UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN INGENIERÍA DE SOFTWARE

Aplicación con realidad aumentada para el diseño de interiores

Augmented reality app for interior design

Realizado por
Miguel Peláez Medina

Tutorizado por
Rafael Marcos Luque Baena

Departamento
Departamento Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2021

Resumen

En este trabajo de fin de grado creamos una aplicación de realidad aumentada centrada en el diseño de interiores, donde el usuario podrá escoger entre una selección de muebles para colocar su modelo en tres dimensiones en la realidad aumentada. Con el objeto de poder ofrecer el inventario de una tienda de diseño de interiores desde cualquier lugar a través de un dispositivo móvil.

La arquitectura de nuestro proyecto se basa en el diseño de *frontend* y *backend*, siendo el *frontend* nuestra aplicación de realidad aumentada creada con el motor gráfico Unity, y que se comunica con el *backend*, capa de acceso a datos que está definida cómo un servidor en NodeJs, el cual gestiona una base de datos no relacional de MongoDB. Igualmente, definimos una aplicación web de administración creada con el *framework* web Angular, para manejar los datos sensibles del sistema.

Palabras clave:

Realidad aumentada, aplicación móvil, *frontend*, *backend*, aplicación web.

Abstract

In this project we create an augmented reality application focused on interior design, on which the user can choose among a selection of furniture to place its three dimensional model into the augmented reality. In order to offer an interior design shop inventory from any place with a mobile device.

Our project architecture is based on a frontend and backend design, the frontend being the augmented reality application created with the graphic engine Unity, which establishes communications with the backend, access layer defined as a NodeJS server, who manages a non-relational mongoDB database. Furthermore, we define a web administrator application to manage our system sensitive data made with the web framework Angular.

Keywords:
Augmented reality, mobile application, *frontend*, *backend*, web application.

Índice

Índice de figuras

1

Introducción

Con el aumento del comercio a través de la red, existe una necesidad de mostrar los productos de una tienda web de forma que sea atractiva y dinámica para poder captar posibles clientes. Y con las últimas innovaciones tecnológicas en materia de realidad aumentada se presentan posibilidades nunca vistas.

La realidad aumentada es la tecnología que nos permite incluir elementos virtuales que se superponen a la realidad física, utilizando algún dispositivo como ventana a este nuevo mundo. Aunque este tipo de tecnologías comienza a tener fuerza ahora, su origen se remonta a principios del siglo pasado con la novela *'The master key'* de L.Frank Baum, en la que el protagonista consigue unas gafas con unas lentes que marcan a las personas en la frente con indicaciones sobre la personalidad del individuo (Norman, 2021). Con el avance tecnológico en el campo de la informática, hemos podido ver cómo está tecnología avanzaba en pequeños pasos en la segunda mitad del siglo veinte, y revolucionándose continuamente en el siglo veintiuno. Actualmente contamos con muchas herramientas y soportes para crear aplicaciones y experiencias de realidad aumentada.

Este TFG se centra en el diseño de interiores, con una aplicación para que los usuarios puedan observar, a través de un dispositivo móvil, los diferentes muebles de una tienda, utilizando la realidad aumentada para proporcionar una visión fidedigna del producto final. El usuario podría también acordar con la tienda una selección de muebles creando una composición específica que podría descargarse en su teléfono en vez de colocar mueble a mueble.

Motivado por los últimos avances en realidad aumentada y en tecnologías web pretendemos crear una aplicación moderna y con una arquitectura adaptada a los nuevos tiempos. Además, debido a la pandemia mundial del COVID-19, con sus

consecuentes restricciones de movilidad, sería conveniente ofrecer la posibilidad al usuario de poder gestionar una compra desde dónde quiera. Si, por ejemplo, un cliente quisiera reformar una habitación entera sería capaz de hacerlo virtualmente con la aplicación, colocando cada mueble en su respectivo lugar, viendo si las medidas son adecuadas. Todo esto desde la propia habitación ahorrando tiempo y dinero al usuario.

La aplicación de realidad aumentada debe establecer una comunicación con una base de datos de forma remota, para obtener toda la información necesaria de la aplicación a través de un *backend*. Aparte, para crear elementos en la base de datos y gestionarlos debemos crear una aplicación de administración donde usuarios autorizados pueden acceder a la información sensible de nuestro sistema.

Nuestro objetivo principal es crear 4 sistemas o aplicaciones que se comunican entre sí formando una unidad:

1. Una aplicación de realidad aumentada para dispositivos móviles dónde el usuario puede seleccionar diferentes muebles y colocarlos en el espacio virtual proporcionado por la realidad aumentada. Esta aplicación precisa ciertos requisitos:
    - Usar un *framework* de realidad aumentada, el cual nos permita interactuar a través de la cámara con el entorno, detectando las superficies con cierta precisión.
    - Tener una interfaz de usuario fácil de usar que permita al usuario navegar de forma intuitiva por la app, ya que la función principal y la carga de trabajo se la lleva la realidad aumentada no queremos sobrecargar la aplicación de funciones.
    - La aplicación debe conectarse con una API, tiene que poder hacer las peticiones de manera adecuada y responder ante los errores y los resultados de estas peticiones.
2. Una API que se comunique con la aplicación de realidad aumentada a través de peticiones HTTP, la cual proporcione la lista de muebles con información relevante para el usuario (descripción, foto, detalles, etc.), además del modelo 3D del objeto. El objetivo es ofrecer un inventario actualizado de muebles sin la necesidad de actualizar la aplicación del dispositivo móvil de forma constante. Para crear la API se tendrá que definir las peticiones GET, POST, PUT o DELETE y sus rutas de acceso.
3. La API deberá conectarse a una base de datos no relacional que almacene todos los datos de la aplicación como muebles, usuarios y archivos, para poder enviar los datos.
4. Por último, una aplicación de administración dónde usuarios autorizados puedan editar los elementos de la base de datos que requieran

un nivel de acceso más restringido debido a la confidencialidad de estos datos.

2

Estado del arte

La realidad aumentada es el conjunto de técnicas que permiten la aplicación de elementos virtuales sobre una representación con la realidad física (Oxford english dictionary, 2021).

El término realidad aumentada fue usado por primera vez por Thomas P.Caudell en 1992, cuando desarrolló un prototipo de unas gafas que ayudaban en las tareas de revisión del Boeing 747, dado que cada vez los aviones tenían piezas más pequeñas, por lo tanto, los revisores tenían perdían mucho tiempo mirando las instrucciones, y que estos procesos de revisión eran muy difíciles de automatizar (P.Caudell, 1992).

Ya en la década de los 60, apareció el primer visor de realidad aumentada/realidad virtual, en aquel momento eran el mismo concepto. La espada de Damocles, creada por Ivan Sutherland y David Cohen en 1968. Gracias a un sistema de espejos ayudado por dos tubos de rayos catódicos, el dispositivo se conectaba a un ordenador, que enviaba las imágenes por el sistema de espejos reflejando las imágenes. Para conocer la posición y la rotación del usuario, el dispositivo estaba sujeto a un brazo mecánico en el techo (Servin, 2021).

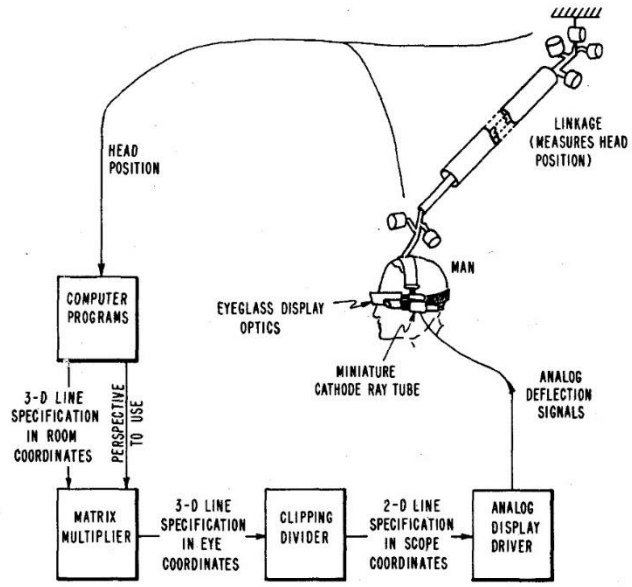


Figura 2.1 Esquema de la espada de Damocles

En los años 70, en la Universidad de Connecticut, Myron Krueger creó el Videoplace, un laboratorio de realidad aumentada dónde los usuarios podían interactuar con un entorno de realidad aumentada, sin necesidad de tener dispositivos atados al cuerpo del usuario. Los usuarios se veían reflejados en una pantalla en la habitación y podían interactuar con elementos virtuales que aparecían en ella (Krueger, 2021).

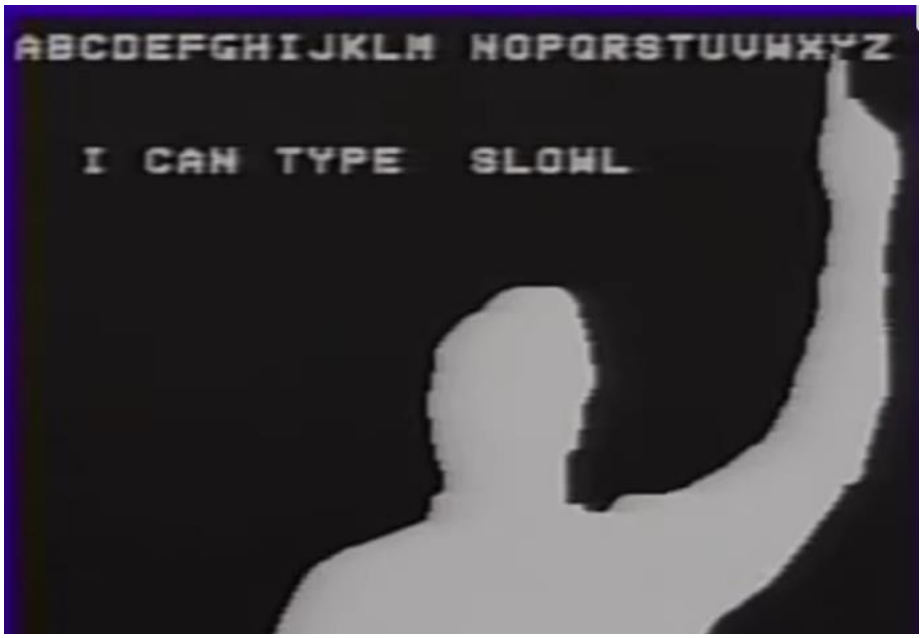


Figura 2.2 Un teclado en Videoplace, se puede escribir moviendo el dedo hacía la letra.

A comienzos del siglo XI, podemos encontrar varios hitos de la realidad aumentada:

Bruce H. Thomas y Wayne Piekarski, llevaron el videojuego Quake a la realidad aumentada con un dispositivo que se componía de un visor, un ordenador a la espalda y un mando con forma de pistola, cómo se muestra en la figura 2.3, creando la primera experiencia de videojuego que se podía jugar al aire libre (H.Thomas, 2002).

Figura 2.3 Dispositivo de ARQuake.

Figura 2.4 Ejemplo de imagen que veía el usuario en el visor.

En el año 2000, la primera biblioteca de realidad aumentada de código abierto ARToolkit, es publicada por la Universidad de Washington, desarrollada originalmente por Hirokazu Kato en el instituto de ciencia y tecnología de Nara (Billinghurst, 1999).

Más tarde, en 2009, ARToolkit lanzaría su versión compatible para navegadores web.

En 2012, Google presentó Google Glass, un dispositivo con forma de gafas ,como se muestra en la figura 2.5, que disponía en el lateral de un trozo de cristal que servía para proyectar imágenes en la retina. Se dejaron de producir en 2015 debido a una mala recepción en sus ventas.

Figura 2.5 Dispositivo Google Glass

Actualmente, impulsado por el auge de los dispositivos móviles, existen diversas herramientas para desarrollar aplicaciones de realidad aumentada:

ARCore

Figura 2.6 Logo de ARCore

ARCore es la plataforma de Google para crear aplicaciones de realidad aumentada en dispositivos Android usando diversas APIs, se fundamenta en 3 pilares (Google, 2021):

- Estimación de la luminosidad de la imagen obtenida por la cámara para, por ejemplo, sombrear un elemento que se encuentre en una zona oscura de la instancia.
- Control del movimiento del dispositivo para comprender su posición en el espacio.
- Compresión del entorno detectando puntos en las superficies de las imágenes, creando planos que la aplicaciones pueden usar para colocar objetos descansando en dichas superficies.

ARKit

Figura 2.7 Logo de ARKit

En 2017 Apple lanzó ARKit, su *framework* para realidad aumentada para dispositivos IOS. Recientemente se lanzó su cuarta versión, además Apple ha creado varias herramientas para desarrollar aplicaciones en realidad aumentada que se complementa junto a ARKit, como RealityKit o ARCreationTools (Apple, 2021).

Vuforia

Figura 2.8 Logo de Vuforia

Vuforia es un SDK que nos permite crear aplicaciones de AR en diferentes lenguajes cómo C++, C# o Java. Además ofrece los VuMark, marcadores personalizables que la SDK puede reconocer en tiempo de ejecución (Vuforia, 2021).

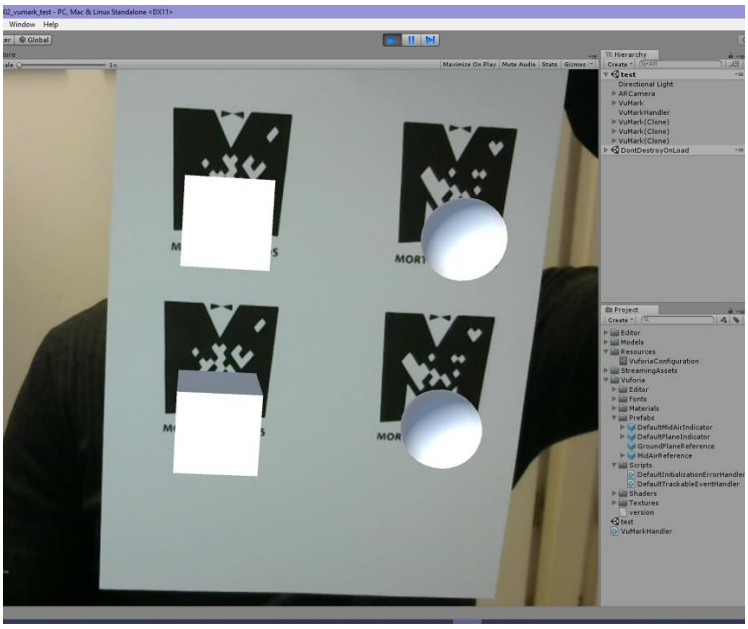


Figura 2.9 Ejemplo de elementos siendo colocados en los VuMark detectados (innerdrive studios, 2018).

Hololens

Figura 2.10 Persona trabajando con Hololens.

Microsoft dio su paso en la realidad aumentada con el dispositivo sin ataduras Hololens. Recientemente lanzó al mercado su segunda iteración, Hololens 2 con un enfoque profesional, ya que sólo se venden a empresas y no a desarrolladores (Rus, 2019). Además permite integrar servicios de Azure.

Magic Leap

Figura 2.11 Dispositivo Magic Leap

Bastante parecido a Hololens, un dispositivo ligero con lentes que se adhiere a la cabeza. Tras un batacazo en sus ventas iniciales, decidieron centrarse en venderse cómo solución a empresas, además la compañía se prepara para lanzar su segunda versión mejorada con acceso a tecnología 5G (Pérez, 2019).

Unity

Figura 2.12 Logo de Unity

Para nuestra aplicación, hemos usado Unity que es un motor gráfico multiuso: 3D, 2D, VR y AR. Para multiplataformas, usa el *framework* ARFoundation que sirve como capa de abstracción sobre los *framework* ARKit y ARCore para que se puedan crear aplicaciones móviles que puedan usarse en Android y en IOS. Además permite desarrollar para Hololens y Magic Leap.

Disponible en el mercado desde 2005, fue la creación de 3 daneses: Joachim Ante, Nicholas Francis y David Helgason, con el objetivo de acercar el desarrollo de videojuegos a más personas (GlobeNewswire, 2011). Según los datos de 2018, Unity supone el motor elegido por los desarrolladores un 60% de las veces al desarrollar aplicaciones de realidad aumentada o realidad virtual. (cbinsights, 2018)

Además fue el motor gráfico dónde se desarrolló el exitoso juego de realidad aumentada, Pokemon Go que cuenta con más de un billón de descargas y 166 millones de usuarios (Iqbal, 2021).

En Unity se puede usar Vuforia ya que tiene integración, pero descartamos esta posibilidad por el alto coste de su licencia y además, no pudimos crear una aplicación funcional con su versión de prueba.

Una alternativa al motor gráfico Unity podría ser Unreal Engine que es otro motor gráfico como Unity, y ofrece un *framework* parecido a ARFoundation para la realidad aumentada, pero descartamos esta posibilidad por problemas técnicos que impedían crear una aplicación con éxito.

3

Desarrollo

Metodología de trabajo

La metodología que se ha escogido para realizar este proyecto es el desarrollo en espiral. Esta se basa en 4 fases secuenciales que se van iterando hasta obtener el producto deseado:

- Determinar objetivos, en esta primera fase debemos establecer cuáles son los objetivos, requisitos y restricciones que debe cumplir el producto en esta iteración.

- Identificación de riesgos, a continuación se realizará un análisis sobre los riesgos y problemas que podemos encontrar en el desarrollo de los objetivos detallados en la fase anterior.

- Desarrollo y pruebas, esta fase abarca el desarrollo de los objetivos y requisitos establecidos en la primera fase, y las pruebas para asegurar que las complicaciones y riesgos establecidos en la segunda fase han sido solventados y tomados en cuenta.

- Planificación, al final de cada iteración debemos obtener una versión del producto más refinada que la anterior. En esta fase debemos examinar cual es la dirección que debemos tomar en la siguiente iteración, un bosquejo sobre el futuro del proyecto.

El objeto de este desarrollo es obtener una versión del producto en cada iteración que no sólo nos permita ver un esbozo del producto final, sino también ver cuáles

son los problemas y los riesgos de las decisiones que hemos tomado durante el desarrollo.

La decisión de usar esta metodología de trabajo recae en el hecho de que es un trabajo individual, por ejemplo, las metodologías ágiles cómo scrum se centran en el trabajo y la interacción de equipos.

Entre otras metodologías de trabajo que se puedan aplicar a proyectos individuales podríamos encontrar el desarrollo en cascada, similar al desarrollo en espiral pero las fases son cerradas, es decir, un vez se completa una fase no se vuelve a iterar. Descartamos esta metodología debido a su poca flexibilidad ya que un error en las primeras fases supondría empezar de nuevo por lo que el coste de una equivocación supone un coste mucho mayor que en la metodología en espiral.

Esquema de desarrollo

Nuestro objetivo principal es crear un sistema que se compone de 3 elementos:

- *Backend*, un servidor creado en Nodejs, cuyo objetivo es servir de capa intermedia entre la base de datos y la aplicaciones de administrador y usuario. Debe cumplir los siguientes requisitos:
    - Disponer de una base de datos que contenga información de usuarios, muebles y archivos como imágenes o los modelos en tres dimensiones de los muebles.
    - Conectar y tratar con los datos de la base de datos previamente descrita alojada en el mismo servidor.
    - Proporcionar una API web con la que poder conectarse a través de peticiones HTTP.
    - Diferenciar entre usuarios normales y usuarios administrador, y bloquear o permitir el acceso a ciertos datos según la sensibilidad de los mismos.
- Aplicación de usuario, es la aplicación de realidad aumentada para dispositivos móviles creada en Unity. Debe cumplir con los siguientes requisitos:
    - Conectar con la API web de nuestro *backend* a través de las peticiones HTTP.
    - Utilizar un *framework* de realidad aumentada para poder disponer los muebles que obtenemos del *backend.*
    - Renderizar los archivos 3D de los muebles obtenidos por el *backend* en tiempo de ejecución.
- Aplicación de Administrador, se encarga de tratar con datos sensibles de la aplicación, cómo gestionar muebles y/o usuarios, por lo tanto debe

comunicarse con el *Backend* de forma segura. Debe tener los siguientes requisitos:

- Conectar con la API web de nuestro *backend* a través de las peticiones HTTP.
- Ser capaz de acceder a los datos sensibles de la aplicación.

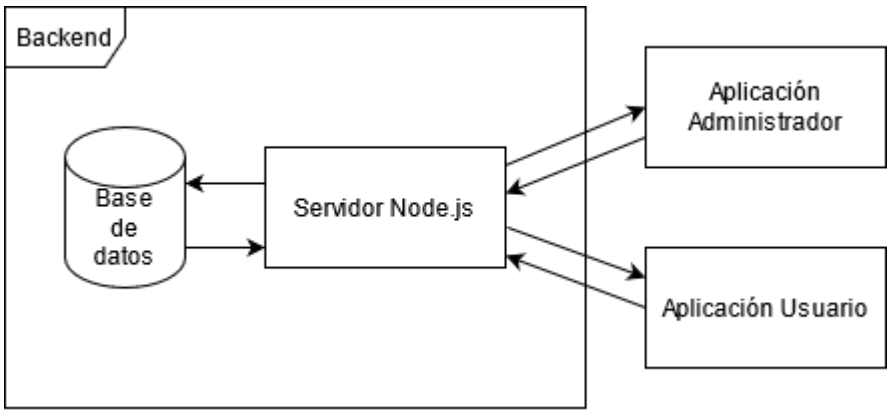


Figura 3.1 Esquema del proyecto.

Casos de uso

Según los requisitos identificados en el capítulo anterior podemos destacar dos diagramas de caso de uso, uno centrado en el usuario de la aplicación de realidad aumentada (figura 3.2) y otro en el usuario de la aplicación de administración (figura 3.3).

En el caso del usuario de la aplicación de realidad aumentada, el usuario deberá poder iniciar sesión, registrarse y poder cerrar sesión. Además la función principal es colocar muebles en la realidad aumentada y poder interactuar con ellos, por lo que se le debe proporcionar muebles que pueda seleccionar, el mueble tiene que tener un modelo 3D que se obtiene del *backend*, y al obtenerlo, renderizarlo.

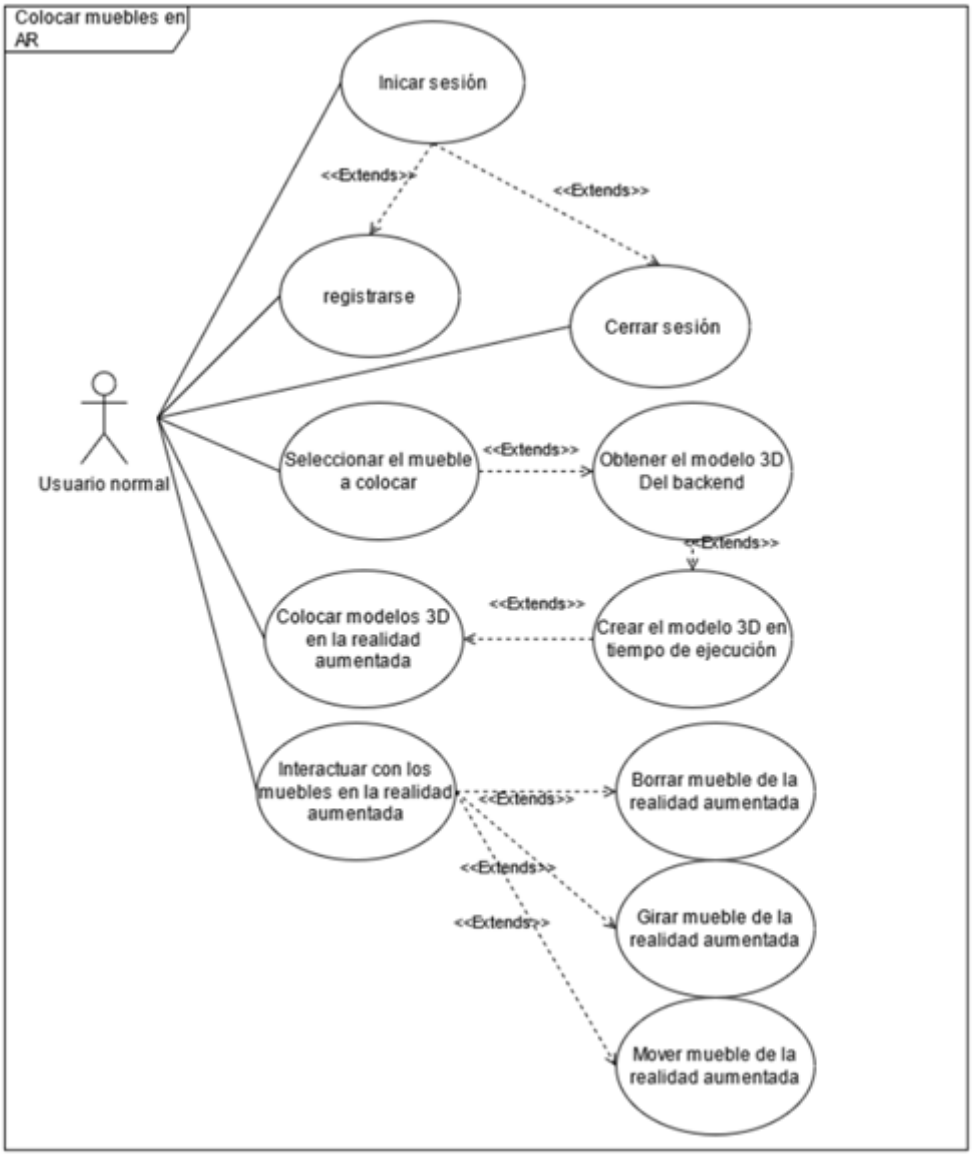


Figura 3.2 caso de uso de un usuario de la aplicación de realidad aumentada.

Para los usuarios administradores, deberán poder registrarse para demostrar que son usuarios que pueden acceder a información confidencial, a partir de que se inicie sesión, se debe disponer la visualización de los muebles de nuestra base de datos y de los usuarios, y realizar acciones sobre ellos: borrado, modificación y creación.

Figura 3.3 caso de uso de un usuario de la aplicación de administración.

3.1

Desarrollo del Backend

Entendemos como *backend* a la capa de la aplicación que se centra en la gestión de datos, y que comunica con las capas superiores que utilizan los usuarios.

El objetivo de esta división por capas es el de separar la funcionalidad de la aplicación para simplificar y sistematizar el desarrollo. También tiene una funcionalidad de seguridad, hay operaciones que deben de ser invisibles al usuario para evitar posibles malas intenciones con la gestión de datos personales y/o datos sensibles de la aplicación.

Descripción de la base de datos

Para la base de datos hemos usado MongoDB.

Figura 3.1.1 Logo de MongoDB

MongoDb es un sistema de datos de código abierto y NoSQL, esto quiere decir que la información se guarda en formato BSON (Binary JSON) en contraste con los sistemas SQLs que utilizan tablas (MongoDB Inc., 2021).
Las entidades se organizan en documentos, y en nuestra aplicaciones hemos definido los siguientes documentos:

- *Models*, aquí se encuentra la lista de muebles o conjuntos de muebles, cada uno de ellos contiene:
    - Id: identificador en la base de datos.
    - Nombre.
    - Descripción.
    - Fecha de creación.
    - Visibilidad: existen muebles que son hechos a medida por clientes, estos no se deben mostrar a los demás usuarios por lo que se deben

acceder a través de un código o iniciando sesión con el usuario asociado al correo electrónico descrito en el campo *userEmail* descrito a continuación. Toma dos valores: público o privado.
    - Código: este campo estará vacío si el mueble es público.
    - *userEmail*: Si el usuario de un mueble hecho a medida tiene cuenta registrada en nuestra base de datos, este campo contendrá el correo electrónico de dicho usuario.
- *Users*, los usuarios de la aplicación pueden, si quieren, registrarse. Cada uno de ellos contiene:
    - Id: identificador en la base de datos.
    - Nombre.
    - Email de registro.
    - Contraseña, encriptada por seguridad.
    - Rol, diferencia entre usuarios comunes y administradores de la aplicación que tienen acceso a datos sensibles.
    - Fecha de registro.
- *Fs.files* y *fs.chunks*, estos dos documentos pertenecen a la estructura que sigue GridFS, el cual es una representación para guardar archivos de más de 16 MB que usa MongoDB. *Fs.files* guarda los metadatos del archivo mientras que *fs.chunks* representa el conjunto de trozos en los que se dividen los archivos para su almacenamiento. Aquí es dónde guardamos los archivos de la aplicación: imágenes de muebles y sus respectivos modelos 3D.

Descripción de la API

Nuestro *backend* es, realmente, un servicio web alojado en un servidor con la que las demás aplicaciones se comunican. Este software utiliza una arquitectura de Transferencia de Estado Real (Representational state transfer o REST) que utiliza peticiones HTTP como vía de comunicación para acceder a los datos (Gillis, 2020).

Las comunicaciones están representadas como peticiones GET, POST, PUT, DELETE a rutas o URLs.
Por ejemplo, en este proyecto la petición GET a la dirección *https://localhost:3003/api/furniture/public* nos devuelve una lista de todos los muebles que son públicos. Si diseccionamos esta URL:

- *https//*: representa el protocolo para enviar los datos.
- *localhost*: es el anfitrión, es localhost ya que el servicio web se está ejecutando localmente en nuestra máquina.
- *3003*: es el puerto dónde el servicio web está escuchando a las llamadas

- */api/furniture/*: este fragmento se identifica con las comunicaciones que operan con los muebles
- */public*: hace referencia a los muebles públicos, es decir, que todos los usuarios tienen acceso.

A continuación se dará una descripción detallada de todas las operaciones que el servicio web ofrece, pero primero daremos una exposición de las tecnologías usadas.

Node.js

Figura 3.1.2 Logo de Node.js

Node.js es un entorno de ejecución Javascript centrado en el uso de eventos asíncronos para crear aplicaciones escalables que puedan manejar altas cargas de trabajo. Su filosofía asíncrona contrasta con el modelo concurrente basado en hilos ya que evita el bloqueo de los procesos debido a no usar operaciones de escritura y lectura directamente (Nodejs, 2021).

Además Node.js cuenta con su administrador de paquetes NPM (Node Package Manager) que nos permite, a través de una simple línea de comando, incluir diversas bibliotecas y módulos a nuestro proyecto. Los módulos más relevantes que hemos usado en nuestro proyecto son:

- *Bcrypt*, este módulo nos permite encriptar las contraseñas de usuarios con criptografía tipo *hash* (Bcrypt, 2021).
- *CORS*, activa en nuestra aplicación el intercambio de recursos de origen cruzado (*Cross-Origin Resource Sharing*) para permitir que se soliciten recursos restringidos desde otros dominios.
- *Mongoose*, es una librería ODM (*Object-Document Mapping*) la cual nos permite interactuar con nuestra base de datos de MongoDB en Javascript.
- *JsonWebToken*, es un estándar que nos permite declarar la comunicación entre dos partes a través de un objeto JSON encriptado como se establece en RFC 7519 (Internet Engineering Task Force, 2021). Este módulo lo usaremos para generar los *tokens* que se utilizan en las comunicaciones de nuestro *backend* con las demás aplicaciones de nuestro sistema.

Por último, usamos el módulo *expressjs*, debido a su peso en este proyecto le hemos dedicado un subcapítulo.

Express

Figura 3.1.3 Logo de expressjs

Express es una infraestructura para Node.js que nos permite crear APIs web de forma sencilla y flexible.

Por cada documento de la base de datos creamos un controlador: *Models*, *Users*, *Storage* (para los archivos) y uno más llamado *Auth* que sirve para iniciar sesión en la aplicación.

Auth

Este controlador sólo contiene una operación POST en la ruta */api/auth/* en la que, dado un email y una contraseña en el cuerpo de la petición, genera un *token* que contiene la información del usuario y que sirve para acceder a los datos de la aplicación.

Este *token* se introduce en la cabecera '*Authorization*' de las demás peticiones a la API, utilizando el esquema estándar *Bearer* definido en RFC6750, esto quiere decir que el contenido que escribimos en la cabecera es 'Bearer [*token*]' (Internet Engineering Task Force, 2012). Nuestro sistema recibe este *token*, comprueba que es correcto y realiza operaciones de seguridad si es necesario. Por ejemplo, hay datos a los que sólo puede acceder un usuario con el rol de administrador, si un usuario normal intentara acceder a estos datos, nuestro sistema respondería a dicho usuario con un código de error.

Models

La ruta de entrada para los modelos es */api/furniture/* y las diferentes operaciones HTTP son:

- GET */api/furniture/public*: obtiene los modelos públicos que todos los usuarios tienen acceso.

- GET */api/furniture/private*: obtiene los muebles hechos a medida de un usuario.
- GET */api/furniture/code/{código}*: obtiene un elemento privado si su código coincide con el especificado en el parámetro de la ruta.
- GET */api/furniture/{id}:* obtiene un elemento cuyo id coincida con el especificado con el parámetro.
- POST */api/furniture/*: crea un mueble nuevo en la base de datos, los datos van el cuerpo de la llamada.
- PUT */api/furniture/*: actualiza los datos de un mueble en la base de datos, si existe.
- DELETE */api/furniture/{id}*: Elimina de la base de datos un mueble con el identificador pasado por parámetro.

Users

La ruta de entrada para las operaciones de usuario es */api/user/*

- GET */api/user/*: obtiene la lista de usuarios (no se devuelve la contraseña).
- GET */api/user/{id}*: obtiene un usuario con el id especificado en el parámetro.
- POST */api/user/*: crea en la base de datos un usuario.
- PUT */api/user/*: actualiza en la base de datos los datos de un usuario.
- DELETE */api/user/{id}*: elimina de la base de datos un usuario con el identificador pasado por parámetro.

Storage

Este controlador se encarga de la gestión de ficheros y tiene la ruta /api/storage

- GET */api/storage/{filename}*: devuelve un archivo que contenga el nombre pasado por el parámetro '*filename*'.
- POST */api/storage*: almacena en la base de datos un archivo.
- DELETE */api/storage/{filename}*: elimina de la base de datos un archivo que contenga el nombre especificado por el parámetro '*filename*'.

Middleware

Un *middleware* en Express es una función que se ejecuta antes o después del manejo de una ruta de una petición Http. Para nuestro sistema hemos especificado dos:

- *Auth*, este middleware intercepta la petición y comprueba que el *token* que se encuentra en la cabecera *'Authorization'* es válido, con el objetivo de asegurar de que sólo los usuarios que están registrados y han iniciado sesión, es decir, se les ha concedido un *token*, pueden acceder correctamente al recurso. En caso de que el *token* no sea válido, la petición es cancelada y se le devuelve al usuario una respuesta con el código 401 *Unauthorized*

indicando al usuario que no tiene las credenciales válidas para acceder al recurso. Si el *token* no existe en la cabecera devuelve un error 400 genérico.

- *Authorize*, recibe por parámetro una lista de roles que debe tener un usuario para acceder el recurso, en caso de que no tenga el rol adecuado la petición será respondida con el código de error 403 *Forbidden.* Este middleware es el que nos permite diferenciar entre usuarios normales y los administradores que acceden a recursos más sensibles.

Los códigos de estado de respuesta que devuelven las peticiones HTTP de nuestro *backend* siguen el esquema definido en RFC2616 (Internet Engineering Task Force, 1999).

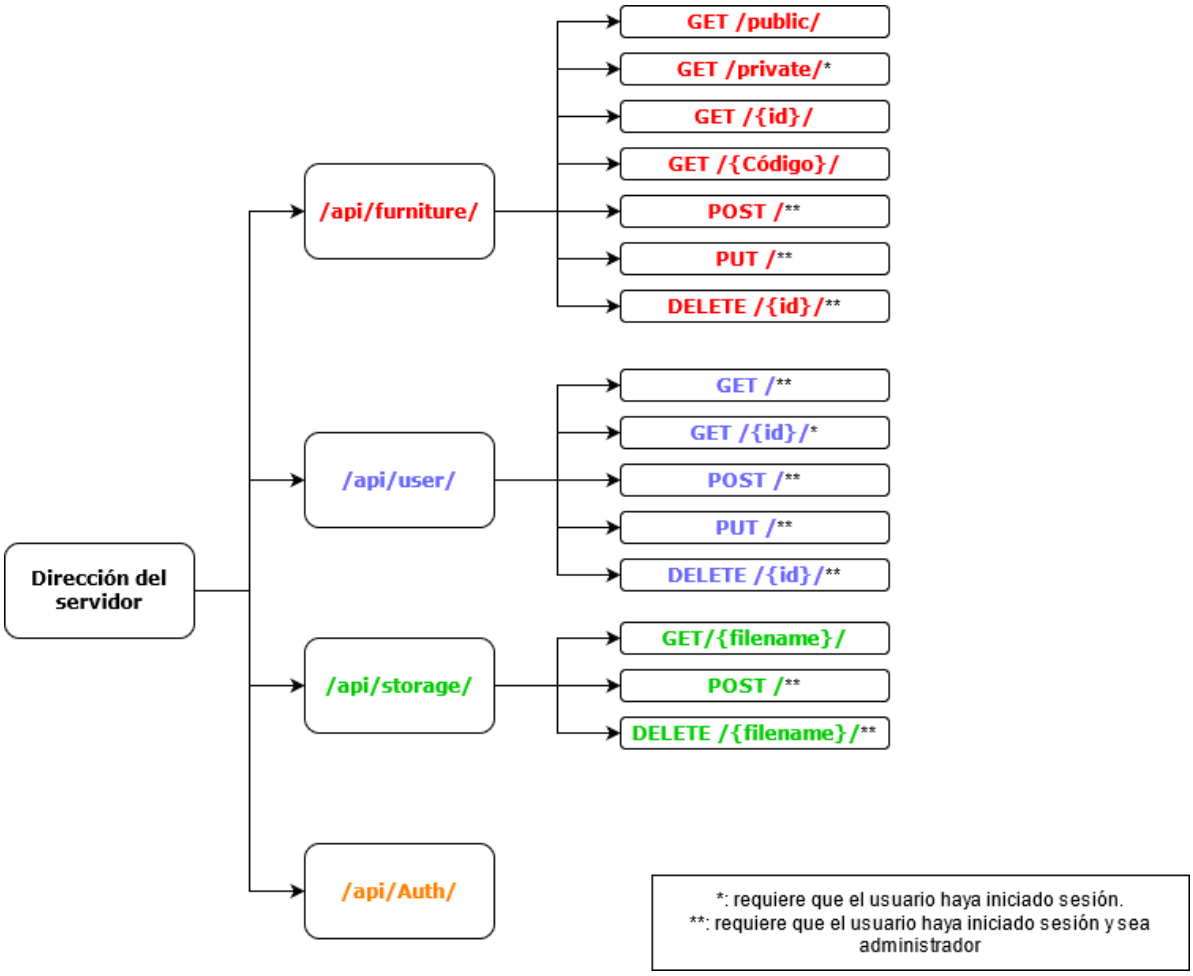


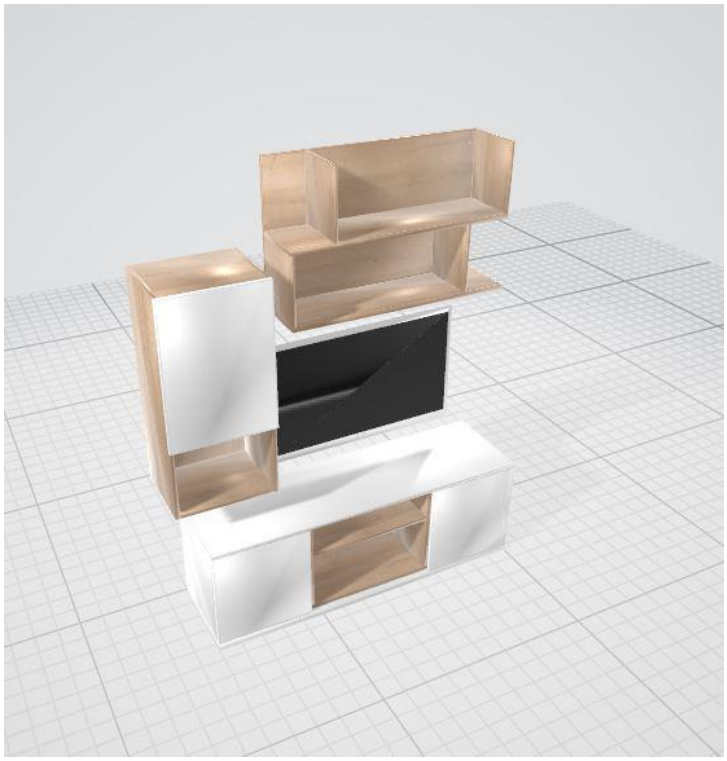
Figura 3.1.4 esquema de la API

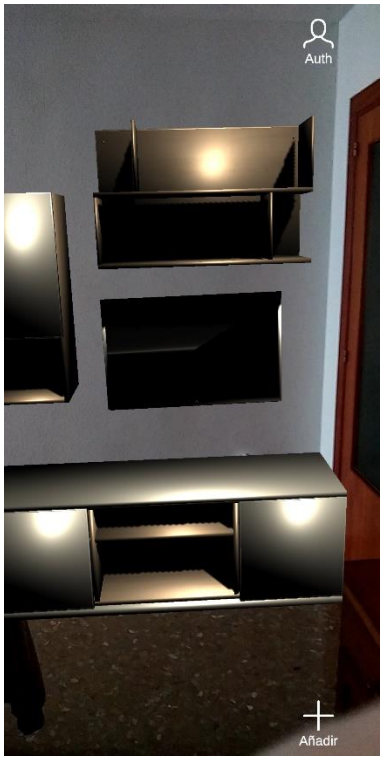
3.2

Desarrollo de la aplicación de Usuario

Nuestra aplicación de usuario se trata de una aplicación móvil de realidad aumentada que permite obtener muebles de nuestro *backend* y disponerlos en el espacio e interactuar con ellos.

En la figura 3.2.1 vemos un modelo 3D de un mueble que se encuentra en nuestra base de datos y en la siguiente imagen 3.2.2 podemos ver una captura de la aplicación de realidad aumentada, en la que se percibe el mismo modelo del mueble emplazado en una habitación. Se puede notar una diferencia en la iluminación de las dos imágenes, ya que al ser diferentes programas en las que se visualiza el modelo, la luz no se renderiza de la misma forma.





Figuras 3.2.1 y 3.2.2 modelo 3D de un mueble de nuestra aplicación y captura de nuestra aplicación de usuario con el mueble colocado en una habitación en la realidad aumentada.

Esta aplicación está implementada con el motor gráfico Unity, escrito en C/C++ pero utiliza .NET y C# para que los desarrolladores interactúen con sus elementos. Utiliza una arquitectura basada en componentes que se disponen en objetos llamados *"GameObject"* que están contenidos en "escenas". Estos componentes pueden ser componentes para simular físicas, elementos de la interfaz gráfica, *scripts* programados por los desarrolladores, etc (Baron, 2019).

Como hemos explicado en la introducción, usamos ARFoundation para implementar la funcionalidad de realidad aumentada. Este *framework* nos proporciona los componentes necesarios para el desarrollo, siendo los más importantes *ARSession* que se encarga del ciclo de vida de toda la experiencia de realidad aumentada y *AR Session Origin* que transforma las coordenadas de nuestro dispositivo en la realidad aumentada a coordenadas globales de Unity. Además, nos ofrece herramientas para depurar como *ARPlaneManager*, la cual nos permite ver que superficies se han detectado a través de la cámara del dispositivo como se muestra en la figura 3.2.3.

Figura 3.2.3 superficie amarilla representa la superficie detectada por el *framework.*

La detección de superficies de ARFoundation se basa en detectar "puntos de característica" (*feature points*), estos representan un punto distinguible en la imagen detectada por la cámara de nuestro dispositivo, estos elementos se agrupan en nubes de puntos que acaban formando las superficies detectadas (Unity, 2021).

Existen inconvenientes en esta forma de detección, una superficie que tenga pocas características destacables no será detectada de forma correcta. En nuestro caso usamos en las imágenes una esquina, la cual una de las paredes es blanca sin adornos y la otra contiene una puerta. La pared blanca apenas hemos conseguido que el programa la detectase, en cambio la puerta al tener diferentes formas y destacar, el *framework* ha sido capaz de detectar superficies como se ve en la imagen 3.2.4.

Figura 3.2.4 superficie de la puerta siendo detectada.

La interfaz de nuestra aplicación se compone principalmente de la emisión de la cámara para ver los muebles en el espacio y de dos botones, uno con el inicio de sesión en la esquina superior derecha y otro con la lista de muebles en la esquina inferior derecha.

Figura 3.2.5 Imagen de la aplicación.

Inicio de la aplicación

Al comienzo de la aplicación se debe obtener los muebles que sean públicos y volcarlos en la lista de muebles, para ello realiza una llamada a nuestra api en la dirección */api/furniture/public/*. Al recibir la respuesta, un JSON con la lista de muebles, nuestra aplicación debe procesarlos y rellenar la lista creando un botón para cada mueble en la lista que aparece en la figura 3.2.7.

Si el usuario está registrado en la aplicación, deberá obtener también los muebles privados de ese usuario, repitiendo el mismo proceso pero realizando una llamada a la dirección */api/furniture/private/*.

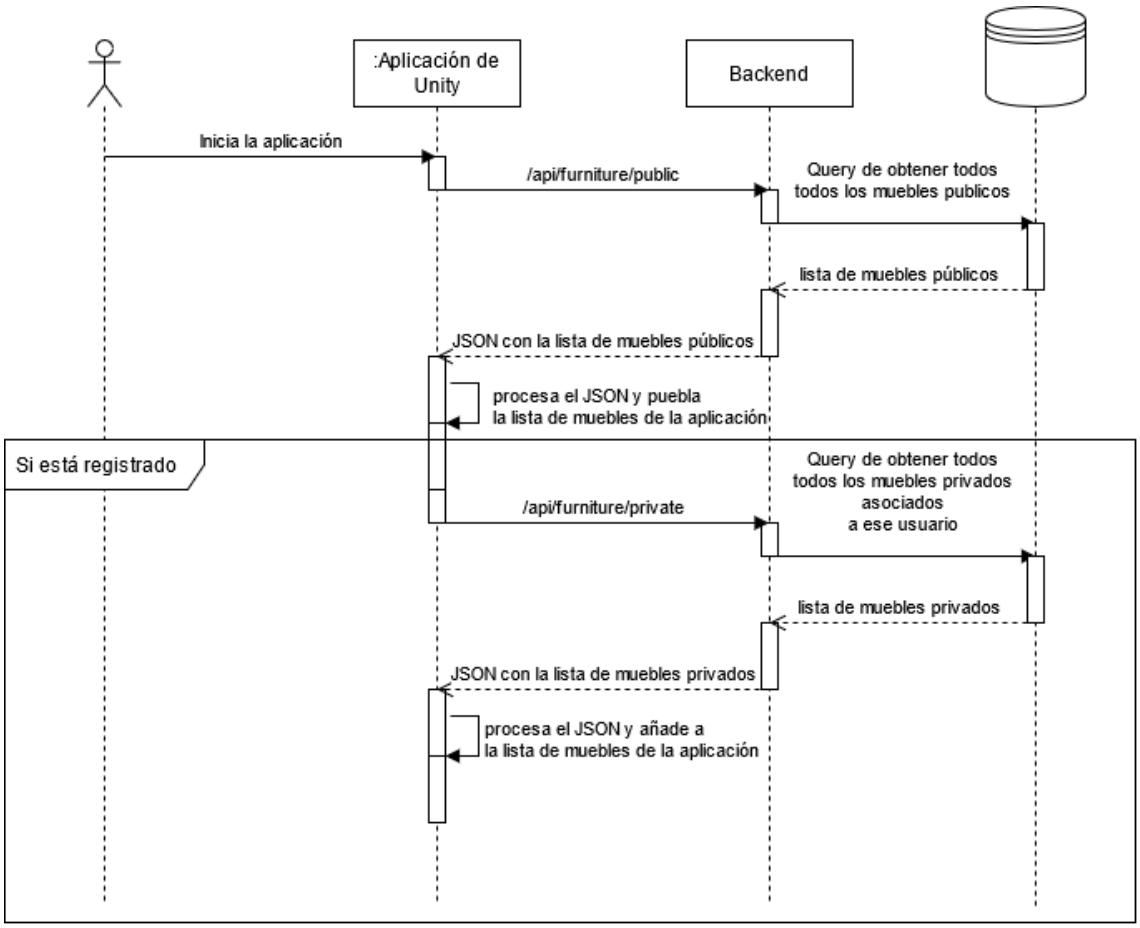


Figura 3.2.6 Diagrama de secuencia del inicio de aplicación.

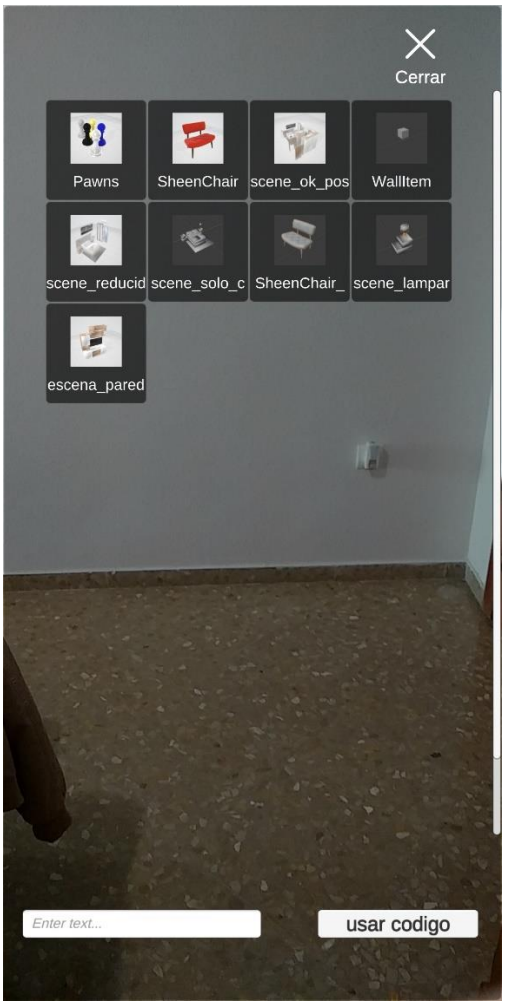


Figura 3.2.7 Lista de muebles con objetos.

Auth

En el botón de *Auth* tenemos las opciones de inicio de sesión y registro de usuario. Primero encontraremos una pantalla con dos entradas de texto, una para el correo y otra para la contraseña cómo se muestra en la figura 3.2.8.

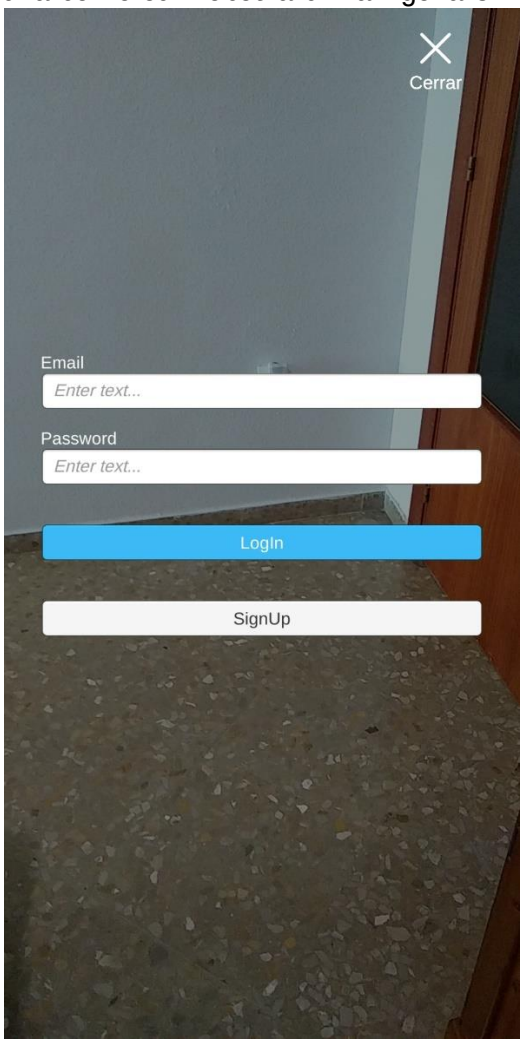


Figura 3.2.8 Pantalla de inicio de sesión.

Si tuviéramos una cuenta, rellenaríamos los campos y pulsaríamos LogIn, al hacerlo la aplicación llama a la ruta */api/auth* de nuestro *backend*. Este proceso está reflejado en el esquema de la figura 3.2.9

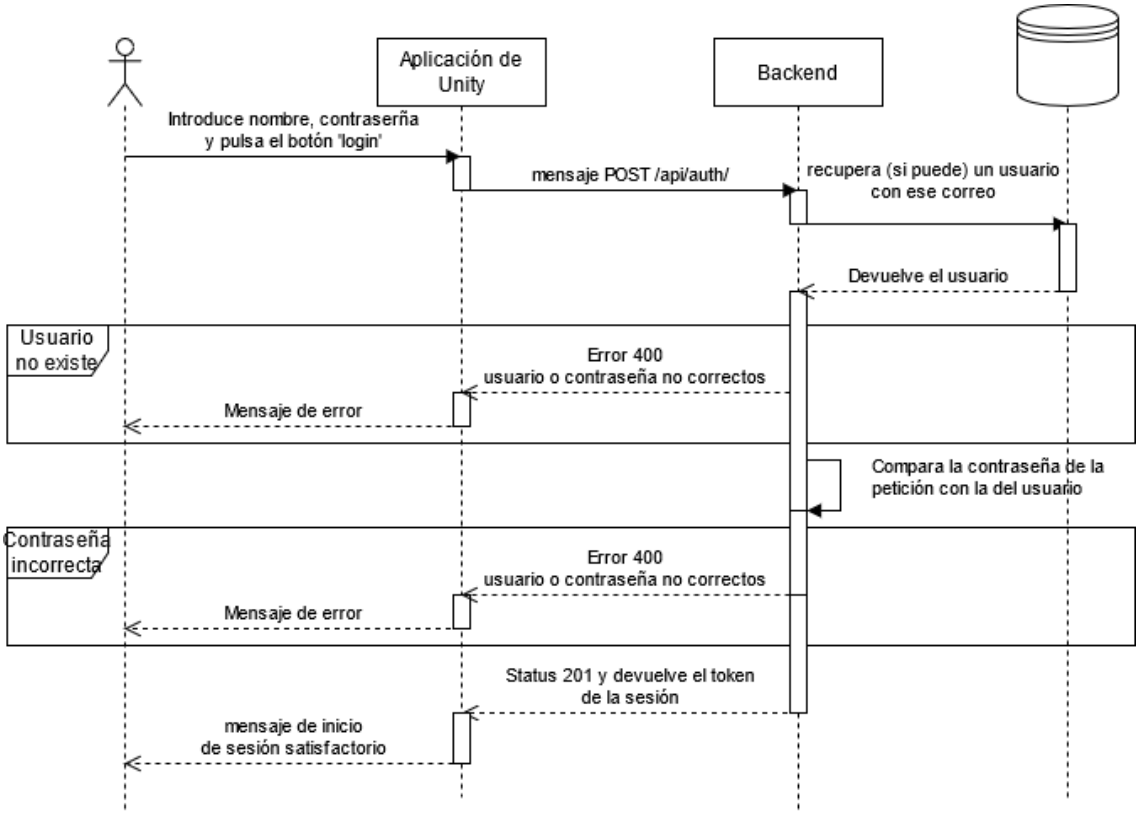


Figura 3.2.9 Diagrama de secuencia de inicio de sesión.

En el caso de que no tuviéramos una cuenta deberíamos crearla, para ello, pulsamos el botón de *SignUp* en la pantalla de inicio de sesión. Entonces, la pantalla cambia a la de crear usuario con un formulario que consta de 3 campos: nombre, correo y contraseña plasmado en la figura 3.2.10.

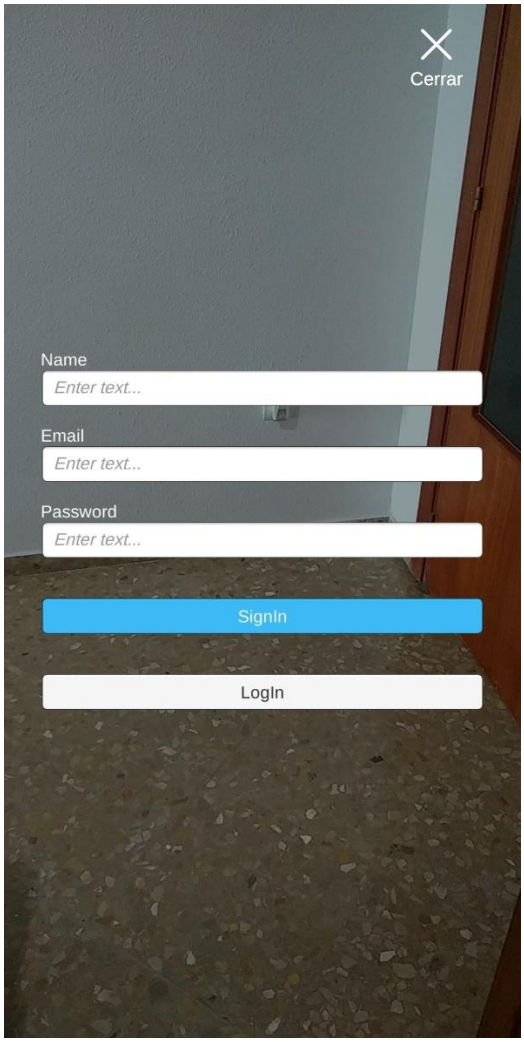


Figura 3.2.10 Pantalla de registro de usuario.

Una vez hayamos iniciado sesión o creado un usuario, se nos mostrará una pantalla con el mensaje de que ya estamos registrados y un botón para salir de la sesión por si queremos ingresar con otro usuario. Tras esta acción la aplicación le pedirá al *backend* la lista de muebles privados del usuario si tuviera alguno.

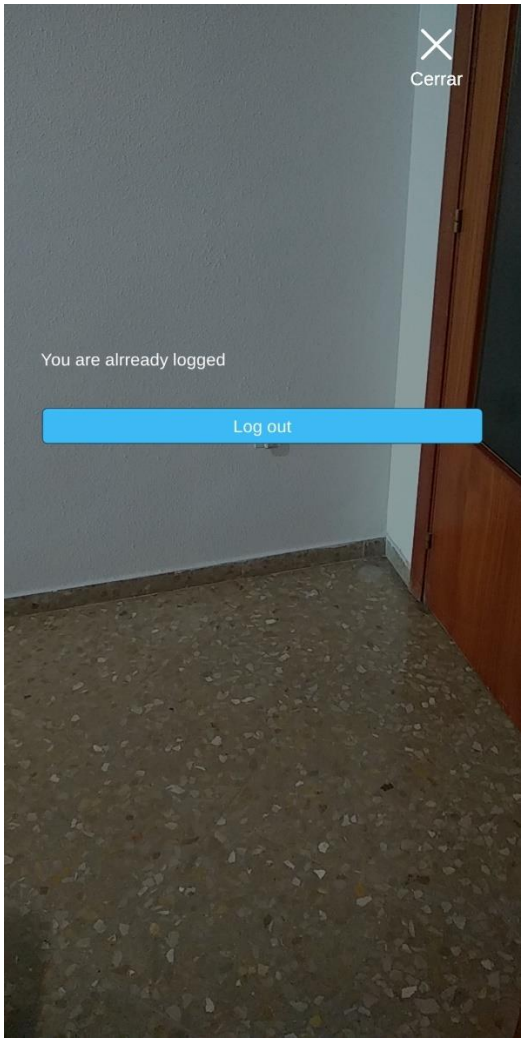


Figura 3.2.11 Pantalla cuando ya se está registrado.

Emplazamiento de muebles en la realidad aumentada

Para colocar muebles en la aplicación debemos pulsar el botón 'añadir', nos desplegará la lista de mueble cómo se muestra en la figura 3.2.12.

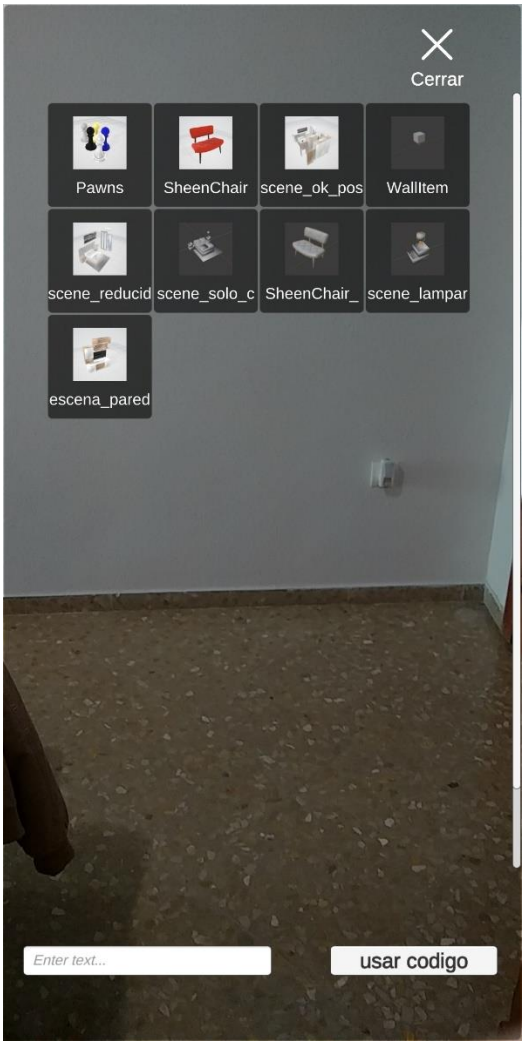
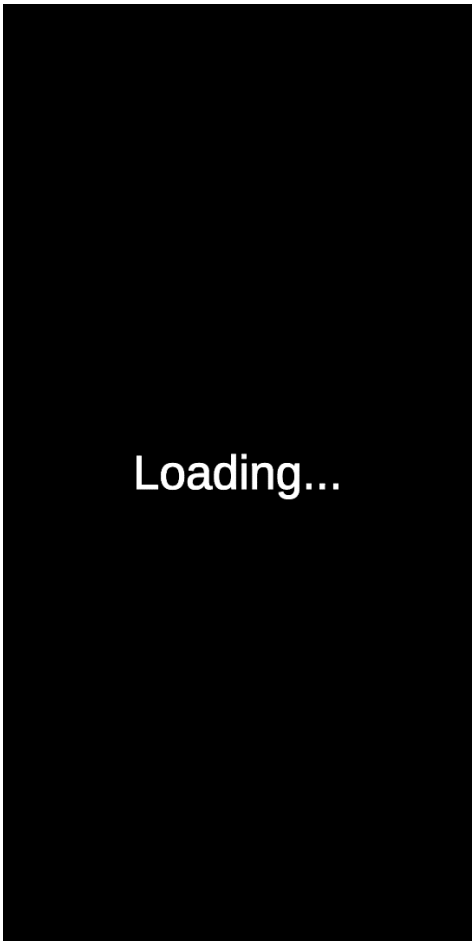


Figura 3.2.12 Lista de muebles.

A continuación pulsamos uno de los botones con muebles o bien, introducimos un código en la entrada de texto abajo y pulsamos el botón contiguo, 'usar código'. La aplicación, entonces, pedirá a nuestro *backend* el archivo con el modelo 3D del mueble, al recibirlo el usuario verá una pantalla de carga (figura 3.2.13) y creará el modelo 3D en el espacio. A continuación, el usuario puede mover la cámara para colocarlo en la posición deseada y pulsar el botón de confirmar de la figura 3.2.14 o cancelar la acción.



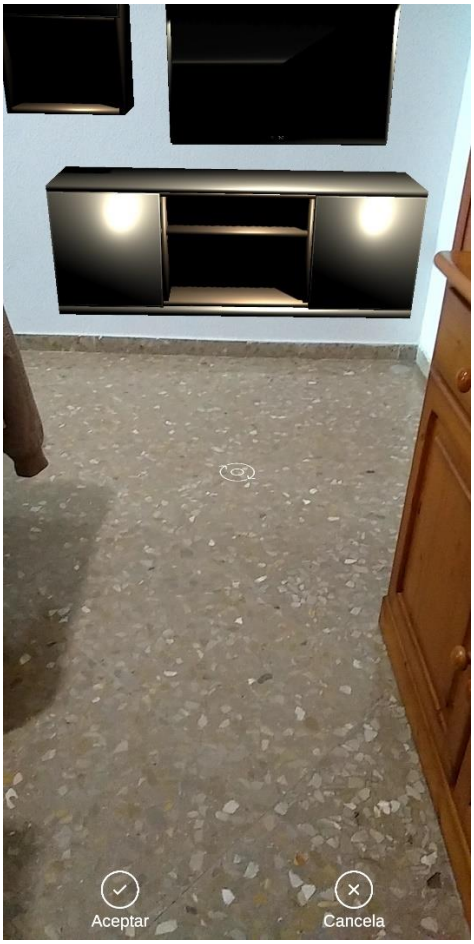


Figura 3.2.13 y 3.2.14 Pantalla de carga y emplazamiento del mueble en la realidad aumentada.

Una vez emplazado el mueble podemos movernos de posición con nuestro dispositivo y los muebles colocados deberían quedarse en el sitio escogido realizando la ilusión de que el mueble está en la habitación, en las figuras 3.2.15 y 3.2.16 observamos el conjunto descargado en diferentes ángulos.

Figuras 3.2.15 y 3.2.16 mueble emplazado desde diferentes ángulos.

Podemos pulsar en la pantalla del dispositivo móvil algún mueble que esté colocado en la realidad aumentada, esta acción provocará que se despliegue un menú con cuatro botones como se puede apreciar en la figura 3.2.17:

- Mover, nos permite mover el mueble por la realidad aumentada, para ello presionamos el dedo sobre la superficie dónde deseemos que el mueble se coloque.
- Rotar, a través de presionar el dedo sobre la pantalla y deslizando a la derecha o a la izquierda el mueble rotará sobre sí mismo.
- Eliminar, elimina el mueble de nuestra aplicación.
- Volver, cierra el menú.

Figura 3.2.17 Menú de edición de un mueble. En este caso hemos seleccionado el mueble de abajo y lo hemos rotado un poco.

Los modelos 3D de los muebles son obtenidos de nuestro *Backend,* a través de la ruta */api/storage* previamente descritos en el capítulo anterior.
Además, tuvimos que tener en cuenta que estos modelos 3D se cargan en tiempo de ejecución, debido a esto, los archivos están en formato GLTF (GL transmision format). GLTF es la extensión de archivo para objetos y escenas 3D del grupo Khronos (desarrolladores de WebGL, OpenGL y Vulkan entre otros) con el objeto de reducir su tamaño y el tiempo de procesamiento para para disponerlos (Grupo Khronos, 2021).

Estos muebles que colocamos deben ajustarse al espacio que se detecta a través de nuestro *framework* de realidad aumentada. Imaginemos una superficie que es irregular, los muebles deben responder a los cambios de la superficie y ajustar su altura para que su emplazamiento sea lo más fiel a la realidad. Estos problemas son más notables cuando tenemos una colección de muebles ya que al colocarlo inicialmente, todo el conjunto se trata como una unidad y puede haber problemas de ajuste, como por ejemplo, un objeto que se coloca en una pared se quede en el aire porque el modelo 3D del conjunto se haya colocado en un espacio abierto lejos de una pared.

Para ello discernimos entre dos tipos de objeto: *"OnFloorItem"*, objetos que se tienen que ajustar al suelo, y *"OnWallItem"*, muebles que deben estar pegados a la pared.

El ajuste del mueble comienza cuando este se coloca en el espacio de realidad aumentada, en cada instante se genera un *raycast,* una línea que tiene un origen y una dirección, y comprueba si ha colisionado en algún punto con una superficie detectada. Para los muebles que van al suelo, el *raycast* se realiza en dirección al suelo con el centro del mueble como origen, este procedimiento se muestra en las figuras 3.2.18, 3.2.19 y 3.2.20.

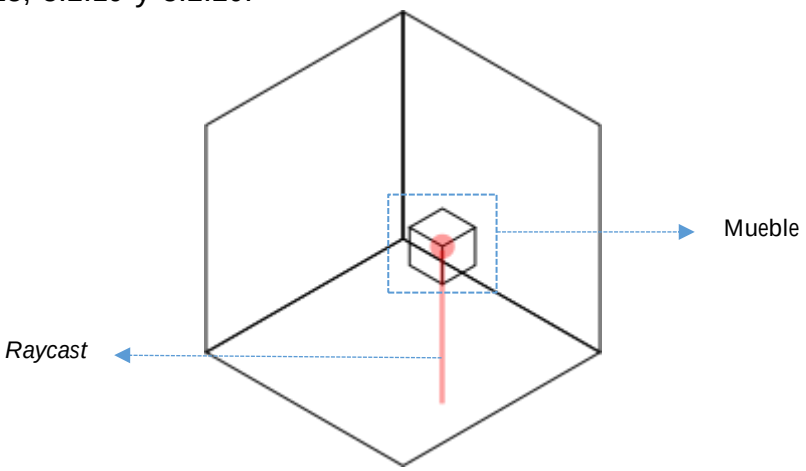


Figura 3.2.18 Mueble *"OnFloorItem"* y *raycast* en el eje de la altura comprobando si colisiona.

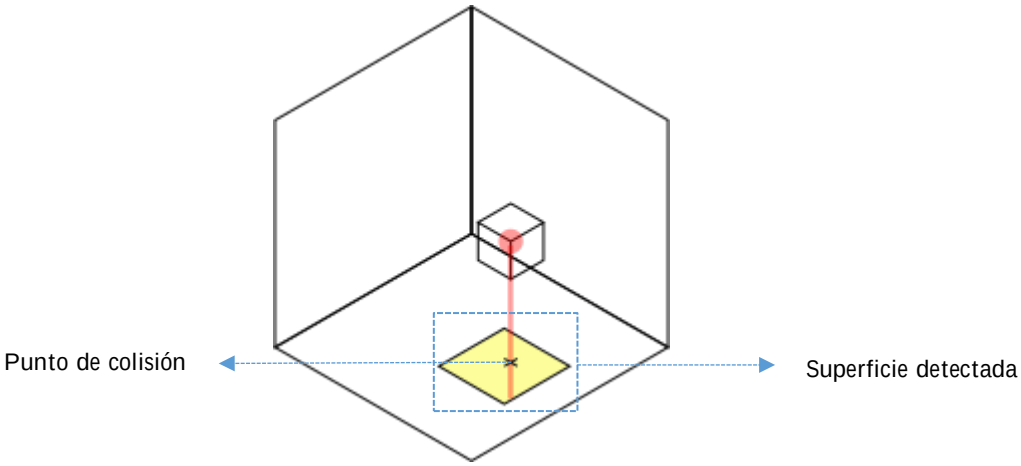


Figura 3.2.19 *Raycast* colisiona con una superficie detectada en un punto.

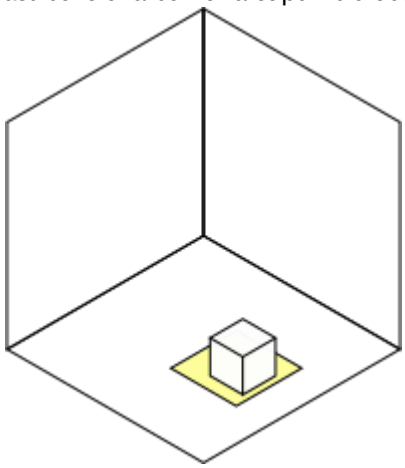


Figura 3.2.20 El mueble se coloca en el punto de colisión.

En el caso de los muebles *"OnWallItem"*, realizamos cuatro *raycasts* en las direcciones norte, sur, este y oeste para detectar alguna superficie vertical en la que colocar nuestro mueble.

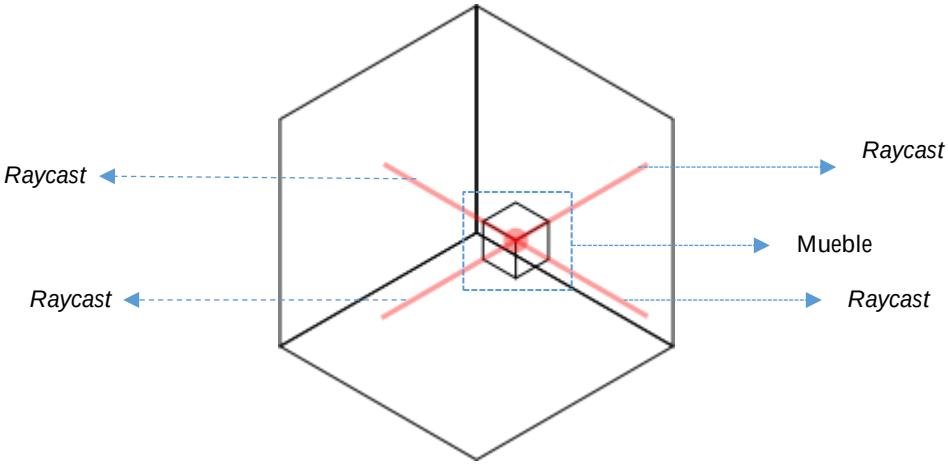


Figura 3.2.21 Mueble "OnWallItem" y los 4 *raycast* comprobando si colisiona con alguna superficie vertical.

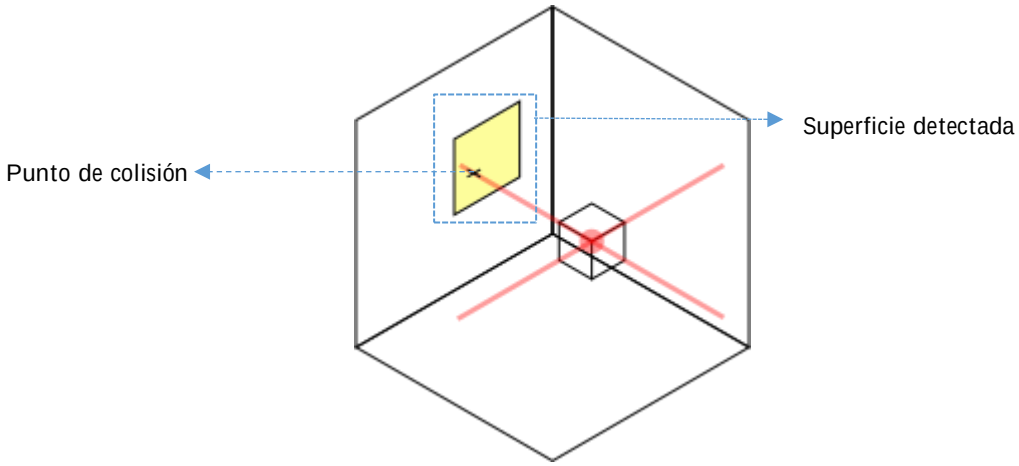


Figura 3.2.22 *Raycast* colisiona con una superficie vertical detectada en un punto.

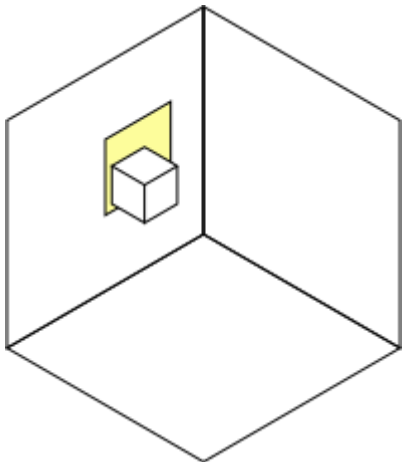


Figura 3.2.23 El mueble se coloca en el punto de colisión.

3.3

Desarrollo de la aplicación de administración

La aplicación de administrador es la aplicación que se encarga de gestionar los datos sensibles de la aplicación, en nuestro caso, de los muebles y los usuarios. En esta ocasión hemos decidido realizar una página web, ya que pensamos que sería el entorno de ejecución más sencillo para este tipo de aplicaciones.

El *framework* que hemos elegido para crear esta aplicación es Angular, desarrollado por Google y orientado a aplicaciones de página única, es decir, toda la web está contenida en una única página en la que los elementos se cargan dinámicamente para evitar recargar el código HTML, Javascript y CSS dando una experiencia más fluida en su uso.

Figura 3.3.1 Logo de Angular.

Escrito en TypeScript, Angular basa su arquitectura en componentes que se organizan en *NgModules*, existe un *NgModule* raíz con el mecanismo de arranque de la aplicación (*Bootstrapping*). Cada componente está asociado a un *template,* un código HTML que define elementos visuales que nuestro componente es capaz de interactuar. Además, los componentes usan servicios que se 'inyectan' cómo dependencia, estos servicios definen una funcionalidad determinada con el objeto de separar las diferentes características de la aplicación (Angular, 2021).

Servicios

Para nuestra aplicación hemos definido diversos servicios:

- AuthService, gestiona la autenticación del usuario. Realiza la petición HTTP POST a la ruta de nuestro *backend, /api/auth/* para registrar al usuario, cómo resultado de esta petición se nos devuelve un *token* que debemos almacenar en el almacenamiento local del navegador para usar más tarde.

- TokenInterceptorService, implementa la interfaz '*httpInterceptor'*, esta interfaz nos permite hacer un servicio el cual intercepta las peticiones HTTP y realiza alguna operación antes de lanzar la petición. En este caso, intercepta las peticiones y añade la cabecera *'Authorization'* con el valor 'Bearer ' más el *token* que hemos guardado en el almacenamiento local del navegador cómo se ha explicado en el servicio AuthService.

- UserService, este servicio se encarga de comunicarse con las rutas de nuestro *backend* relacionadas con usuarios.

- FurnitureService, este servicio se encarga de comunicarse con las rutas de nuestro *backend* relacionadas con los muebles.

- StorageService, este servicio se encarga de comunicarse con las rutas de nuestro *backend* relacionadas con los ficheros.

Componentes

Cada componente define un elemento visual de nuestra web a través de su *template*, hemos definido los siguientes componentes en nuestra aplicación.

- Header, este componente dibuja la cabecera o barra de navegación de nuestra aplicación cómo muestra la figura 3.3.2. Contiene botones para ir a la lista de muebles, la lista de usuarios o a ir a la pantalla de *login*. En caso de que ya estemos registrados el botón cambia a *log out* para eliminar el *token* del almacenamiento local del navegador.

Figura 3.3.2 Cabecera de la aplicación.

- Login, contiene un pequeño formulario con el nombre y la contraseña para iniciar sesión.

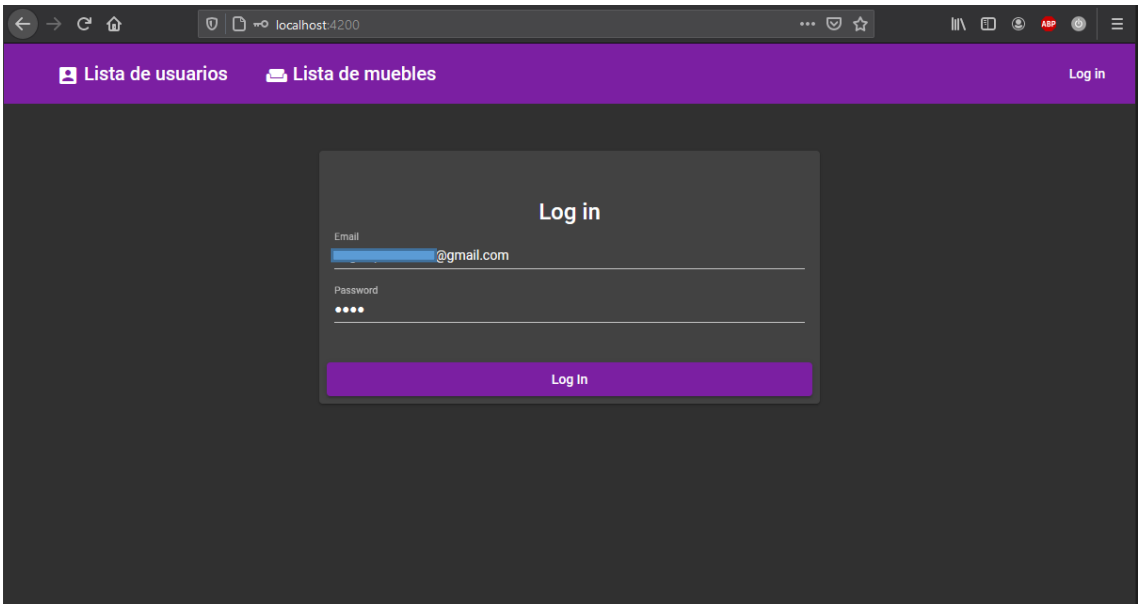


Figura 3.3.3 Pantalla de inicio de sesión.

En caso de que ocurra un error al iniciar sesión se mostrará un mensaje de error.

Figura 3.3.4 Mensaje de error en el inicio de sesión.

- UserList, Muestra la lista de usuarios que están registrados en la aplicación.

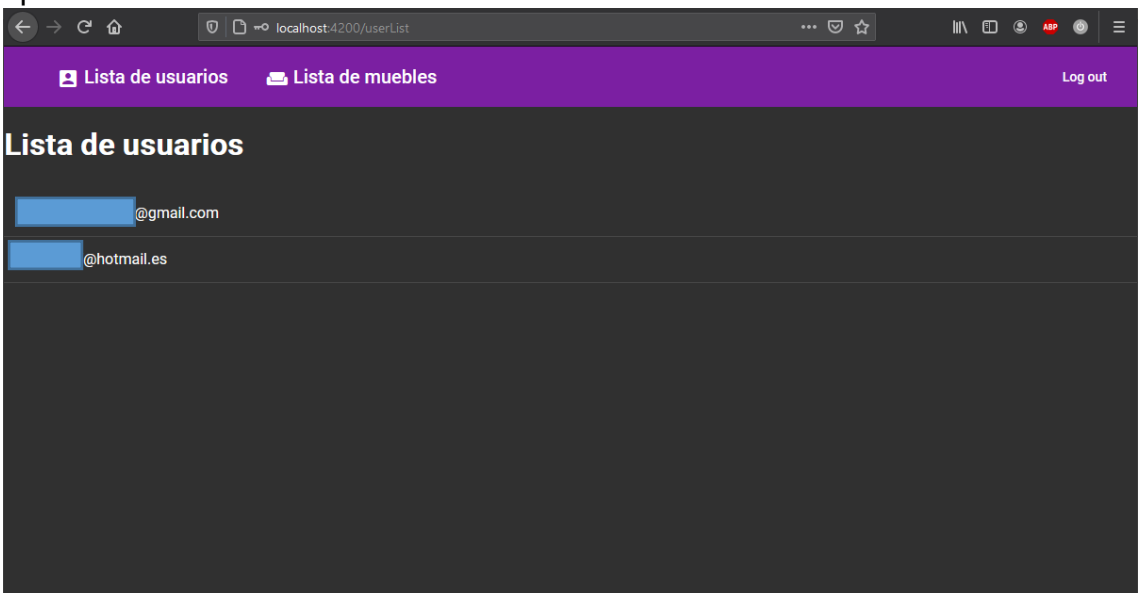


Figura 3.3.5 Lista de usuarios.

- UserInfo, Cuando hacemos clic en un elemento de la lista de usuarios, la aplicación nos dirige a una página con la información de ese usuario. Aparte presenta 3 botones adicionales: un botón de edición, un botón para volver a la lista de usuarios y un último botón, para eliminar el usuario.

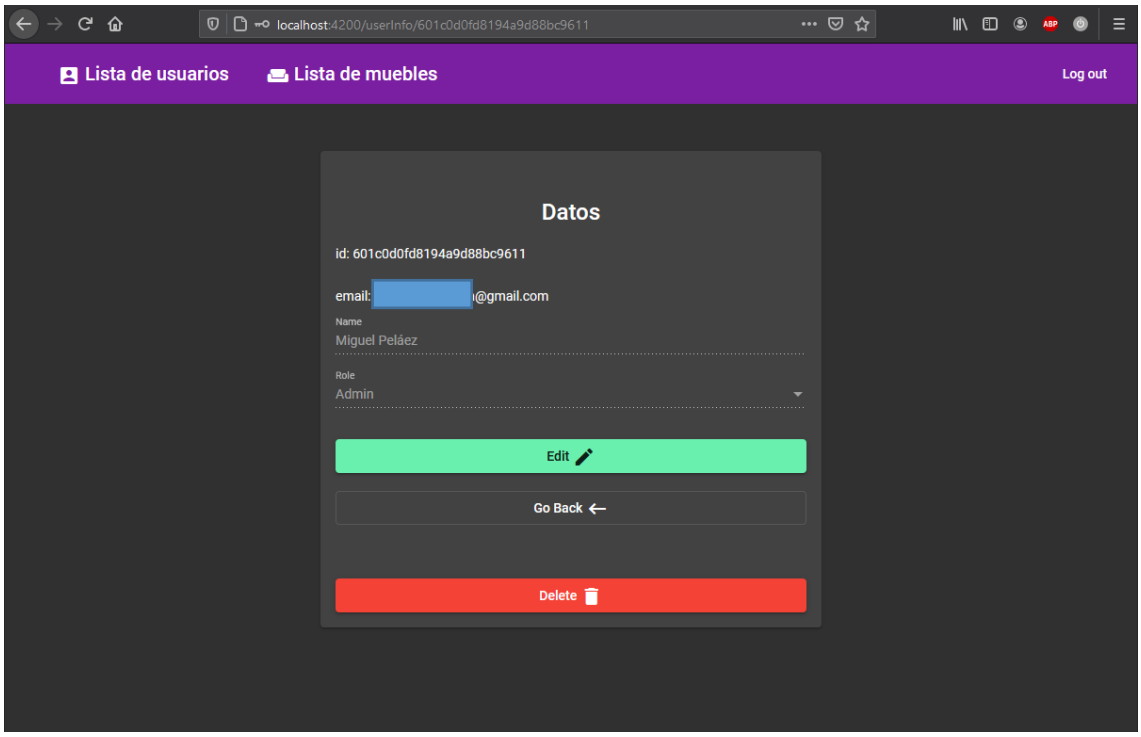


Figura 3.3.6 Datos de usuario.

Si presionamos el botón *'edit'* los elementos *name* y *role* se activan para que podamos editar sus valores, por ejemplo podemos cambiar el rol de este usuario para que en vez de ser un usuario normal pase a ser un usuario administrador. Los botones de abajo cambian a dos: *save edit* que guarda los cambios que hemos hecho o *cancel* que descarta los cambios que hemos hecho y vuelve a la pantalla anterior.

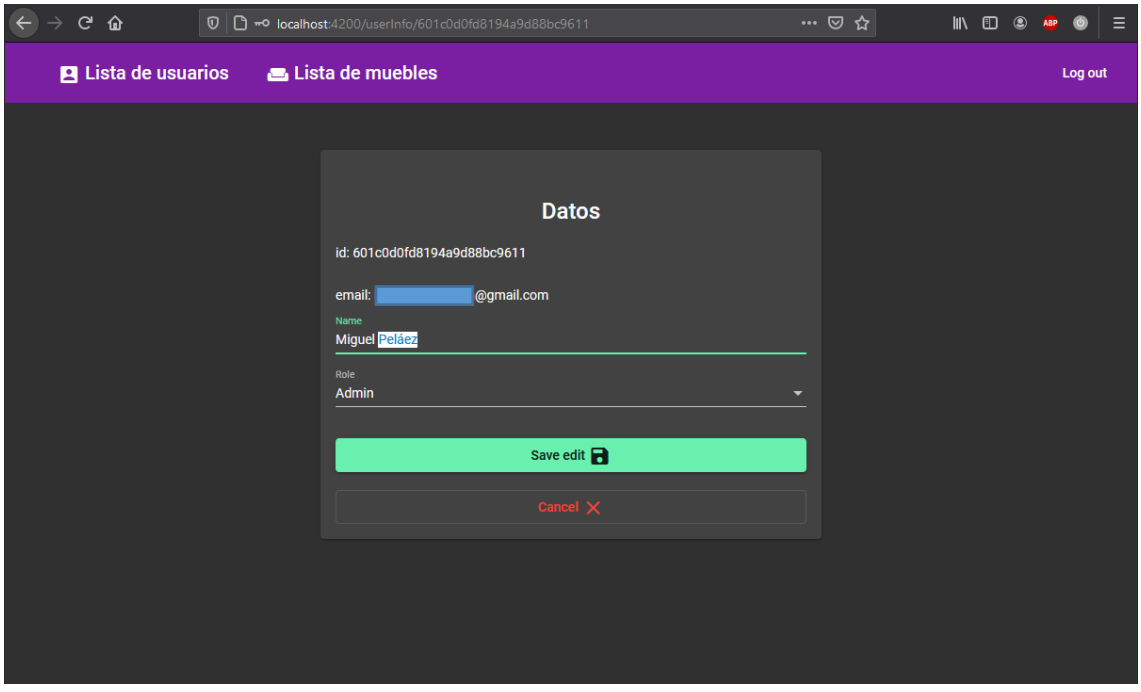


Figura 3.3.7 Edición de datos de usuario.

- FurnitureList, muestra la lista de muebles que existen en la base de datos. En la parte superior podemos encontrar un botón que nos lleva a un formulario para crear un mueble en la base de datos.

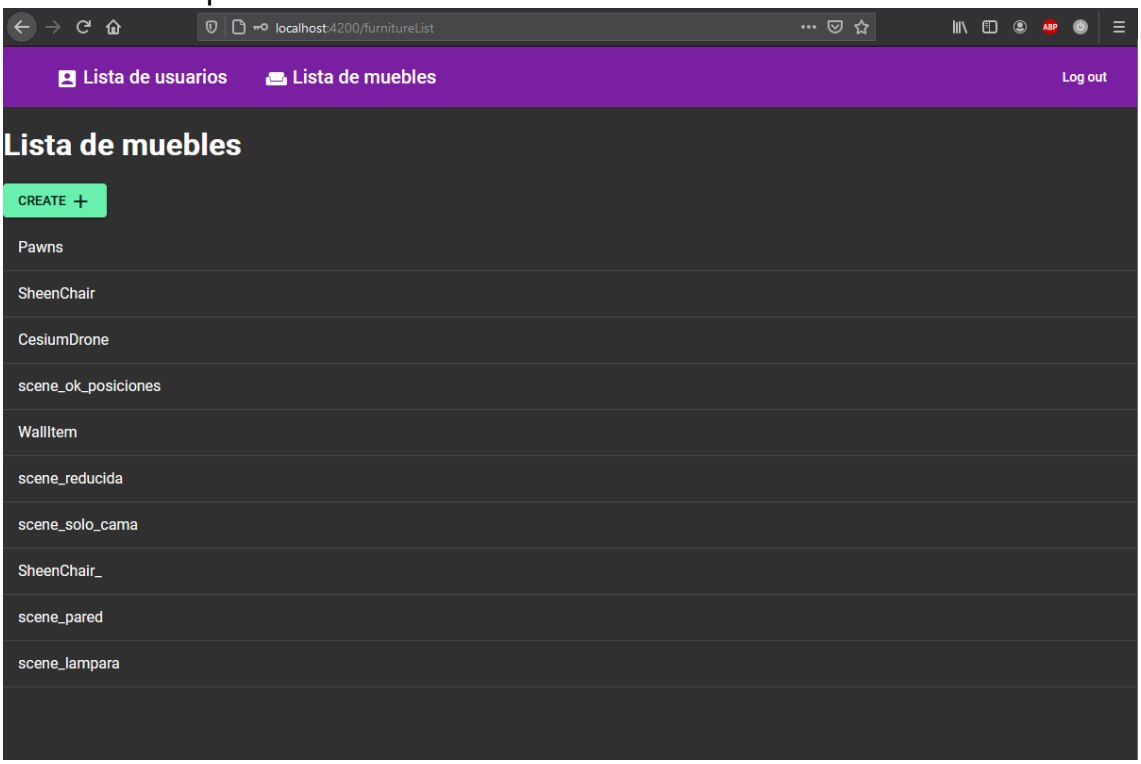


Figura 3.3.8 Lista de muebles.

- CreateFurniture, Este componente contiene un formulario nos permite crear un mueble en la base de datos. El campo visibilidad contiene un desplegable con los valores '*public*' y '*private*'. Y los dos campos inferiores permiten subir archivos, el GLTF con el modelo 3D del mueble y su imagen en formato JPG.

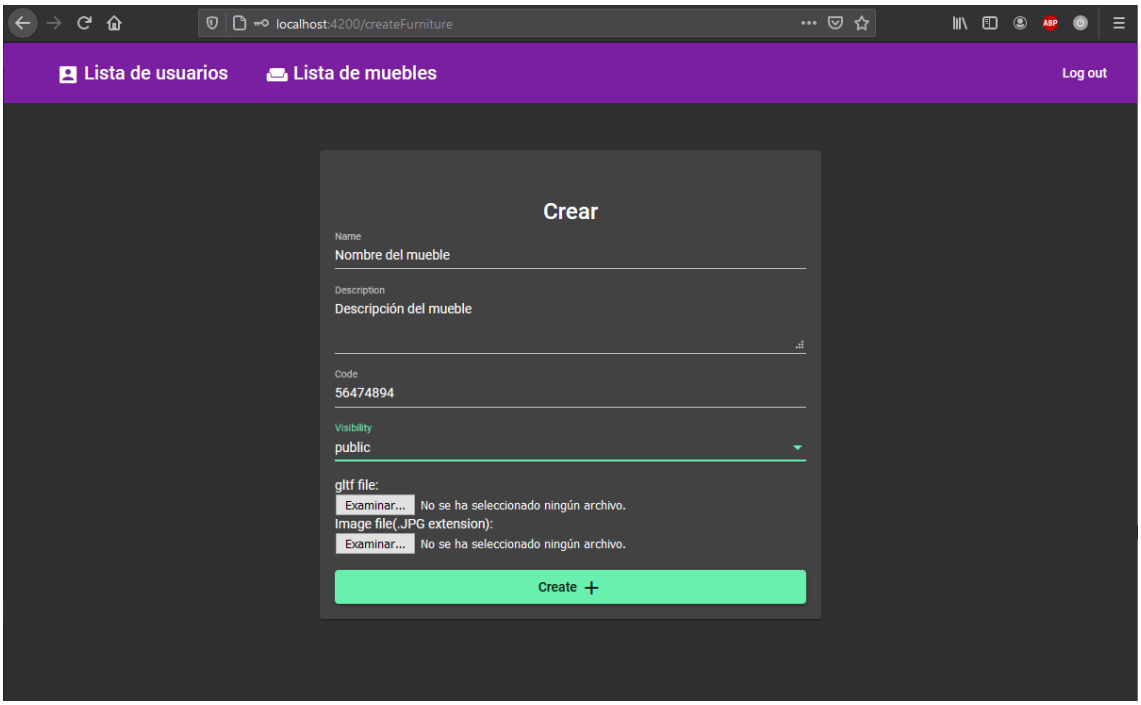


Figura 3.3.9 Formulario para crear un mueble.

- FurnitureInfo, contiene la información en la base de datos de un mueble, cómo en el componente *UserInfo* muestra varios botones para editar los datos, volver a la lista de muebles o eliminar ese mueble de la lista de datos.

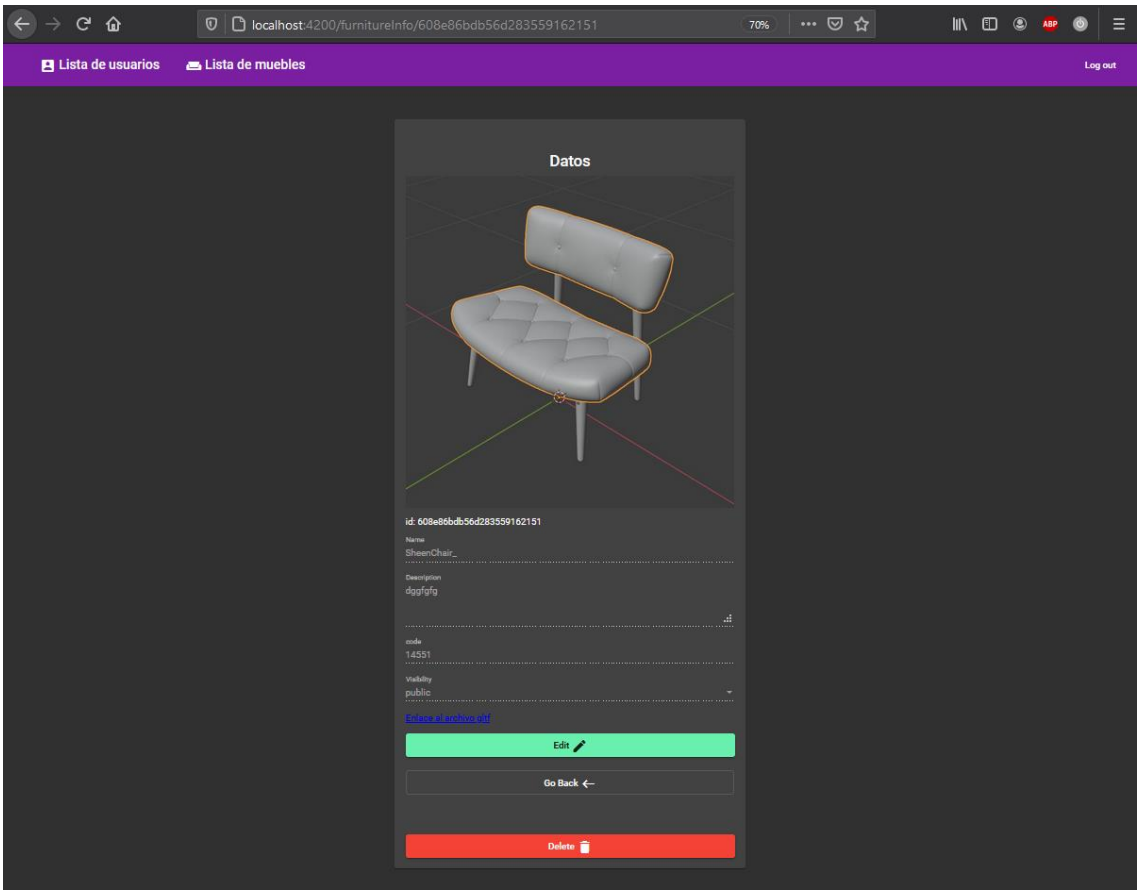


Figura 3.3.10 Información de un mueble.

Si pulsamos el botón de *edit* varios elementos de la información del mueble se desbloquearían para su edición y los 3 botones cambian a dos: *save edit* para guardar los cambios que hemos hecho y *cancel* para descartar los cambios de edición, de manera similar al componente *UserInfo* descrito anteriormente.

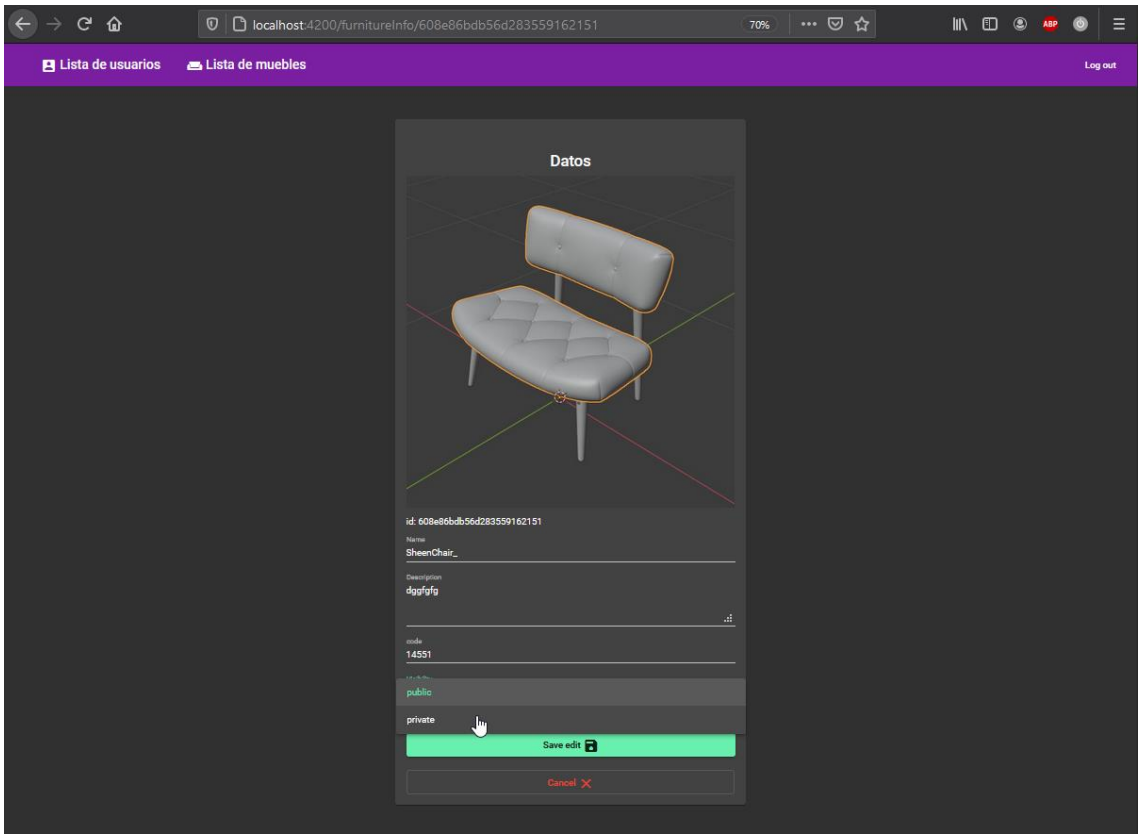


Figura 3.3.11 Edición de un mueble.

4

Conclusiones

A través de la metodología de desarrollo en espiral hemos obtenido versiones tempranas del proyecto que, aunque carente de funcionalidad, nos ha permitido establecer desde un primer momento las comunicaciones entre las aplicaciones, ya que a partir de este punto, pudimos trabajar y depurar cada pieza de nuestro proyecto de forma independiente. Utilizar HTTP como medio de transmisión en las comunicaciones nos ha dado gran libertad a la hora de elegir tecnologías, ya que casi todos los lenguajes y/o *frameworks* disponibles implementan funciones para realizar peticiones HTTP por ser uno de los protocolos de transmisión más extendidos y utilizados.

Backend

Nodejs nos ha dado la posibilidad de crear un servidor de forma rápida y simple, pero nos hemos centrado sobre todo en la biblioteca de express, que nos ha ayudado a crear nuestra api de forma sencilla, incluyendo funciones para tratar las peticiones HTTP con los middleware, los cuales nos permitieron filtrar que tipo de usuarios accedían a los recursos de la base de datos, proporcionado robustez y seguridad a nuestro *backend*.

La base de datos, al ser no relacional, nos ha aportado una gran flexibilidad, dado que no contiene reglas o restricciones sobre los datos, aunque esto podría provocar, a la larga, problemas de inconsistencia al no comprobar que los datos que se introducen son correctos. Esto se ha solucionado en las aplicaciones de usuario y de administrador, que comprueban que los datos que se van a enviar al *backend* son correctos y en el formato adecuado.

Aplicación de usuario

ARFoundation nos ofrece una alta flexibilidad en cuanto a plataformas, ya que seriamos capaces de crear aplicaciones tanto en dispositivos Android como en IOS sin tener que realizar cambios en nuestro código. Sin embargo, solo hemos podido comprobar aplicaciones creadas en Android, dado que para crear aplicaciones de IOS se necesita un dispositivo IOS y un ordenador que compile el proyecto que lleve un sistema operativo MacOS, elementos que no teníamos disponibles en el desarrollo de este proyecto.

La documentación de ARFoundation fue un poco decepcionante, no está muy ilustrada y no contiene ejemplos claros para entender su funcionamiento, tuvimos que aprender cómo usar el *framework* a través de tutoriales y fuentes externas no asociadas a ARFramework. Una vez superada la curva de aprendizaje fue bastante sencillo utilizar este *framework*.

Aunque la realidad aumentada avanza constantemente hemos encontrado un par de limitaciones: por ejemplo, en Android pocos dispositivos soportan ARCore, esto es debido a su reciente salida (2018), a más dispositivos que puedan usar la realidad aumentada, más posibles usuarios podrían utilizar nuestra aplicación; la detección de superficies no es perfecta, algunas paredes, sobre todo si son blancas o tiene pocos elementos colocados, la superficie detectada no coincide con la superficie de la pared.

Cargar los modelos 3D en tiempo de ejecución es la parte más exigente a nivel computacional y uno de los requisitos más importantes del proyecto. La biblioteca que hemos usado cumple su función, y los archivos GLTF han supuesto un buen formato para trasmitir modelos 3D a través de peticiones HTTP. Esta exigencia de computación puede suponer, en varios dispositivos con poca capacidad de procesamiento, el congelamiento de la aplicación, y si dicho congelamiento continua varios segundos, el sistema operativo del dispositivo puede enviar una señal a la aplicación para acabar con el proceso de la aplicación.

Aplicación de Administrador

A través de Angular hemos podido crear una aplicación de forma rápida que se comunica con nuestro *backend* y capaz de procesar toda la información. Su *framework* por componentes nos ha permitido crear una fragmentación de la interfaz creando independencia sobre estos componentes.

Decidimos dejarlo en una aplicación web debido a su accesibilidad, desde cualquier navegador podríamos gestionar los datos de nuestra aplicación. Una posible

ampliación de este programa sería desprenderlo de su entorno web, dado que si alojáramos nuestro proyecto en un servidor web sería accesible desde el navegador por cualquier usuario y podría ser objetivo de posibles ataques. Para ello no haría falta crear la misma aplicación en un *framework* para aplicaciones de escritorio, podríamos usar Electron, un *framework* que nos permite encapsular aplicaciones web para crear un ejecutable de escritorio sin apenas realizar cambios en nuestra aplicación.

Referencias

Angular. (2021, mayo 10). Retrieved from https://angular.io/guide/architecture

Apple. (2021). *developer.apple.com*. Retrieved from https://developer.apple.com/augmented-reality/

Baron, D. (2019). *https://www.packtpub.com/*. Retrieved from https://subscription.packtpub.com/book/game_development/9781789349337

Bcrypt. (2021, mayo 11). *www.npmjs.com*. Retrieved from https://www.npmjs.com/package/bcrypt

Billinghurst, H. K. (1999, octubre). *hitl.washington.edu*. Retrieved from http://www.hitl.washington.edu/artoolkit/Papers/IWAR99.kato.pdf

cbinsights. (2018, septiembre 20). *www.cbinsights.com*. Retrieved from https://www.cbinsights.com/research/game-engines-growth-expert-intelligence/

Gillis, A. S. (2020, septiembre). *techtarget*. Retrieved from https://searchapparchitecture.techtarget.com/definition/RESTful-API

GlobeNewswire. (2011, julio 20). *www.globenewswire.com*. Retrieved from https://www.globenewswire.com/news-release/2011/07/21/1251946/0/en/Unity-Technologies-Lands-12-Million-in-Series-B-Funding-Led-by-WestSummit-Capital-and-iGlobe-Partners.html

Google. (2021). *developers.google.com*. Retrieved abril 2021, from https://developers.google.com/ar/discover

Grupo Khronos. (2021, mayo 10). Retrieved from https://www.khronos.org/gltf/

H.Thomas, W. P. (2002, enero). *researchgate*. Retrieved from https://www.researchgate.net/publication/220427063_ARQuake_The_Outdoor_Augmented_Reality_Gaming_System

innerdrive studios. (2018, junio 25). *innerdrivestudios*. Retrieved from https://www.innerdrivestudios.com/home/controlling-content-visibility-through-vumark-ids/

Internet Engineering Task Force. (1999, junio). *tools.ietf.org*. Retrieved from https://tools.ietf.org/html/rfc2616#section-10

Internet Engineering Task Force. (2012, octubre). *tools.ietf.org*. Retrieved from https://tools.ietf.org/html/rfc6750

Internet Engineering Task Force. (2021, mayo 11). *datatracker.ietf.org*. Retrieved from https://datatracker.ietf.org/doc/html/rfc7519

Iqbal, M. (2021, mayo 6). *www.businessofapps.com*. Retrieved from https://www.businessofapps.com/data/pokemon-go-statistics/

Krueger, M. (2021). *aboutmyronkrueger*. Retrieved from https://aboutmyronkrueger.weebly.com/videoplace.html

MongoDB Inc. (2021). *www.mongodb.com*. Retrieved from https://www.mongodb.com/what-is-mongodb

Nodejs. (2021). *nodejs.org*. Retrieved from https://nodejs.org/es/about/

Norman, J. M. (2021, mayo 20). *www.historyofinformation.com*. Retrieved from https://www.historyofinformation.com/detail.php?entryid=4698

Oxford english dictionary. (2021). *oed.com*. Retrieved from https://oed.com/view/Entry/13081

P.Caudell, T. (1992). *ResearchGate*. Retrieved from https://www.researchgate.net/publication/3510119_Augmented_reality_An_application_of_heads-up_display_technology_to_manual_manufacturing_processes

Pérez, E. (2019, diciembre 12). La promesa incumplida de Magic Leap: de querer revolucionar la realidad aumentada a tener que reinventarse tras vender solo 6.000 unidades. *Xataka*. Retrieved from https://www.xataka.com/realidad-virtual-aumentada/promesa-incumplida-magic-leap-querer-revolucionar-realidad-aumentada-a-tener-que-reinventarse-vender-6-000-unidades

Rus, C. (2019, febrero 24). HoloLens 2 es oficial: reconocimiento de iris, resolución 2K por ojo y más a cambio de 3.500 dólares. *Xataka*. Retrieved from https://www.xataka.com/accesorios/hololens-2-caracteristicas-precio-ficha-tecnica

Servin, C. (2021). *ProyectoIdis*. Retrieved from https://proyectoidis.org/espada-de-damocles/

Unity. (2021, abril 10). *docs.unity3d.com*. Retrieved from https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/

Vuforia. (2021). *library.vuforia.com*. Retrieved from https://library.vuforia.com/features/objects/vumark.html

Apéndice A

Manual de Instalación

Instalación del *backend*

Requerimientos

Tener node.js a la versión 12.13.1 a través del enlace https://nodejs.org/en/blog/release/v12.13.1/ y tener npm en la versión 6.12.1 (el instalador de npm viene en el propio instalador de Node), tener ambos programas en la variable de sistema *path* (en Windows) para poder ejecutarlos desde una consola de comandos en cualquier dirección de nuestro ordenador.
Para la base de datos se deberá tener mongoDB community en la versión 4.4.1.

Instalación

En la carpeta raíz del proyecto, lanzamos una consola de comandos y escribimos el comando *npm install*. Este comando instalará todos los paquetes y bibliotecas que el proyecto necesita para funcionar, una vez ese proceso termine, lanzamos el comando *npm start* cuando aparezcan los mensajes "Escuchando puerto 3003", que indica que se ha abierto el servidor y el mensaje "conectado a MongoDb" sabremos que se ha conectado a la base de datos y nuestro servidor está listo para usarse.

Instalación de la aplicación de usuario

Requerimientos

Tener instalado Unity hub (se puede obtener el instalador del enlace https://unity3d.com/es/get-unity/download), ejecutarlo, abrir la pestaña de *"installs"* y pulsar el botón de *"ADD"* como se muestra en la figura apA.1 seleccionar una versión del motor de Unity que empiece por 2020.2 como se puede apreciar en la figura apA.2.

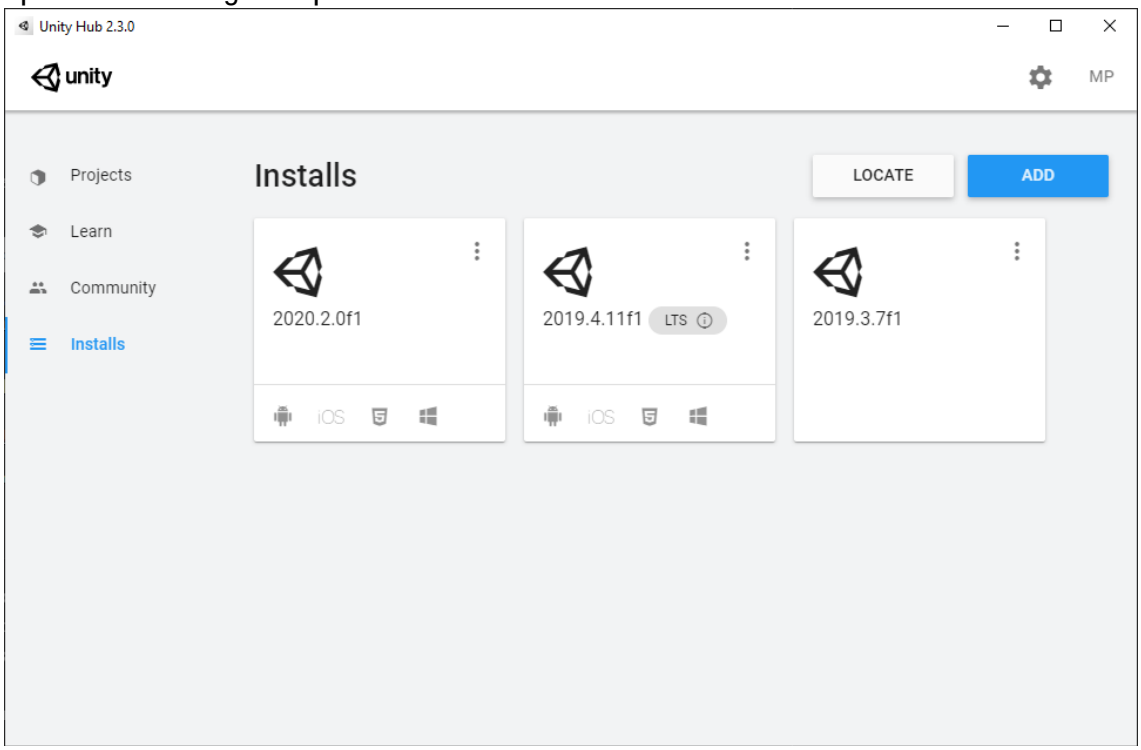


Figura apA.1 Interfaz de Unity Hub.

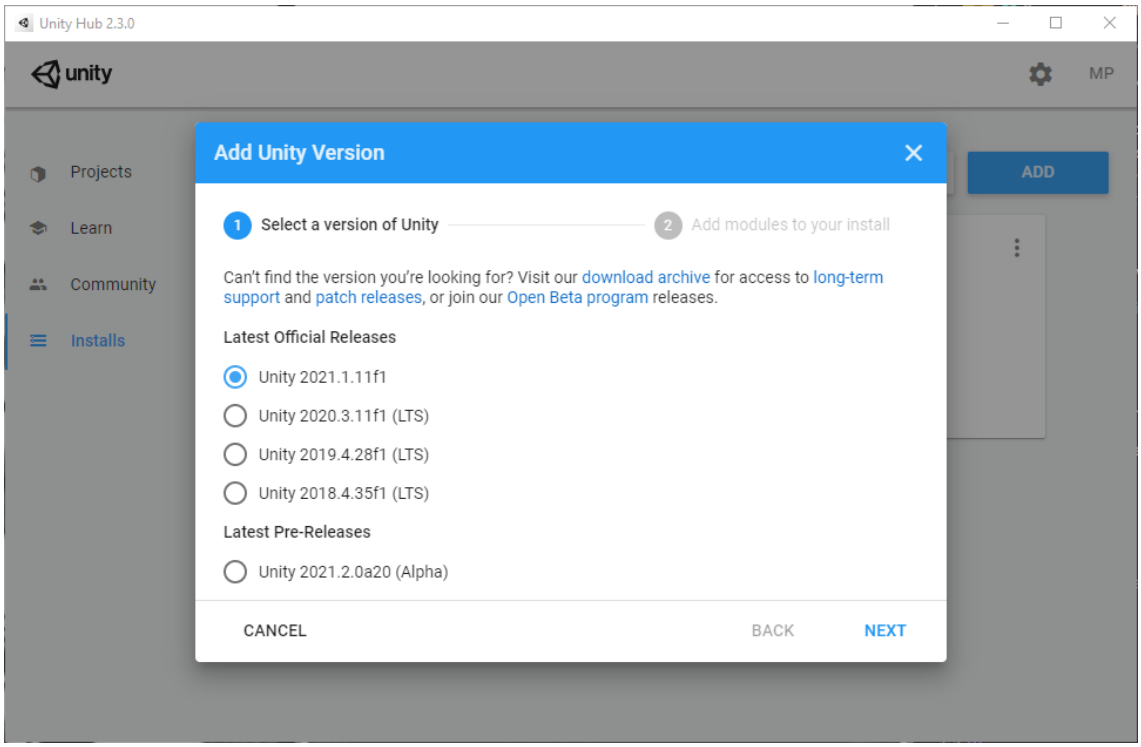


Figura apA.2 Selección de la versión de Unity a instalar.

Además, el motor debe tener los módulos de Android build support en caso que necesitemos hacer una *build* de Android o IOS build support en caso de IOS como aparece en la figura apA.3.

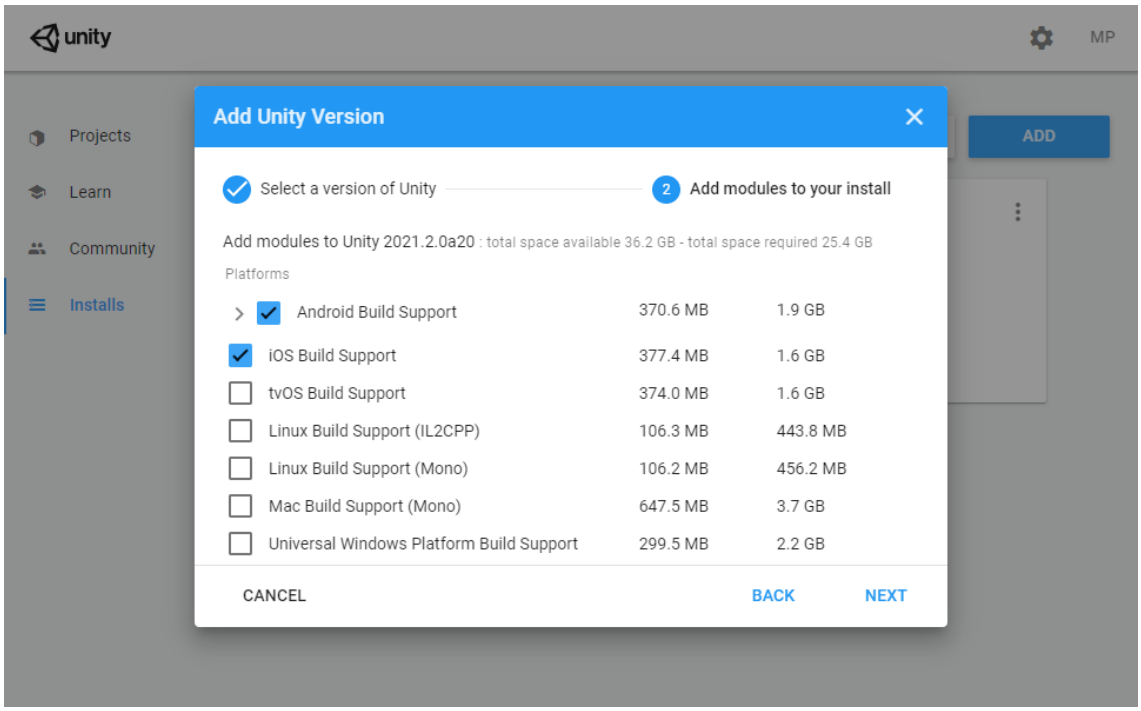


Figura apA.3 Selección de modulos a instalar en el motor de Unity

También necesitaremos un dispositivo móvil que pueda usar ARCore o ARKit. Para Android debemos tener en nuestro ordenador el programa ADB para instalar

la aplicación en nuestro dispositivo. (preferiblemente si se realiza en Windows tener la dirección del programa en la variable del sistema *path* para lanzarlo desde cualquier lugar)

Instalación

En la carpeta del proyecto buscamos la ruta */assets/Resources/* allí, encontraremos un fichero llamado config.txt, lo abrimos y cambiamos el valor que vaya después del fragmento de texto "ip=" con la dirección ip de nuestro servidor, si es en local debemos añadir el puerto de escucha que en este caso es 3003 como aparece al lanzar el servidor.

Una vez hecho esto, abrimos el proyecto en Unity. Para abrirlo, ejecutamos Unity hub y en la pestaña "projects" seleccionamos el botón ADD, se nos abrirá una ventana con el explorador de archivos y seleccionamos la carpeta del proyecto.

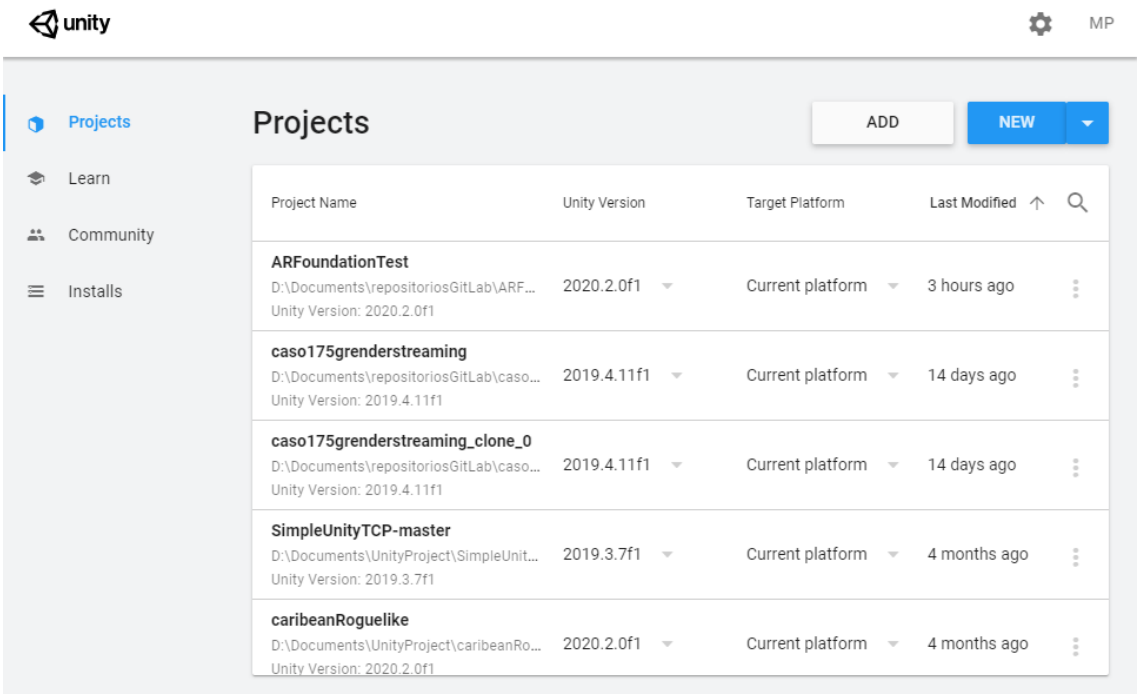


Figura apA.4 Unity Hub ventana de proyectos.

Seleccionamos el proyecto de la lista y cuando se abra el programa Unity, nos dirigimos a *File* y después a *Build Settings* aparecerá una ventana emergente. Debemos fijarnos en la parte inferior izquierda en la sección de *platform* y ver si estamos en la plataforma objetivo como se ve en la figura apA.5 si no, debemos pulsar en la plataforma objetivo y presionar el botón *switch platform*.

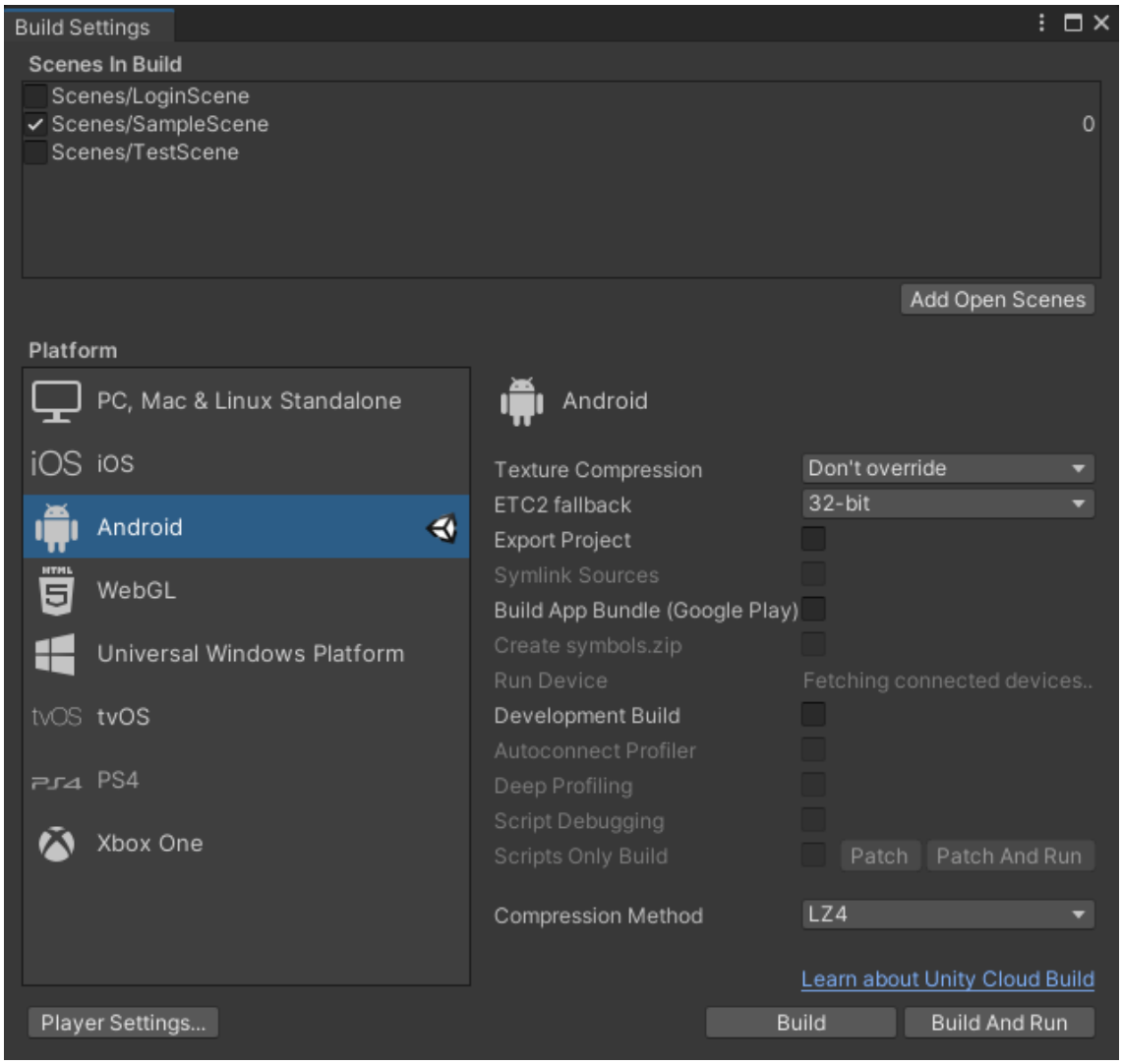


Figura apA.5 ventana build settings de Unity.

En la esquina inferior derecha de la ventana emergente veremos un botón de *build,* al pulsar nos aparecerá una ventana con el explorador de archivos preguntándonos dónde guardar el archivo .apk en el caso de Android que representa la *build* de la aplicación.

Conectamos el dispositivo móvil al ordenador vía USB. En la carpeta dónde hayamos puesto el archivo .apk debemos abrir una consola de comandos y escribir el comando *adb install* junto con el nombre del archivo apk con su extensión para instalarlo en el dispositivo. Si el programa ADB no lo tenemos en la variable *path,* se recomienda guardar el archivo .apk en la misma carpeta del programa.

Instalación de la aplicación de administrador

Requerimientos

Tener node.js a la versión 12.13.1 y tener npm en la versión 6.12.1, tener ambos programas en la variable de sistema *path* (en Windows) para poder ejecutarlos desde una consola de comandos en cualquier dirección.
Instalar Angular CLI en la versión 10.1.6 podemos instarlo usando el comando *npm install -g @angular/cli@10.1.6.*

Instalación

En la carpeta raíz del proyecto lanzamos una consola de comandos y escribimos el comando *npm install*, este comando instalará todos los paquetes y bibliotecas que el proyecto necesita para funcionar, una vez ese proceso termine buscamos el archivo config.json que se encuentra en la dirección */src/assets/* de la carpeta del proyecto.

Abrimos el archivo y configuramos la variable *serverIP* del *json* por la dirección de nuestro servidor, si es en local debemos añadir el puerto de escucha que en este caso es 3003 como aparece al lanzar el servidor.

Finalmente, con una ventana de comandos en la carpeta raíz del proyecto ejecutamos el comando *npm start* cuando termine de compilar el proyecto se estará ejecutando en la dirección *http://localhost:4200.*

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA

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