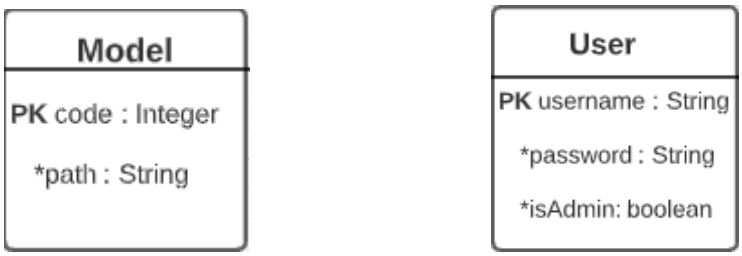


Figura 33: Modelos de la base de datos.

Se define también la navegación entre páginas. Para ello se crea un diagrama de navegación cuya función es especificar las transiciones entre páginas y cómo se realizan.

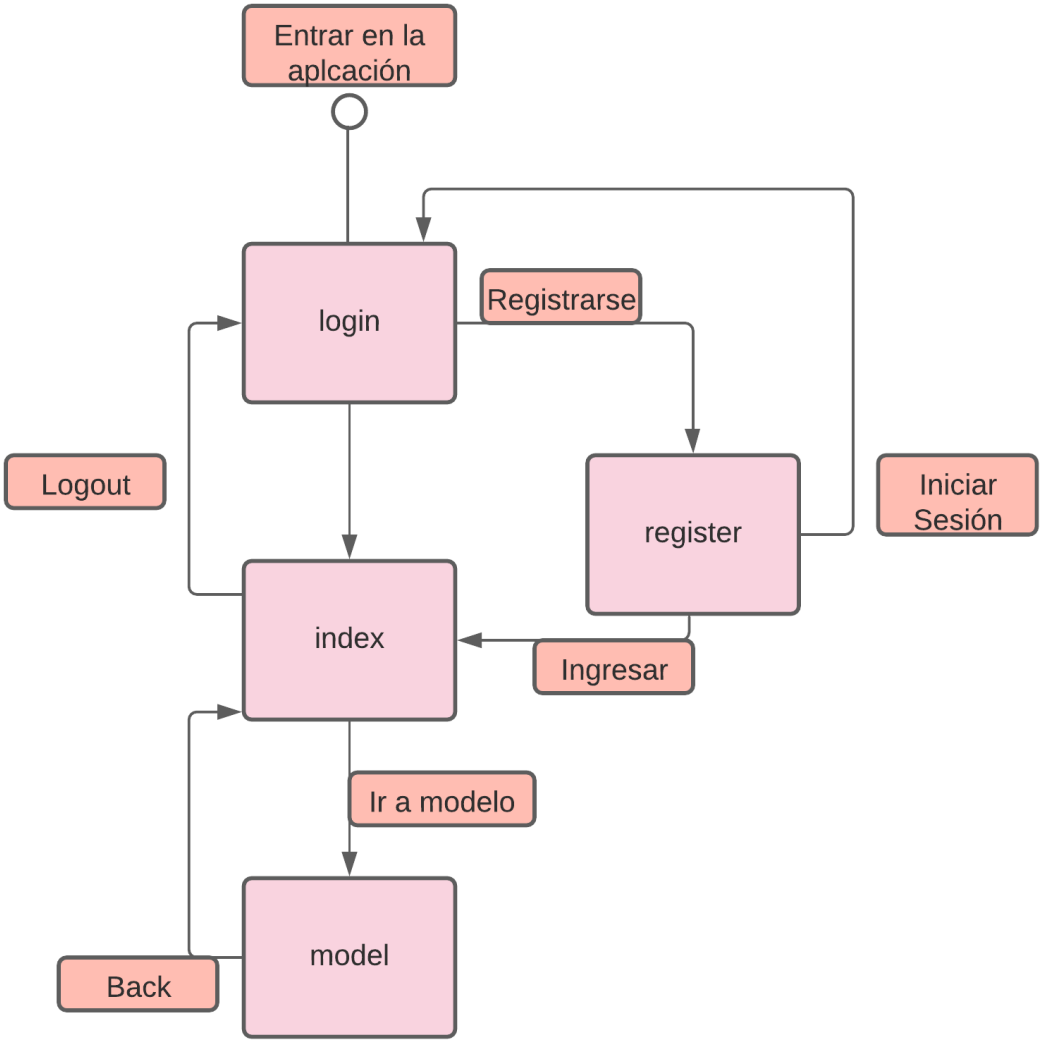


Figura 34: Diagrama de navegación.

Estando definidos todos los requisitos base de la aplicación, se procede a su implementación. Se comienza por la creación de la base de datos y los modelos de datos ya definidos. Funciona en *MongoDB* y se usa la herramienta *MongoDB Atlas* para su funcionamiento en la nube.

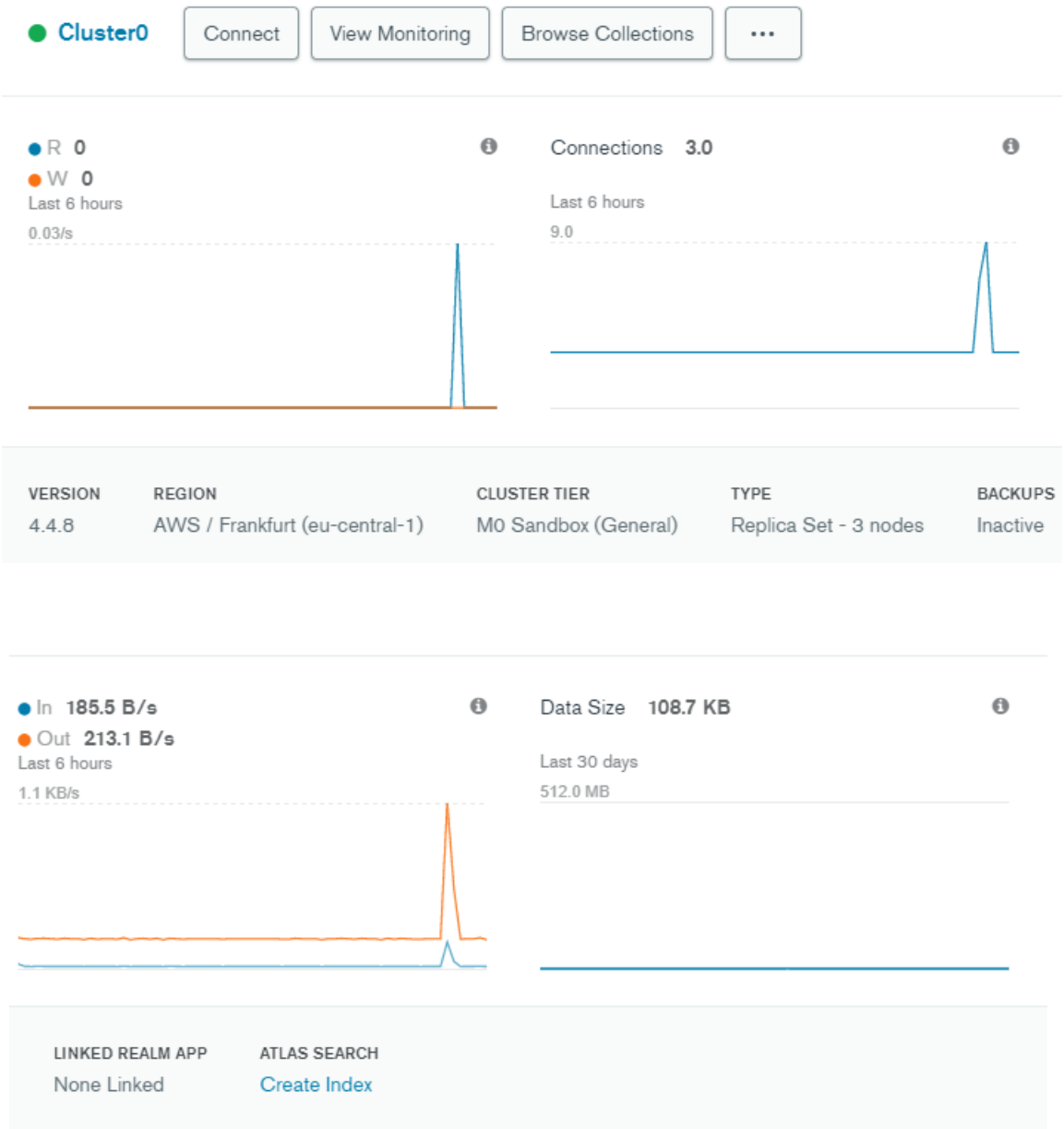


Figura 35: Información de la base de datos.

Esta es la base de datos inicializada y se puede ver la información que nos proporciona la misma herramienta. Por orden de aparición de izquierda a derecha significan: cantidad de

procesos de lectura y escritura, conexiones a la base de datos, cantidad de datos extraída e incluida y cantidad de datos almacenados.

El siguiente paso es modelar los esquemas de la base de datos e iniciar la conexión con la misma. En este caso se utiliza *Mongoose* que es una librería para *Node.js* que nos permite escribir consultas para una base de datos de *MongoDB*. La parte fundamental de *Mongoose* es la definición del esquema de los objetos como se muestra a continuación.

```
4  const modeloSchema = new Schema({
5      code: {
6          type: Number,
7          unique: true,
8          required: true,
9          trim:true
10     },
11     url: {
12         type: String,
13         default: '',
14         trim:true
15     }
16 });
17
18 module.exports = mongoose.model('Modelos', modeloSchema)
```

```
6  const userSchema = new Schema({
7      local:{
8          username: String,
9          password: String,
10     admin:{
11         type: Boolean,
12         default:false,
13         }
14     }
15 });
16
17 module.exports = mongoose.model('User', userSchema);
```

Figura 36: Código referente a los esquemas de la base de datos.

Finalmente, nos conectamos a la base de datos mediante la instrucción *connect* en la cual debemos introducir nuestro usuario administrador previamente creado en *MongoDB Atlas*.

```
28  mongoose.Promise = global.Promise
29  mongoose.connect('mongodb+srv://admin:12345@cluster0.5elcn.mongodb.net/TFG?retryWrites=true&w=majority', {
30      useNewUrlParser: true,
31      useUnifiedTopology: true
32  })
```

Figura 37: Conexión a la base de datos.

5.2. Iteración 1

Conociendo los requisitos base y teniendo en marcha el funcionamiento de nuestra base de datos, en esta etapa se comienza el desarrollo a nivel de código de la aplicación. Se tiene como objetivo un funcionamiento básico donde el usuario podrá registrarse en la aplicación y visualizar los entornos de realidad virtual existentes en la base de datos. A continuación, se explican los diferentes pasos en la integración de las gafas en la aplicación, además se divide esta sección en las diferentes vistas creadas en esta iteración.

5.2.1. Integración de *Oculus Go*

Sin duda este fue el apartado más frustrante de la aplicación, pues la escasa información acerca de como integrar funcionalmente el hardware hizo necesario una investigación del tipo ensayo y error.

En primer lugar, se plantea visualizar el escenario más simple posible, un simple plano estático de forma local. Simplemente agregando esta entidad al documento *HTML* y ejecutándolo era capaz de visualizarlo. Sin embargo, para la visualización mediante *Oculus Go* no resultó tan sencillo.

Tras investigar, aparece *Firefox Reality* como el navegador que realiza la transmisión entre el computador y las gafas. Tras comprobar que este dejó de ser compatible con *Oculus Go* se decide optar por una aplicación propia de *Oculus*, *Oculus Desktop*. Esta aplicación se conecta a la computadora y la transmite a las gafas, lo cual parece ideal pues el código se visualiza correctamente. Por segunda vez fue un fracaso, en esta ocasión si era capaz de visualizar la escena, pero no ocurría la integración a nivel visual. Es decir, la inmersión en la escena no ocurría, simplemente se visualizaba en 2D. Finalmente se encuentra un editor de código *online*,

Glitch, recomendado para el desarrollo web en realidad virtual. Gracias a este se es capaz de visualizar mi escena simple en las gafas de forma apropiada. Recalcar que mediante el uso de *Glitch*, se puede visualizar escenas simples en los principales navegadores.

El siguiente paso es visualizar un modelo *gltf* pues era la base del proyecto. Integrar esto en *Glitch* era posible pero no la creación de una aplicación web. Por lo tanto se busca otra alternativa. Se opta por montar un servidor local mediante *XAMPP*, en la búsqueda encontramos como otros modelos de hardware *VR* eran capaces de conectarse al servidor local. En este caso, tras diversos intentos, tampoco se puede mediante este método pues a pesar de conectar las gafas a la computadora y probar las diferentes aplicaciones no parecía existir la compatibilidad.

Finalmente, tras indagar en esta necesidad, y sabiendo que la integración funcionaba en aplicaciones web de realidad virtual existentes, la única opción era crear una en la nube. Con este cometido se seleccionó *Heroku*, el cual ofrece la función de crear aplicaciones web alojadas en la nube con ciertas limitaciones. Tras aprender esta tecnología, se crea la aplicación a la que se puede acceder a través de cualquier dispositivo (incluido las gafas) mediante esta *URL* *https://webvr-tfg-app.herokuapp.com/*.

5.2.2. Pantalla de Registro

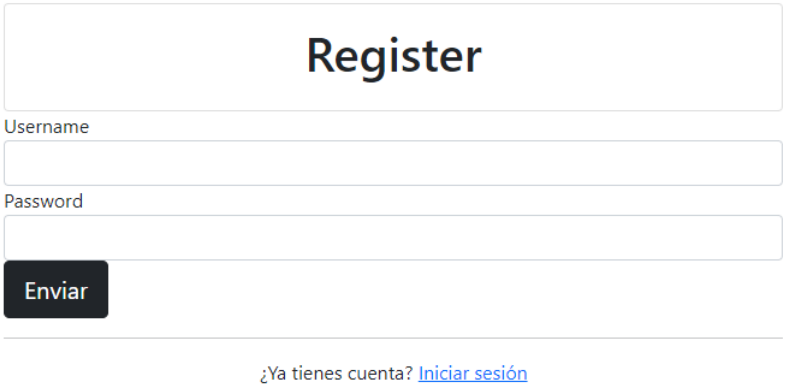


Figura 38: Vista de registro.

Esta pantalla esta conformada por un registro de usuario al uso. Contiene un formulario donde se introduce el usuario y la contraseña correspondiente al mismo. Además, esta pantalla debe almacenar este nuevo usuario en la base de datos cumpliendo con las restricciones impuestas. Además, esta pantalla implementa un botón que redirige al usuario a la pantalla de Ingreso, en el caso de que ya estuviera registrado, deberá ingresar en esta pantalla.

5.2.3. Pantalla de Ingreso

Esta pantalla en cuestión es bastante similar a la pantalla de registro especificada anteriormente. Contiene un formulario donde ingresar el nombre de usuario y la contraseña. En este caso se debe asegurar que los credenciales introducidos corresponden con los almacenados en nuestra base de datos, recalcar que las consultas a la base de datos son posibles gracias a la librería *Mongoose*. Tras finalizar satisfactoriamente la validación, se redirige al usuario a la pantalla *Home* de la aplicación.

Además, para complementar la función del registro, se implementa un botón para acceder a la pantalla de registro en caso de que el usuario no conste de una cuenta registrada.

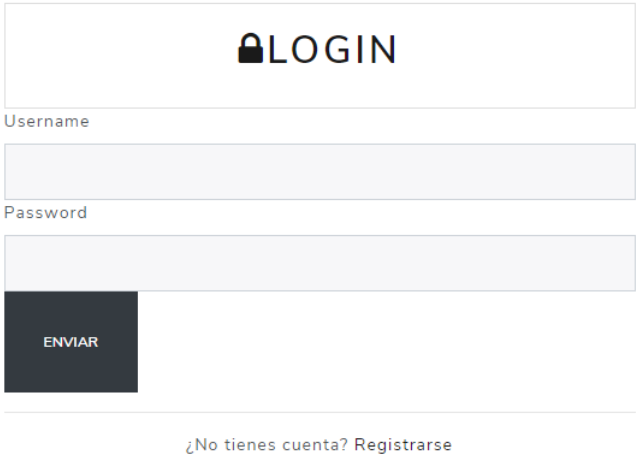


Figura 39: Vista de ingreso.

5.2.4. Pantalla *Home* para Usuario

En esta pantalla solo aparecerá un formulario que consta de un código y un botón. Este código está enlazado a un modelo de realidad virtual.

Figura 40: Vista *Home* tipo usuario.

Para el funcionamiento de esta pantalla es necesaria la existencia de objetos de tipo modelo en la base de datos. Sin embargo, esta función esta encomendada al administrador y se implementa en futuras iteraciones, por lo que en esta, introduciremos el objeto directamente en la base de datos.

```
_id: ObjectId("6122940c7084764e84802bc7")
url: "/modelos/1629656076373.gltf"
code: 1
__v: 0
```

Figura 41: Ejemplo de objeto tipo modelo en *MongoDB Atlas*.

Vemos que el objeto se compone de 4 atributos. El atributo *code* almacena el código enlazado al modelo, mientras *url* indica la dirección y el nombre con el que se guarda el mismo. Esta función de almacenamiento merece la mención de dos librerías de *Node.js*, *multer*, que nos permite cargar archivos, y *mime-types*, que permite identificar la extensión de los mismos. El resto son atributos usados por *Mongoose* y *MongoDB*. El atributo *_id* identifica de manera única a un elemento de la base de datos, y *__v* contiene información sobre el versionado, en mi caso no se hará uso de ellos.

5.2.5. Pantalla de visualización de modelo

Tras los problemas expuestos en el apartado 5.2.1 esta vista fue sencilla simplemente muestra el modelo requerido de forma dinámica gracias a *EJS*.

5.3. Iteración 2

En esta fase se implementa el rol de administrador, el cual posee la función de agregar pares código/modelo a la base de datos. Para esto se implementan cambios en la pantalla de ingreso, y agregaremos una vista adicional a la pantalla *Home* para el rol de administrador.

5.3.1. Pantalla de Ingreso

Se realiza un cambio en la lógica interna de la pantalla. Al tomar los credenciales y autenticar la entrada, se revisa el atributo *admin* almacenado en la base de datos. Dependiendo de su valor se le permitirán las funciones pertinentes.

5.3.2. Pantalla de *Home* vista administrador

Para el rol de administrador en esta pantalla se agrega un formulario adicional cuyo objetivo es el registro de modelos. Este formulario consta de un código y un botón para seleccionar un modelo perteneciente al dispositivo. Con este botón de tipo archivo, somos capaces de cargar el modelo, pero a partir de ahí se realiza un proceso de almacenamiento para el que requerimos de las librerías *multer* y *mime-types*. La biblioteca *multer* nos sirve como *middleware* para tratar los archivos que obtenemos mediante el botón, nos proporciona diferentes opciones como por ejemplo, elegir el directorio al que van a parar. Por otro lado, *mime-types* es una extensión que permite identificar la extensión del archivo y actuar en consecuencia para evitar errores de incompatibilidades.

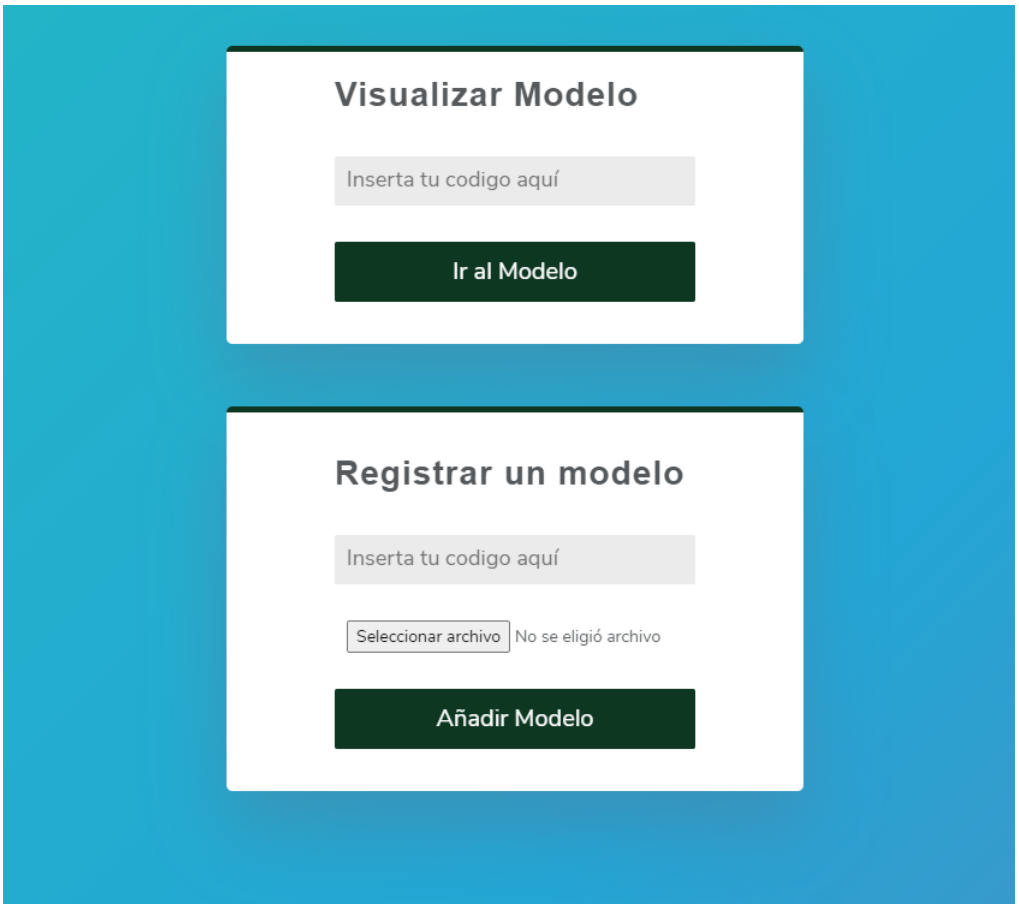


Figura 42: Vista Home tipo administrador.

5.4. Iteración 3

En esta última iteración se liman detalles de la implementación y se introducen diferentes mensajes para informar al usuario del estado de sus acciones. Importante destacar las funciones añadidas en la seguridad de la aplicación y la importancia de dar *feedback* al usuario sobre lo que acontece. Al finalizar la implementación se puede mostrar el diagrama de secuencia. El cual permite observar el flujo de la aplicación, desde a dónde nos dirigimos cada vez que realizamos una acción, hasta cuando y cómo se realizan las consultas a la base de datos.

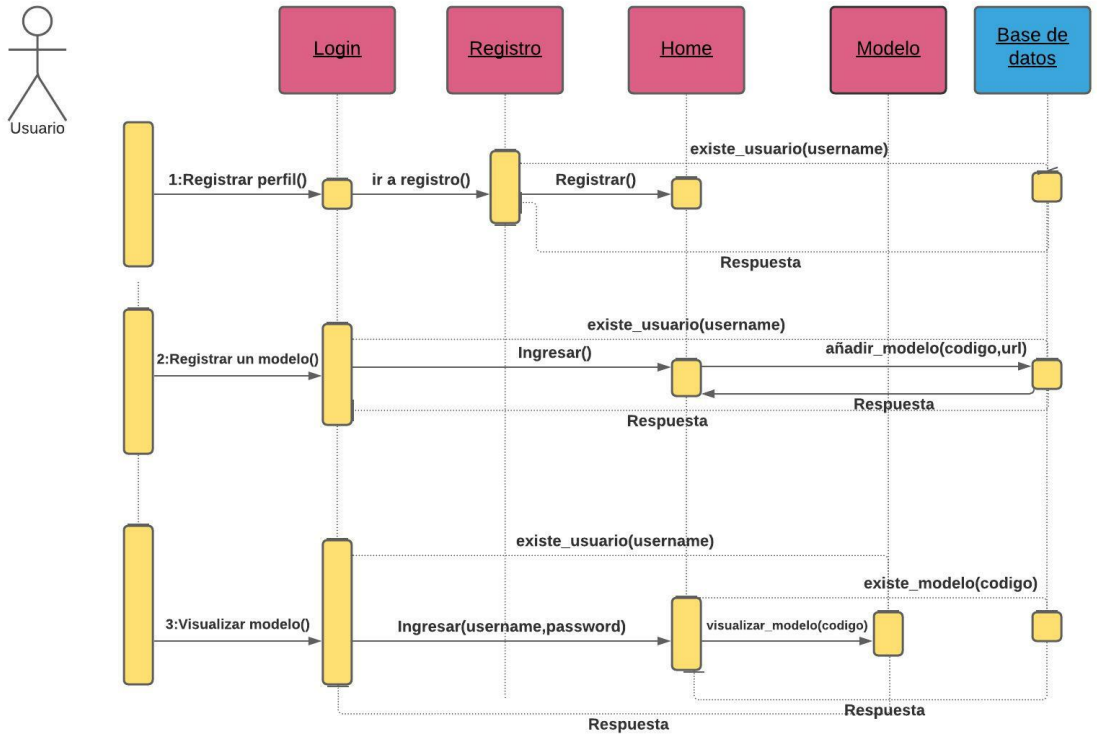


Figura 43: Diagrama de secuencia.

5.4.1. Pantalla de Registro

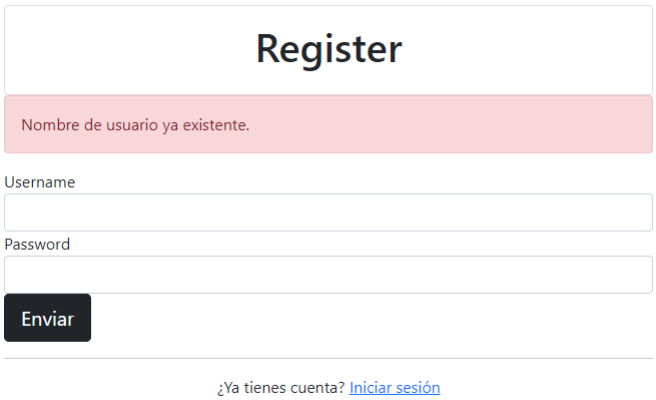


Figura 44: Ejemplo de mensaje mostrado al usuario.

El intento por parte de un usuario de registrarse con un *username* ya existente en la base de datos será manejado imprimiendo un mensaje en pantalla.

5.4.2. Pantalla de Ingreso

En esta pantalla se muestran nuevos mensajes en el caso de intento de ingreso con un usuario no existente e intento de ingreso con usuario existente y contraseña errónea.

5.4.3. Pantalla *Home* vista administrador

En este caso también se introducen dos mensajes de información a la hora de registrar un modelo. Tanto para en el caso de una correcta como de una fallida inserción.

5.4.4. Seguridad

Se desarrollan dos funciones esenciales para la preservación de esta seguridad, ambas vinculadas al esquema de *User*. Una función *generateHash* cuyo objetivo es realizar el *hash* de la contraseña asociada a nuestro usuario para almacenarla encriptada en la base de datos. En segundo lugar, una función *validatePassword* con la capacidad de comparar los *hashes* de nuestra contraseña introducida con la almacenada. Ambas hacen uso de la librería *bcrypt-nodejs* que nos permite mecanismos criptográficos en *Node.js*.

Para darle un carácter de seguridad a la aplicación se implementan mecanismos mediante *Passport*. Las funciones complementarias *serializeUser* y *deserializeUser*, las cuales mediante *Passport* almacenan los datos del usuario en la sesión actual.

Por otro lado, dos funciones vinculadas al inicio de sesión y al registro de usuario. Ambas son similares con ligeros cambios. Al introducir una dupla usuario contraseña se busca este usuario en la base de datos y en caso de que exista se realizan diferentes acciones en ambos casos. En el caso del inicio de sesión, la existencia de usuario y una validación correcta de la contraseña derivan en un inicio de sesión satisfactorio. Sin embargo, en el registro, la existencia del usuario será penalizada con un mensaje de error en la vista.

Para finalizar, se ha implementado mediante un seguimiento de la sesión, que el usuario no pueda introducir directamente la dirección en la *url* y acceder a una ventana a la que no tiene permisos.

5.5. Funcionalidades *Oculus Go* y su aplicación en modelos integrados

En esta fase final es crucial trabajar en la interacción con los modelos a nivel de desarrollo. Aquí se explican las capacidades y posibilidades que ofrece *Oculus Go* a los desarrolladores, y cómo se puede implementar en nuestra aplicación.

- Giroscopio y acelerómetro: Esta funcionalidad permite al sistema reconocer la orientación del usuario y la velocidad a la que rota. Estos dispositivos permiten mostrar al sujeto la sección de la escena a la que corresponde su orientación.

- *RayCaster*: Se puede definir como el hilo imperceptible que conecta el mando con su proyección en la escena. Mediante esta funcionalidad el software es capaz de reconocer el lugar donde apunta el mando. Podemos manejar esta funcionalidad hardware mediante el componente *raycaster* en *three.js*, el cual tiene vinculado ciertos eventos como *intersected* (intersección con un elemento) o *intersectedcleared* (fin de la intersección con un elemento).

- *TrackPad*: Parte de la funcionalidad del mando que nos permite realizar movimientos por la escena. Permite movimientos en todas direcciones.

Figura 45: Controlador *Oculus Go*.

Este permite lanzar los siguientes eventos: mover hacia abajo (*touchpaddown*),mover hacia arriba (*touchpadup*), comenzar el movimiento (*touchpadtouchstart*), detener el movimiento (*touchpadtouchend*), movimiento (*touchpadmoved*), modificar la dirección (*touchpadchanged*).

- *Trigger*: Este no es más que un botón situado en la parte frontal del mando, su existencia permite la asociación de un evento a su acción. Permite la interacción mediante estos tres eventos: modificar el botón (*triggerchanged*), pulsar el botón (*triggerdown*), dejar de pulsar el botón (*triggerup*).

Tanto el giroscopio como el *trackpad* son elementos intrínsecos de la experiencia, que adquieren sus funciones (visión y desplazamiento por el modelo) al adherir el controlador de *Oculus Go*. Destacar la incorporación de estas funcionalidades en el proyecto. Por un lado, se adhiere la interacción con el modelo al accionamiento del *trigger*.

```
AFRAME.registerComponent('animar', {
    schema: {
      take: {default: '*'}
    },

    init: function () {
      var data = this.data;
      var el = this.el;

      el.addEventListener('triggerdown', function () {
        el.setAttribute('animation-mixer', {
          clip:  data.take,
          loop: 'once'
        });

      });

      el.addEventListener('animation-finished', function () {
        el.removeAttribute('animation-mixer');
      });

    }
  });
```

Figura 46: Modificación para interacción *trigger*.

Por otro lado, usamos la funcionalidad *raycaster* para añadirla a nuestro modelo y así consta de una referencia de objetivo.

```
<a-entity laser-controls="hand: right"></a-entity>
```

Figura 47: Elemento *raycaster*.

5.6. Aplicación móvil

Para este cometido se usa la aplicación *AppsGeyser* y no implica ningún cambio en nuestro código ni afecta al funcionamiento de nuestra aplicación web. Esta simplemente crea una aplicación móvil cuyo uso nos redirecciona a nuestra aplicación web.

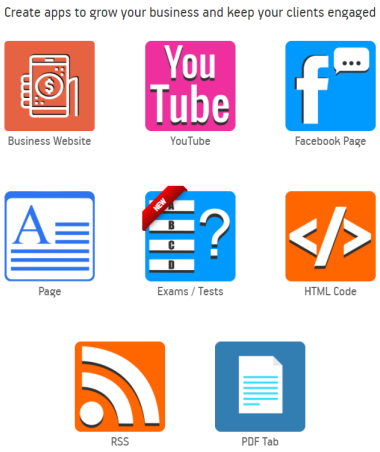


Figura 48: Panel de selección de plantilla.

La hoja de ruta a seguir es simple. Accedemos a su web *https://appsgeyser.com/*, y pulsamos el botón de *Create App*. Posteriormente se muestra una pantalla para seleccionar la vista deseada para nuestra aplicación.

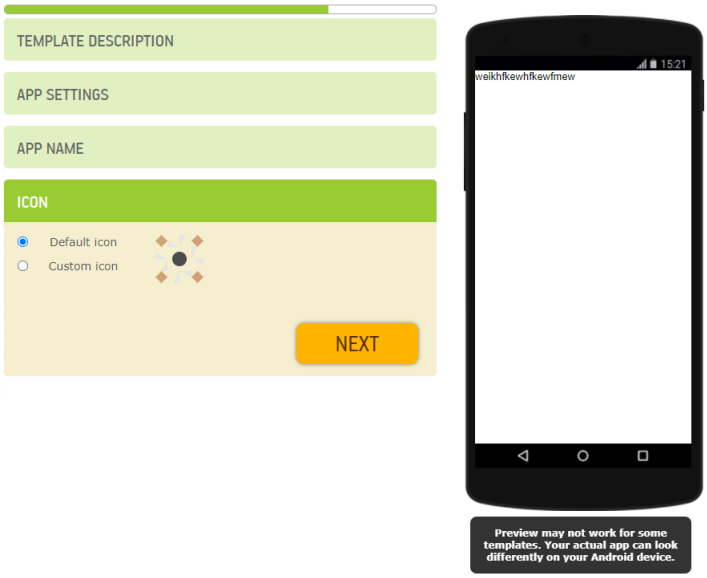


Figura 49: Vista previa de la aplicación.

Aquí se selecciona la opción para código HTML personalizado y posteriormente se abre una vista de pre-visualización de nuestra aplicación, donde podremos seleccionar su dirección web, su nombre y el icono asociado a la misma. Finalmente, confirmando la creación se nos añade la aplicación móvil en nuestro gestor de aplicaciones.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

La principal aportación de este proyecto, es en materia de investigación, aborda el panorama de la realidad virtual hoy en día, cómo de accesible es y sus posibilidades.

Es necesario contrastar nuestro resultado final con los objetivos perseguidos y planteados desde un principio. Se ha realizado una investigación consistente sobre los *frameworks* actuales, desde mi punto de vista lastrada por la escasez de documentación en la red, aún así, se ha seleccionado un entorno apropiado para desarrollar la aplicación.

Por otro lado, se ha logrado un producto final funcional donde el usuario puede obtener la experiencia planteada inicialmente. La aplicación web principalmente permite la visualización de modelos, pero además, se han incorporado tecnologías como el manejo de base de datos o tecnologías *back-end* que le aportan completitud al trabajo. El único pero sería la integración de modelos interactivos mediante la estandarización de los mismos.

Se ha realizado un estudio y comparación para ofrecer un punto de vista cercano sobre las diferencias existentes entre el uso de un dispositivo móvil o de hardware *VR* para experimentar el mundo de la realidad virtual. Desde mi punto de vista, tras haber usado ambas herramientas condensaría este apartado señalando que, si tu propósito es realmente embarcarte en el mundo de la realidad virtual y sus infinitas posibilidades, preferiblemente debes optar por el hardware *VR*, si por el contrario, sólo tienes curiosidad, es mejor comenzar obteniendo unas *Google CardBoard* a modo de introducción.

Como último objetivo, la aplicación móvil. En mi opinión este es el ámbito a mejorar como línea futura en el proyecto. Por las restricciones de tiempo, no se ha podido desarrollar una aplicación móvil nativa, he tomado un generador de aplicaciones. Aun así finalmente cumple su función de todas maneras.

Tras varios meses de trabajo finalmente se ha concluido el proyecto, debo destacar el sentimiento de orgullo y satisfacción personal debido al trabajo y la implicación puesta en este proyecto. Se han cumplido en su mayoría las expectativas y objetivos planteados desde un principio. Sin embargo, hay ciertas dificultades que han surgido en el transcurso y que teniendo consciencia de antemano podrían haber modificado el planteamiento inicial.

Pienso que parte fundamental de este proyecto es superar las adversidades de forma autónoma, saber adaptarse a las novedades y aprender a desenvolverse, preparándonos así para el mundo laboral.

Previo a la realización de este trabajo, mi conocimiento sobre la realidad virtual era muy escaso y la veía muy lejana. Tras haber desarrollado e investigado en ella destaco que es más accesible de lo que en principio pensaba y, sobretodo en *A-Frame* hay cantidad de código libre y comunidad a tu disposición.

6.2. Líneas Futuras

Por último, tras haber finalizado el proyecto es importante ser consciente las posibles mejoras implementables.

- Estandarización de modelos interactivos: teniendo el trabajo de investigación funcional y el conocimiento sobre cómo exponerlo, es necesario lograr una estandarización de estos modelos para poder integrarlos en la aplicación como el resto de modelos y no tratarlos de forma aislada.

- Personalización: una vez implementado el sistema de cuentas, es posible expandirlo creando perfiles personalizados, y además poder asignar modelos a cada perfil para que solo ellos puedan ver sus propios modelos.

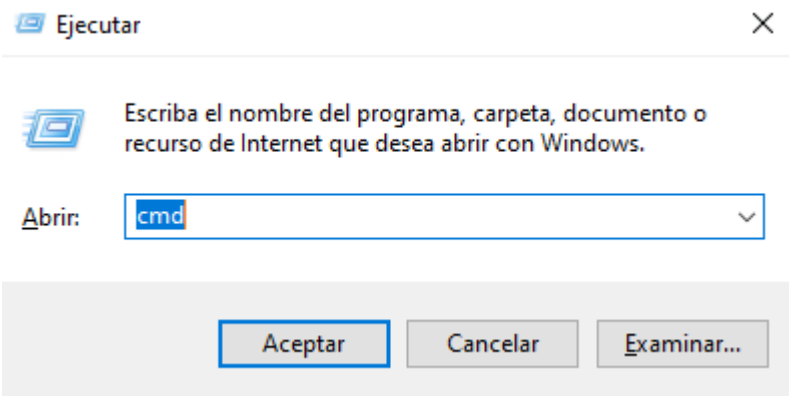
- Mejora de compatibilidad: actualmente la aplicación solo es compatible con dispositivos móviles y las gafas de realidad virtual *Oculus Go*, pero *A-Frame* proviene de capacidad para ser compatible con el resto de hardware de realidad virtual.
- Aplicación móvil: es cierto que se ha hecho una adaptación a una aplicación móvil pero esta realmente te redirige a la web. Se podría crear una aplicación compatible con *iOS* y *Android* nativa de este lenguaje.
- Catálogo de modelos: sería interesante implementar un catálogo de modelos con una breve descripción para poder visualizar las posibilidades que ofrece la realidad virtual.

Apéndice A
Manual de Instalación

A.1. Despliegue del servidor en local

1. En primer lugar, descomprimir el proyecto adjuntado en el directorio deseado, *https://github.com/poldiaz*

2. Abrir el comando de diálogo Ejecutar de *Windows* mediante el atajo de teclado *Windows + R*. En este escribir *"cmd"* y pulsar Aceptar, se abrirá la consola de comandos.



3. Insertar en la consola de comandos la instrucción "cd" seguido de la ruta donde se encuentra almacenada la carpeta del proyecto.

4. Realizamos la instalación de todas las dependencias del proyecto mediante el uso del comando *"npm install"*.

5. Comprobar el fichero raíz *index.js*, debe aparecer esta parte del código de la siguiente forma, con el objetivo de hacer el despliegue de forma local.

```
64  //SERVIDOR LOCAL
65  server.listen('3000', function() {
66      console.log('Servidor web escuchando en el puerto 3000');
67    });
68
69  /*SERVIDOR HEROKU
70  server.set('port',process.env.PORT || 3000);
71
72  server.listen(server.get('port'), () => {
73      console.log('Server on port' ,server.get('port'))
74  })
75  */
```

6. Para iniciar el servidor finalizaremos con el comando *"npm start"*. Finalmente abriremos en el navegador la siguiente dirección donde estará alojada nuestra aplicación. *http://localhost:3000/*

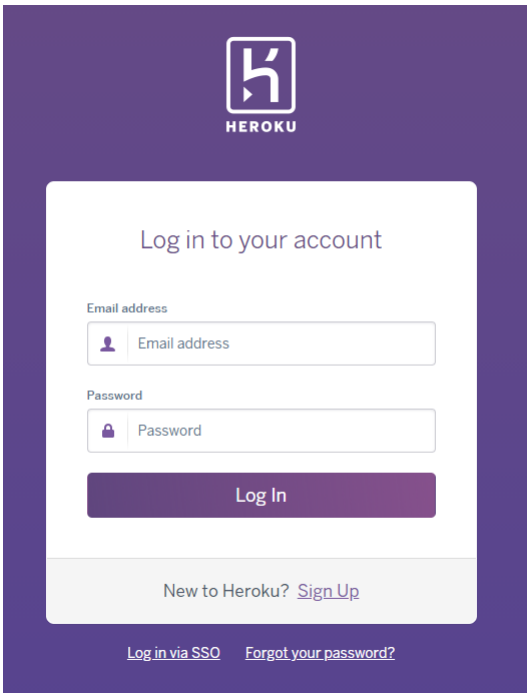
A.2. Despliegue en aplicación *Heroku*

1. Realizaremos hasta el paso 4 (inclusive) del manual para despliegue del servidor local.

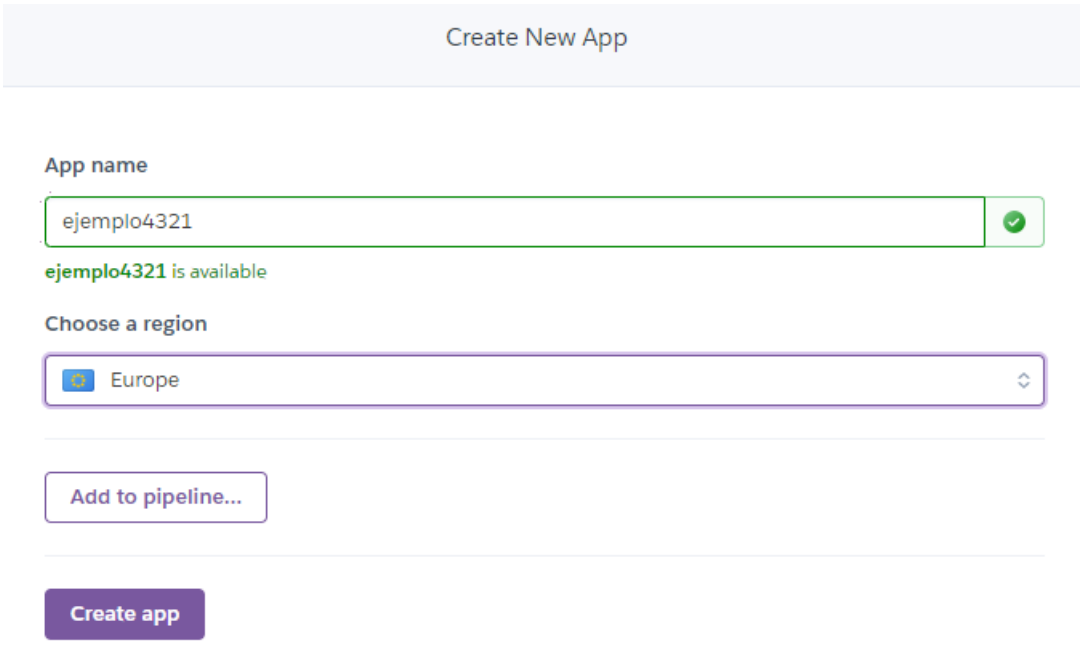
2. Comprobar el fichero raíz *index.js*, debe aparecer esta parte del código de la siguiente forma, con el objetivo de hacer el despliegue en nuestro servidor *Heroku*.

```
64  /*SERVIDOR LOCAL
65  server.listen('3000', function() {
66      console.log('Servidor web escuchando en el puerto 3000');
67    });
68  */
69  //SERVIDOR HEROKU
70  server.set('port',process.env.PORT || 3000);
71
72  server.listen(server.get('port'), () => {
73      console.log('Server on port' ,server.get('port'))
74  })
75
```

3. Comenzaremos creando nuestra cuenta propia en la aplicación web *Heroku*. Aquí dejo donde se puede realizar. *https://id.heroku.com/login*



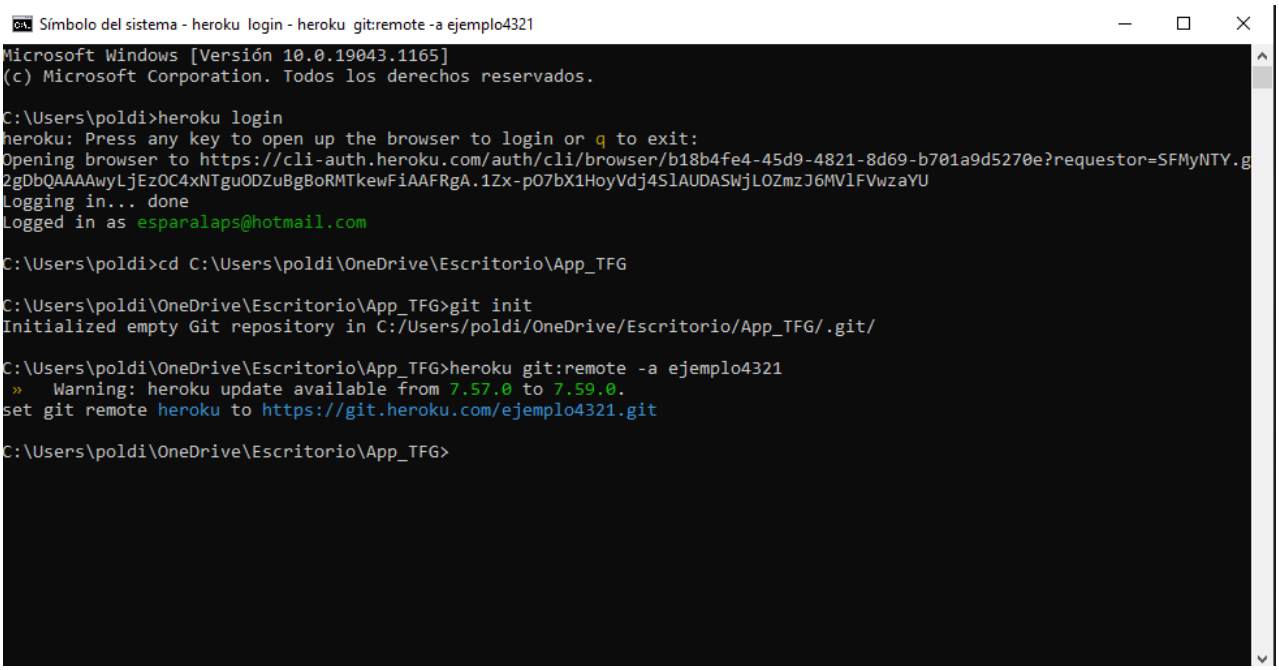
4. Seleccionamos el botón *Create new app* y procedemos a rellenar el formulario indicando el nombre que queremos para nuestra aplicación web.



5. Una vez aquí, descargamos e instalamos *Heroku CLI*, enlace que se encuentra en la pantalla principal tras ingresar en la página. Procederemos a la compilación y subida de nuestra aplicación.

6. Deberemos ingresar en *Heroku* mediante el comando *"heroku login"* en el cmd, se abrirá la web para confirmar el ingreso.

7. En el caso de ser la primera vez, debemos inicializar nuestro repositorio, accedemos mediante consola de comandos a el directorio deseado y ejecutamos los siguientes comandos.



8. Finalmente realizaremos la compilación de nuestros archivos con el comando *git push heroku master*. Y se nos mostrará la dirección web donde se aloja la página.

```
-----> Build

-----> Caching build
       - node_modules

-----> Pruning devDependencies
       removed 202 packages and audited 213 packages in 3.65s

       2 packages are looking for funding
         run `npm fund` for details

       found 3 vulnerabilities (1 moderate, 1 high, 1 critical)
         run `npm audit fix` to fix them, or `npm audit` for details

-----> Build succeeded!
-----> Discovering process types
       Procfile declares types     -> (none)
       Default types for buildpack -> web
-----> Compressing...
       Done: 61.2M
-----> Launching...
       Released v23
       https://webvr-tfg-app.herokuapp.com/ deployed to Heroku
```

A.3. Configuración inicial de *Oculus Go*

1. Descarga y abre la aplicación *Oculus* en el teléfono personal.

2. Sigue las instrucciones que aparecen en la pantalla para conectar el teléfono a las gafas de realidad virtual, y posteriormente a la red *Wi-Fi*.

3. Colócate el visor, aflojando las correas laterales, y ajustándolas correctamente a tu cabeza.

4. Presiona el botón de encendido del visor, situado en el borde frontal del mismo.

5. Configura el controlador *Oculus*, debes insertar una pila, a continuación mantén presionados los botones ***Oculus*** y **Atrás** hasta que el *LED* del controlador parpadee y se quede encendido.

6. Sigue las instrucciones presentadas en el visor para completar la configuración del mismo.

En último lugar, comentar sobre la configuración de *Google CardBoard* que su instalación es simplemente introducir el *smartphone* en el visor, una vez incorporado acceder a nuestra aplicación y presionar el botón de *A-Frame* para visualizar la realidad virtual. Finalmente ajustamos el visor mediante las tiras laterales a nuestra cabeza.

Apéndice B
Manual de Usuario

B.1. Navegación web

En este apéndice se describirá el uso de la aplicación web desde el punto de vista del usuario detalladamente, mediante la descripción exhaustiva de la composición de cada página.

B.1.1. Ingreso

En primer lugar, encontraremos en nuestra web la vista de Ingreso. Esta página se compone de un formulario que a su vez se subdivide en usuario y contraseña como campos a rellenar, y un botón para confirmar la información.

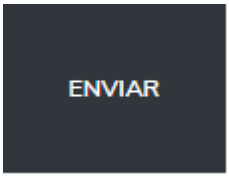


Figura 50: Botón de ingreso

Si el usuario rellena este formulario de forma correcta, este botón le enviará a la pantalla *Home* de la aplicación. En la parte inferior del formulario se encuentra un enlace clicable que nos enviará a la zona de registro. En el caso de completar el formulario de forma incorrecta, la página mostrará un error en relación a la causa.

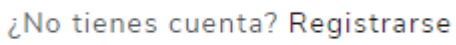


Figura 51: Enlace al registro

B.1.2. Registro

Esta página muestra una información similar a la vista de Ingreso. Se compone de un formulario con usuario y contraseña, y un botón que, en caso de un registro válido, enviará al

usuario a la pantalla *Home*.

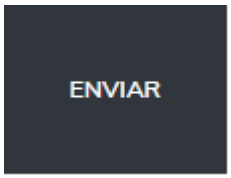


Figura 52: Botón de registro

Además en la parte inferior se muestra un enlace a la vista de Ingreso en caso de que ya poseamos una cuenta registrada. En el caso de completar el formulario de forma incorrecta, la página mostrará un error en relación a la causa.

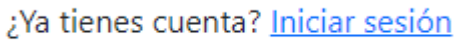


Figura 53: Enlace a la vista de ingreso

B.1.3. *Home*

Esta pantalla se conforma de un formulario en el cual el usuario introducirá el código del modelo deseado. Podremos acceder a la visualización del modelo mediante el botón.

Figura 54: Botón para visualizar modelo

También está compuesta por un botón en la parte superior derecha, cuyo objetivo es finalizar nuestra sesión y devolverlos a la vista de ingreso.

En el caso de un usuario administrador, además, se mostrará un formulario adicional con un campo código y un selector de archivos. Este código estará unido al modelo elegido y se almacenarán en la base de datos al pulsar el botón que conforma este formulario. En el caso de completar el formulario de forma incorrecta, la página mostrará un error en relación a la causa.

Figura 55: Botón para añadir modelo

B.1.4. Visualización de modelo

La única función de esta pantalla es mostrar el modelo deseado, en esta el usuario deberá hacer clic en el icono de *A-Frame* para comenzar el modo realidad virtual mediante el uso de hardware VR.

Figura 56: Botón de *A-Frame*

Referencias

1. B. J. Harris, *The history of the future: Oculus, Facebook and the revolution that swept virtual reality* (Dey St., 2020).
2. (https://threejsfundamentals.org/threejs/lessons/threejs-fundamentals.html).
3. Mongoose. *African American Studies Center* (2005).
4. A. Mardan, Starting with Express.js. *Pro Express.js*, 3-14 (2014).
5. *Documentation*, (http://www.passportjs.org/docs/).
6. (https://ejs.co/).
7. *¿Qué es la WebVR? (Y por qué deberías prestarle atención)*, jun. de 2020, (https://javiersalinas.es/que-es-la-webvr/#1%5C_Definicion%5C_de%5C_WebVR).
8. P. p. D. Blanchard, *Webvr ¿que es y como funciona?*, mayo de 2020, (https://blanchardspace.wordpress.com/2017/07/31/webvr-que-es-y-como-funciona/).
9. O. Laguno y Samuel, *Las MEJORES GAFAS VR del 2021 - Análisis y comparativa*, abr. de 2021, (https://www.destreaming.es/vr/mejores-gafas-vr/).
10. J. C. López, *Oculus Go, análisis: esto es lo que nos ofrecen las gafas autónomas que aspiran a impulsar la realidad virtual por 219 euros*, sep. de 2018, (https://www.xataka.com/analisis/oculus-go-analisis-caracteristicas-precio-especificaciones).
11. *Frame – Make WebVR*, (https://aframe.io/).
12. R. Baruah, *AR and VR Using the WebXR API Learn to Create Immersive Content with WebGL, Three.js, and A-Frame* (Apress, 2021).

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA