UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

Grado en Tecnologías Interactivas

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

"ESCAPARATE RETROVISIÓN: Escaparate interactivo en las ópticas ClaraVisión"

TRABAJO FINAL DE GRADO

Autor/a:

Iván Romero Ruiz

Tutor/a:

Stella María Heras Barberá

María Victoria Torres Bosch

GANDIA, 2021

Resumen y palabras clave

Este proyecto se basa en el desarrollo “bajo demanda” de un software que se implementará dentro de un escaparate interactivo (pantalla táctil) en una de las ópticas ClaraVisión, con el objetivo de atraer clientes a dicho establecimiento haciendo posible que estos se prueben gafas de manera virtual usando la cámara del escaparate.

La aplicación mostrará publicidad mientras no haya ningún usuario interactuando con el escaparate. Al tocar la pantalla, el usuario podrá probarse las gafas disponibles, viéndose a sí mismo en pantalla con las gafas virtuales superpuestas sobre su rostro. Además, podrá realizar búsquedas sobre las gafas filtrando por diferentes criterios (por ej. color, material, forma, género, tipo, etc.). Cuando el usuario haya seleccionado las gafas que le interesen, podrá hacerse fotografías en el momento con la cámara del escaparate virtual y mandárselas al correo electrónico especificado durante la sesión.

Usaremos Unity y el lenguaje C# para la creación de la aplicación, que funcionará en dispositivos Android.

Palabras clave: gafas; óptica; realidad aumentada; Android; Unity;

Abstract and key words

This project is based on the development "on demand" of a software that will be implemented within an interactive showcase (touch screen) in one of the ClaraVisión opticians, with the aim of attracting customers, making it possible for them to test glasses virtually using the camera of the showcase.

The application will display advertising as long as there are no users interacting with the screen. By touching it, users will be able to test the glasses available, seeing themselves on the screen with the virtual glasses superimposed on their face. In addition, glasses could be filtered by different criteria (e.g. colour, material, shape, gender, type, etc.). When the users select any glasses, they will be able to take photographs at the moment with the camera of the virtual showcase and send them to the mail specified during the session.

We will use Unity and C# language for the creation of the application, which will work on Android devices.

Key words: glasses; opticians; augmented reality; Android; Unity;

Índice

Tabla de ilustraciones

Capítulo 1. Introducción

1.1 Presentación y objetivos

Este proyecto se basa en el desarrollo "bajo demanda" de un programa en Android que se implementará dentro de un escaparate interactivo (pantalla táctil) en una de las ópticas **ClaraVisión**, con el objetivo de atraer clientes a dicho establecimiento haciendo posible que estos se prueben gafas de manera virtual usando la cámara del escaparate.

La aplicación mostrará publicidad mientras no haya ningún usuario interactuando con el escaparate. Al tocar la pantalla, el usuario podrá probarse las gafas guardadas en la aplicación, viéndose a sí mismo en pantalla con las gafas virtuales superpuestas sobre su rostro. Además, podrá realizar búsquedas sobre las gafas filtrando por diferentes criterios (cristal, material, forma, color, género). Cuando el usuario haya seleccionado las gafas que le interesen podrá hacerse fotografías en el momento con la cámara del escaparate virtual y mandárselas a su perfil de la aplicación móvil de la óptica, promoviendo de esta forma que el usuario se descargue dicha aplicación. Las fotografías se enviarán al correo electrónico que se haya introducido antes de enviar las imágenes.

Con nuestra aplicación pretendemos:

- Atraer a los clientes mediante tecnologías llamativas.
- Proporcionar un acceso cómodo y diferente a los productos de la óptica.
- Ofrecer un método más higiénico para la selección de gafas.
- Diferenciar a la óptica de la competencia.

1.2 Estructura de la memoria

Esta memoria está organizada de la siguiente forma:

- En el capítulo 2, **Entorno y estado del arte**, exponemos el estado actual de algunas aplicaciones de ópticas existentes, así como de proyectos actuales similares a este.

- En el capítulo 3, **Análisis de requerimientos**, listamos los requerimientos iniciales para el desarrollo del proyecto además de definir y explicar brevemente el mapa de navegación entre pantallas que planteamos inicialmente.
- En el capítulo 4, **Diseño de la aplicación**, mostramos el diagrama de clases de la aplicación y comentamos las clases y su diseño.
- En el capítulo 5, **Implementación**, explicamos la metodología de trabajo y las herramientas de desarrollo utilizadas en el proyecto, la estructura de directorios de la aplicación, la estructura de elementos de Unity, se detallan los algoritmos más complejos y con mayor relevancia y los problemas de implementación más importantes que hemos resuelto.
- En el capítulo 6, **Manuales**, definimos el manual de instalación y la guía de uso de la aplicación.
- En el capítulo 7, **Evaluación**, describimos las pruebas de validación realizadas durante y al final del desarrollo de la aplicación y los resultados de la evaluación con usuarios reales.
- En el capítulo 8, **Modelo comercial**, resumimos nuestra idea de negocio de este proyecto.
- En el capítulo 9, **Conclusiones**, presentamos las conclusiones y proponemos las futuras mejoras que tenemos en mente para cuando comercialicemos la aplicación.
- **Bibliografía**.

Capítulo 2. Entorno y estado del arte

2.1 Entorno

Crecer como empresa hoy en día no es algo fácil, pero lo que está claro es que para ello hay que digitalizar ciertos procesos para llegar a la mayor cantidad de personas posible. Y lo importante no es solo eso, sino investigar las tecnologías emergentes y usarlas en favor de la empresa. Por ello surge este proyecto, como una forma de llamar la atención de los clientes y ofrecerles una experiencia agradable y diferenciadora de la competencia del sector.

Tras ponernos en contacto con el grupo de ópticas ClaraVisión, mostrarnos como trabajan y discutir sobre las necesidades y los objetivos que tenían en mente, surgió la estructura básica de este proyecto.

Retrovisión consiste en una aplicación para una pantalla digital con la cual personas que pasen por delante de ella podrán usar la cámara de la misma para verse “reflejados” y probarse gafas mediante Realidad Aumentada, cosa muy útil tanto por la curiosidad que pueda despertar en los clientes, como por la posibilidad de mostrar los productos sin necesidad de tenerlos físicamente, como por la opción que ofrece de evitar demasiado contacto físico, cosa que en esta época de plena pandemia mundial puede favorecer a todo el mundo implicado.

2.2 Estado del arte

En el sector de las ópticas existen diversas aplicaciones para probarse gafas sin necesidad de tenerlas físicamente, como es el caso de **Afflelou** [1] y **VisionLab** [2], las cuales se pueden usar online en sus respectivas páginas web. La aplicación de Afflelou es una extensión de su catálogo, es decir, es una funcionalidad extra del navegador de sus distintos productos, y la de VisionLab solo permite probarse gafas que la propia aplicación te recomienda. En cambio,

en cuanto a software, nuestra aplicación se diferencia por ser el probador virtual la principal función y por disponer de todo el catálogo disponible fácilmente accesible y con diversos filtros que ayudan al usuario a encontrar lo que busquen de manera intuitiva.

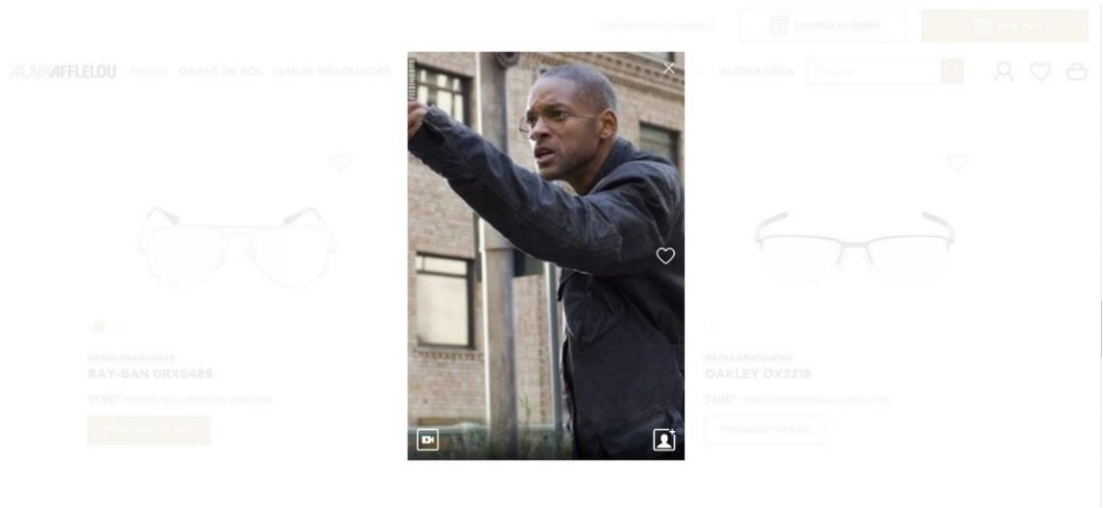


Ilustración 1. Probador de Afflelou (Imagen de https://www.afflelou.es/gafas-graduadas/ visitada el 8 de junio de 2021).

Ilustración 2. Probador de VisionLab (Imagen de https://www.visionlab.es/es/kumer/probador-de-lentes-progresivas visitada el 8 de junio de 2021).

Otro punto a favor de Retrovisión es que al ofrecer una funcionalidad que no tienen todas las ópticas, la pantalla digital, que será visible y accesible desde fuera de la tienda, puede llamar la atención hasta de la gente que pase cerca y no tenga la intención inicial de mirar gafas, aumentando la clientela interesada en los productos de ClaraVisión.

Existe un proyecto que se está llevando a cabo actualmente (2021) por parte de Timiak Tech, llamado **Nice2SeeU** [3], similar a este y con previsión de llegar a todas las ópticas de España antes de fin de año, enfocado también en un probador de gafas virtual presente en las ópticas. Al no haber visto la luz todavía, no hay mucha información pública sobre cómo se ha llevado a cabo el mismo.

Nos hemos dado cuenta de que cada vez más empresas se están interesando en las nuevas tecnologías relacionadas con la Realidad Aumentada, que son una gran apuesta como impulso sobre la competencia, ya que son de gran utilidad y llaman la atención por las distintas interacciones que permiten, por lo que en estos próximos años se espera un gran aumento de su uso. La clave será cuan pronto y cuan bien se implementen y se ajusten estas tecnologías a las características de la empresa que las utilice. Aquí es donde nace Retrovisión, con las expectativas de llegar en el momento adecuado al mercado e impulsar los negocios en los que se implemente, formando parte del comienzo de una nueva era tecnológica.

2.2.1 Unity

Unity [4] ha sido el **motor de desarrollo** utilizado para la creación de la aplicación Android de este proyecto.

Como tal, es un software usado generalmente para el desarrollo de videojuegos que ofrece un entorno donde ir ubicando objetos 3D/2D de manera intuitiva, así

como scripts (archivos de código) que controlan el funcionamiento de dicho entorno y elementos.

Se ha elegido este entorno de desarrollo debido a la gran comunidad que hay detrás de él, la cual genera material de uso público y muchas veces gratuito, no teniendo así que comenzar desde cero cada vez que se empieza un proyecto. De esta manera se puede empezar con una base sólida y darle más importancia al diseño de la interfaz y a la experiencia de usuario, dando como resultado una aplicación más cómoda y funcional.

En el siguiente gráfico podemos observar como **Unity es, en España, el motor de juego más utilizado**, monopolizando casi a la mitad de los videojuegos producidos en España desde 1994 hasta 2020.

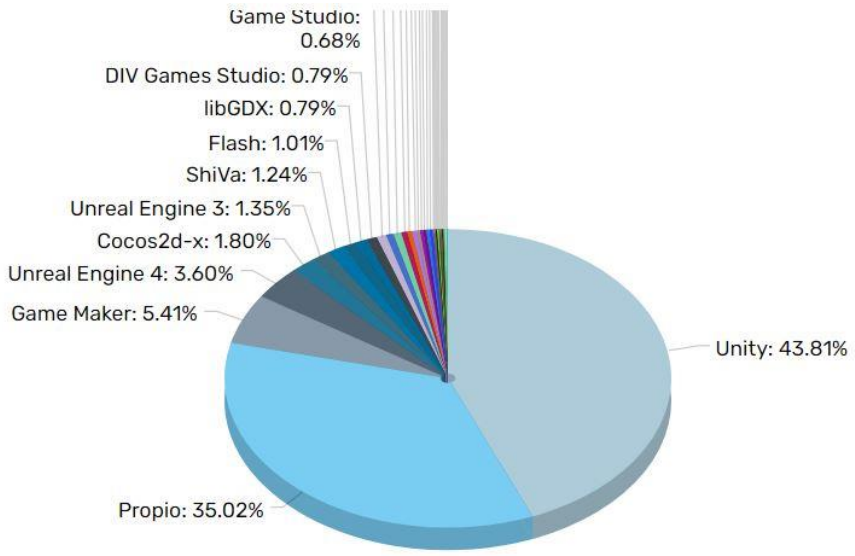


Ilustración 3. Gráfico estadístico del uso de motores de videojuegos en España entre 1994 y 2020. (Imagen de https://www.devuego.es/bd/estadisticas/distribucion-motores/?ano= visitada el 10 de junio de 2021).

Por otra parte y según la página de estadísticas DeVuego [5], en 2019 un 33.33% de los juegos creados en España fue con **Unreal Engine** y en 2020 un total del 100%, por lo que podemos deducir que este motor está en auge y quizá

le vaya ganando parte del terreno a Unity y más con la salida de Unreal Engine 5, que tiene varias mejoras importantes, como soportar proyectos con una gran cantidad de polígonos, cosa muy beneficiosa y que puede llegar a cambiar las reglas del juego tal y como las conocemos. Aun así, habiendo probado estos dos motores, para este tipo de proyectos no relacionados con videojuegos ni animaciones, nos parece una mejor opción usar Unity.

Hay que tener en cuenta que existe la versión gratuita de Unity, que es la que se ha usado para este proyecto, pero si se superan los USD 100.000 de beneficios, será necesario adquirir la versión profesional.

Capítulo 3. Análisis de requerimientos

En esta sección recopilamos los requisitos funcionales (RF) y los no funcionales (RNF) que planteamos inicialmente tras hablarlo con el gerente de ClaraVisión y tras meditar concienzudamente, teniendo en cuenta nuestras experiencias con el uso de aplicaciones Android,

Requisitos de la aplicación:

- **RF1.** Se debe de poder ver el usuario a si mismo con un modelo 3D de gafa superpuesto a su cara centrado correctamente en todo momento.
- **RF2.** Se debe de poder hacer fotografías del usuario con las gafas superpuestas.
- **RF3.** Se deben de poder enviar dichas fotografías al correo electrónico que especifique el usuario.
- **RF4.** Se debe de poder navegar por el catálogo de modelos de gafas disponibles de la óptica que se puedan probar virtualmente, es decir, de los cuales exista un modelo 3D.
- **RF5.** Se debe de poder seleccionar las gafas favoritas de entre todas las disponibles.
- **RF6.** Se debe de poder ver publicidad de la óptica cuando no haya ningún usuario interactuando con la pantalla.
- **RNF1.** Todas las posibles interacciones deben de ser claras para una mejor experiencia del usuario.

Teniendo estos requisitos en cuenta surgió la siguiente **secuencia de navegación preliminar** que ha ido variando ligeramente durante el desarrollo del proyecto.

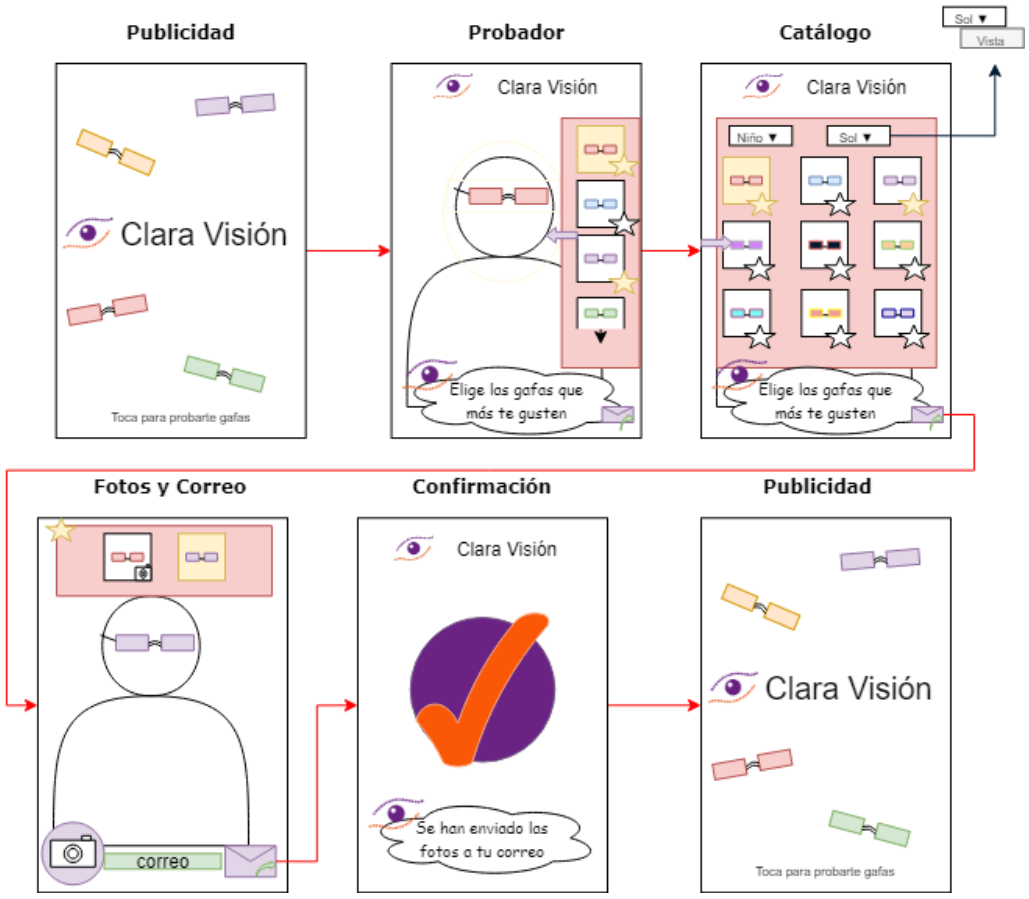


Ilustración 4. Secuencia de navegación de la aplicación diseñada en un principio.

Con todo esto en mente, decidimos que la aplicación se dividiría en los siguientes apartados con distintas funciones cada uno:

- **Apartado publicidad**. Este estará activo por defecto siempre que no haya nadie interactuando con la pantalla. En él se mostrarán imágenes publicitarias de la óptica, así como la existencia del probador virtual para que las personas que lo vean sepan que pueden interactuar con el dispositivo, no solo verlo.
- **Apartado probador**. Apartado principal de la aplicación con diversas funciones:
    - Detección de una cara mediante la cámara instalada en el dispositivo y colocación de las gafas virtuales a la altura de los ojos en tiempo real.
    - Catálogo de gafas disponibles y seleccionables que se podrán probar los usuarios.
    - Filtros para una navegación más precisa por el catálogo.
    - Hacer fotografías de los usuarios con las gafas superpuestas.

    - Selección de gafas favoritas para disponer de ellas más fácilmente a la hora de hacer fotografías.
- **Apartado galería.** Último apartado donde se podrán ver y seleccionar las fotografías obtenidas durante la sesión y enviarlas por correo.

Capítulo 4. Diseño de la aplicación

Nuestra aplicación sigue el siguiente **diagrama de clases** (posteriormente se presentarán los atributos y métodos de las clases más importantes):

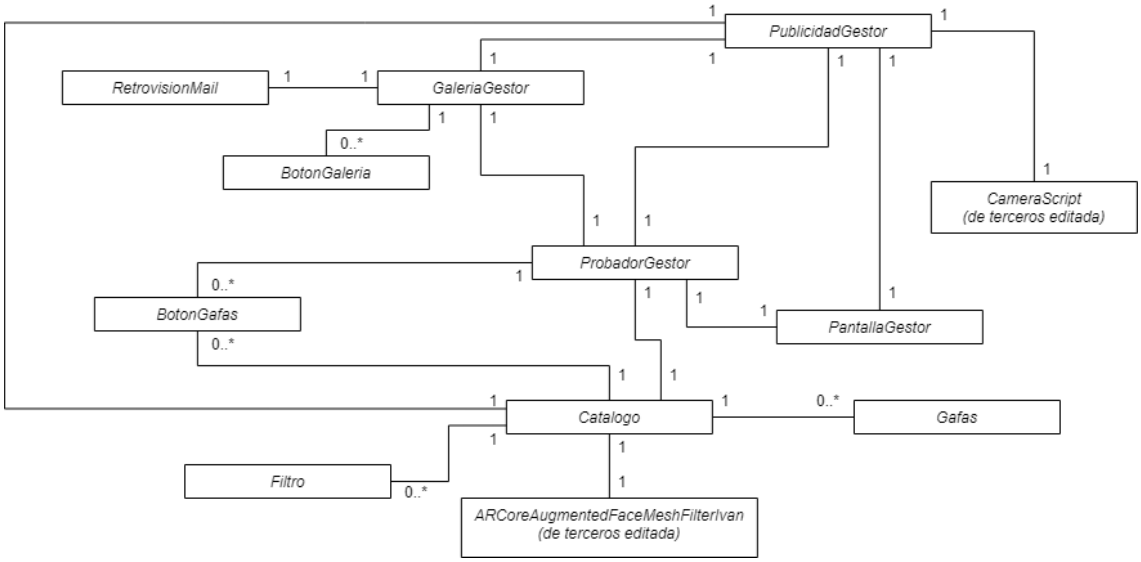


Ilustración 5. Diagrama de clases.

Se puede observar que las clases más importantes y que tienen un número de interacciones mayor que las otras son *PublicidadGestor*, *ProbadorGestor*, *GaleriaGestor*, y *Catálogo*. Las 3 primeras son las encargadas del funcionamiento de cada uno de los 3 apartados de la aplicación mencionados anteriormente en el Capítulo 3, mientras que la última (*Catálogo*) es la clase que organiza y hace posible la selección de las gafas disponibles para ser probadas.

Debido a la gran cantidad de parámetros y métodos de las clases, no se han integrado en el diagrama de clases anterior, pero los mostraremos a continuación.

Seguidamente exponemos las funciones de **las 4 clases más importantes** mencionadas anteriormente:

- *PublicidadGestor*: Inicializa los elementos de la pantalla de publicidad con sus respectivos tamaños, activa el carrusel de imágenes publicitarias y sus transiciones y cambia al siguiente apartado al tocar la pantalla.
- *ProbadorGestor*: Inicializa el apartado probador, controla las interacciones de los botones de información, de cerrar sesión, de desplegar/contraer el catálogo, de alternar la galería y el catálogo, de hacer fotos y de alternar

la visibilidad del catálogo; gestiona el contenedor de gafas favoritas para que aparezca/desaparezca en el apartado fotos dependiendo de cuantos favoritos haya seleccionados y controla las animaciones de los botones.

- *GaleríaGestor*: Configura los tamaños de los botones con las fotografías hechas durante la sesión, añade botones cada vez que se hace una fotografía accediendo a su ubicación en el dispositivo, permite seleccionar o deseleccionar dichos botones y se comunica con *RetrovisionMail* para enviar las imágenes al correo electrónico.
- *Catalogo*: Inicializa el catálogo de botones con las gafas disponibles e instancia todas las gafas con sus respectivos datos (color, forma, material...) dejándolas inactivas, de forma que cada una se activará cuando se toque su botón (BotonGafas).

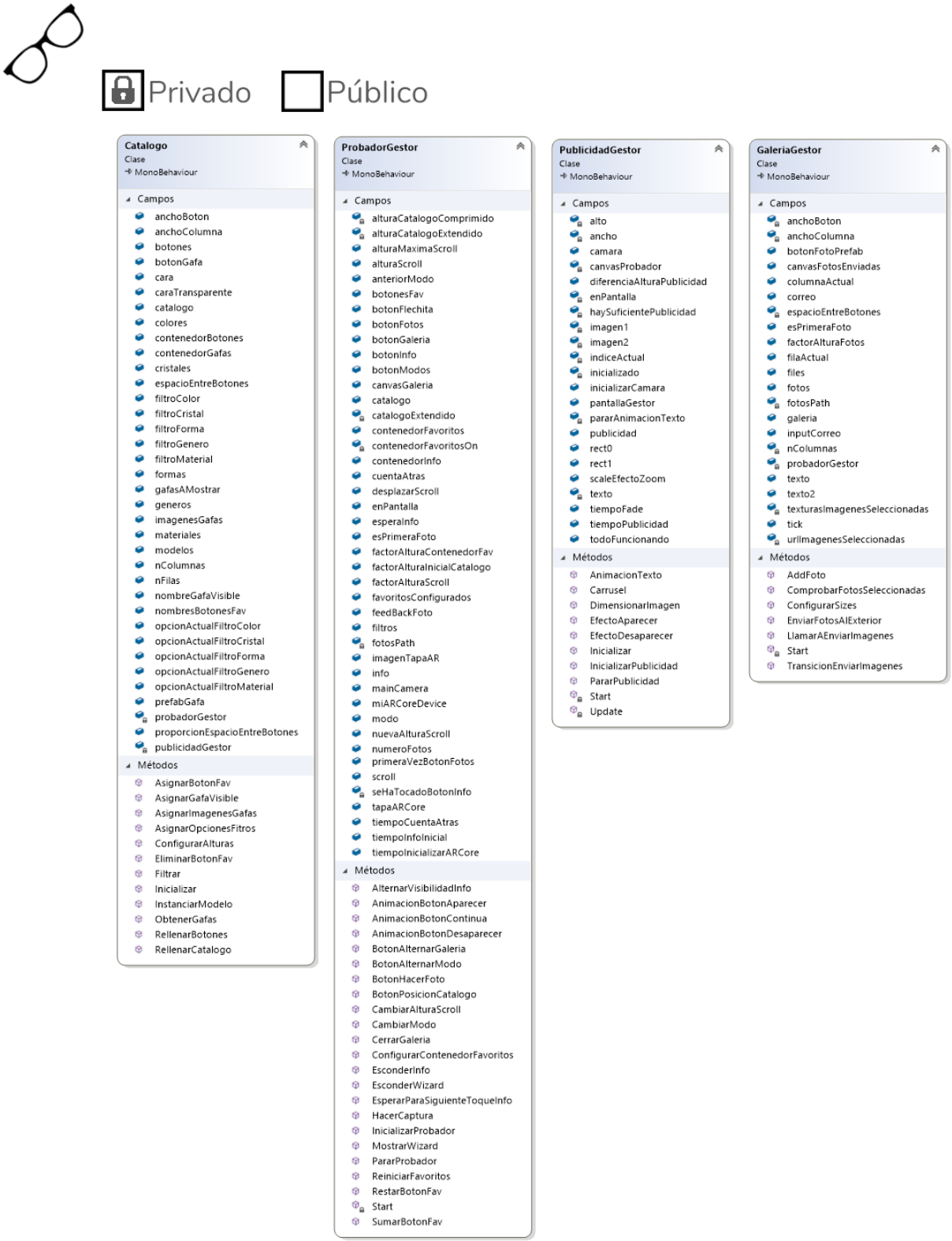


Ilustración 6. Diseño de clases generado por Visual Studio [6] (1).

Las demás clases tienen las siguientes funciones:

- *BotonGaleria* y *BotonGafas*, encargadas de controlar el comportamiento y las interacciones de los botones de la galería y los del catálogo respectivamente. La primera permite seleccionar las fotografías deseadas

para luego ser enviadas mientras la segunda se usa para cambiar la visibilidad de las gafas pertenecientes a cada botón y para seleccionar las gafas favoritas.

- *CameraScript*, controla las imágenes de la cámara del dispositivo mostradas en el apartado publicidad.
- *Filtro*, que se comunica simplemente con la clase Catalogo al cambiar la opción seleccionada.
- *PantallaGestor*, encargada de gestionar los toques en la pantalla para controlar la inactividad.
- *RetrovisionMail*, encargada de componer y enviar emails.
- *Gafas*, almacena la información de cada una de las gafas disponibles.
- *ARCoreAugmentedFaceMeshFilterIvan*, copia de la clase con el mismo nombre (sin Ivan) que permite cambiar las gafas seleccionadas y mostrarlas en Realidad Aumentada.

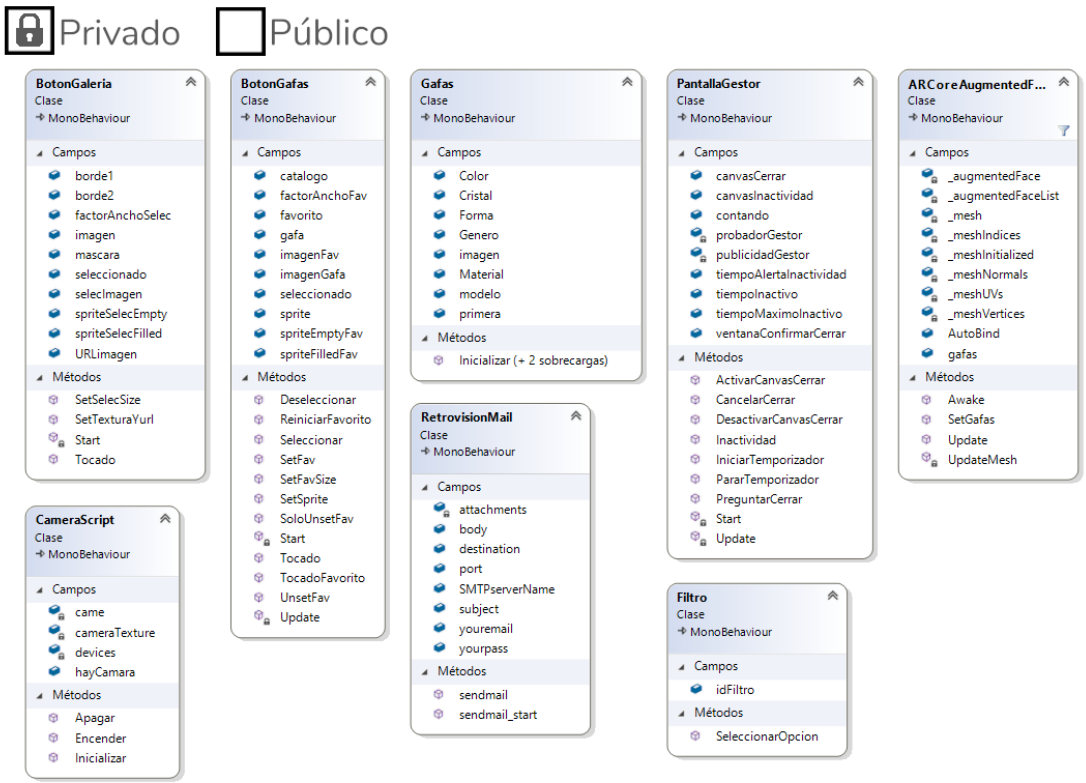


Ilustración 7. Diseño de clases generado con Visual Studio (2).

4.1 Interfaz

Hemos ido creando la interfaz teniendo en mente que el cliente principal será ClaraVisíon, por lo que los colores principales son los mismos que los del logo de esta óptica: naranja y morado. Las imágenes, aunque descargadas la mayoría, todas las hemos editado para que todos los elementos tuvieran una cohesión entre sí.

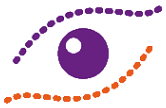


Ilustración 8. Logo de ClaraVisión.

Hemos optado por una interfaz simple y clara con pocos elementos, pero con todas las interacciones que planteamos en un principio, por lo que hemos hecho uso de botones flotantes con dibujos autoexplicativos. Dichos botones están animados gracias a un Asset llamado DOTween [7] que permite animar de forma muy sencilla y con una sola línea de código, pudiendo cambiar el tamaño (*DOScale*), posición (*DOMove*) y rotación (*DORotate*) entre otros, de cualquier objeto con una transición del tipo y duración que le especifiquemos.

Ilustración 9. Botones flotantes y aviso por inactividad.

Aparte de los botones flotantes, podemos observar en la anterior Ilustración 9, que en medio de la pantalla hay un texto, este es el aviso por inactividad que aparece a los 30 segundos desde que se ha detectado el último toque para que se cierre la sesión automáticamente si el último usuario se ha ido sin cerrarla.

4.1.1 Publicidad

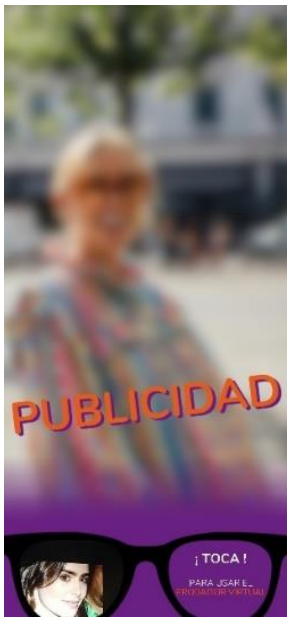


Ilustración 10. Interfaz del apartado publicidad.

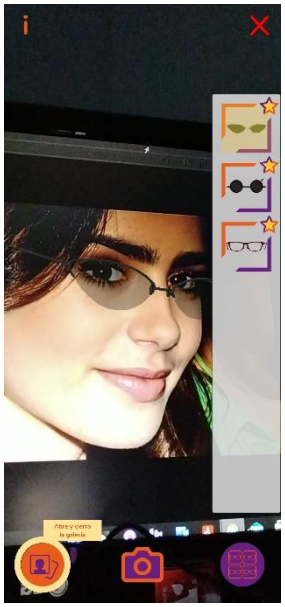
Para el apartado publicidad hemos optado por una interfaz simple con pocos elementos, pero autosuficientes. La mayor parte de la pantalla la ocupan las imágenes publicitarias de la óptica, y la otra unas gafas con un fundido de fondo y con el texto que indica la posibilidad de tocar la pantalla e iniciar el probador en uno de sus cristales y las imágenes de la cámara en tiempo real en el otro.

4.1.2 Probador

En este apartado es donde se encuentra el grueso de elementos de la aplicación, conteniendo el catálogo en la parte inferior, el cual se puede desplegar ocupando toda la pantalla, y los botones información, cerrar sesión, alternar visibilidad del catálogo y abrir/cerrar galería en cada una de las esquinas. Este último solo será visible cuando el usuario se haga una fotografía.

Dentro del catálogo podemos encontrar los filtros disponibles en fila seguidos por todos los botones con las gafas disponibles seleccionables. Estos botones tienen una estrella en la parte superior derecha que permite al usuario seleccionar los modelos que más le gusten para tenerlos a mano más tarde al momento de sacarse las fotografías.

Ilustración 11. Interfaz del apartado probador



Al tocar el botón de bajo a la derecha, el botón que alterna la visibilidad del catálogo, se esconderá el catálogo y aparecerá el botón de hacer fotografías abajo en medio, quedando las imágenes de la cámara al descubierto en toda la pantalla y los botones superpuestos. Si hay favoritos seleccionados, en esta pantalla aparecerá el contenedor de la derecha con los botones de las gafas seleccionadas como favoritas.

Ilustración 12. Interfaz del apartado fotografías.

4.1.3 Galería

En este último apartado volvemos a recurrir a un degradado como el que aparece en la publicidad, solo que esta vez, encima de él se encuentran las instrucciones para enviarse las fotografías por correo y el campo de texto y el botón desde los cuales hacerlo. Se esconden todos los botones excepto el de cerrar sesión y el de abrir/cerrar galería.

Cada una de las imágenes de la galería, que corresponden con las fotografías que se ha sacado el usuario, tiene un círculo amarillo arriba a la derecha que indica que está seleccionada o un círculo vacío que indica que no lo está.

Ilustración 13. Interfaz del apartado galería.

Capítulo 5. Implementación

5.1 Metodología de trabajo y herramientas de desarrollo

Para el desarrollo de este proyecto hemos utilizado GitHub como sistema de control de versiones para mantener en todo momento una copia de seguridad actualizada en la nube y poder retroceder a una versión anterior siempre que sea necesario (el repositorio [8] de este proyecto permanecerá privado hasta que ClaraVisión autorice el acceso público). En cuanto al proyecto, hemos usado el programa Unity en su versión 2019.4.14f1 y hemos programado en lenguaje C#, creando así clases con funciones específicas agrupadas según su objetivo. Para la descarga de imágenes usamos *Google Images* [9] y *Iconfinder* [10], y para su edición usamos el programa gratuito GIMP [11].

Antes de comenzar el proyecto definimos los requisitos principales de la aplicación y diseñamos a grandes rasgos las diferentes pantallas (Capítulo 3) y sus interacciones, aunque a medida que avanzaba el proyecto esto fue variando y mejorando para que se adaptasen a nuestras necesidades y las de la aplicación.

Por otra parte, hemos ido obteniendo información de diferentes usuarios para mejorar la experiencia y el diseño de la interfaz, como es el caso de los botones, que al principio no parecían tocables y los usuarios no interactuaban con ellos. Además, hemos hablado con personal de ópticas (tanto de ClaraVisión como ajeno), y hemos recibido un feedback muy importante de parte de profesionales del campo que nos ha ayudado a adaptar ciertos aspectos del proyecto para que sean más funcionales y autoexplicativos, como por ejemplo que filtros serían los más adecuados, es decir, que cualidades usan los clientes para definir las gafas que estén buscando (forma, color, material...).

5.2 Estructura de la aplicación

El proyecto solo consta de una aplicación, sin base de datos y sin servidor. La estructura de directorios está basada en la que crea Unity por defecto, y hemos ido añadiendo contenido en la carpeta *Assets* así como las carpetas *Fotos*, donde se almacenaban las capturas hechas durante el testeo en el ordenador, y *APK* donde hemos ido almacenando las distintas versiones compiladas de la aplicación en la raíz del proyecto. La estructura es la siguiente, siendo los cuadros amarillos carpetas que hemos creado/modificado y el resto creadas por Unity o paquetes de terceros.

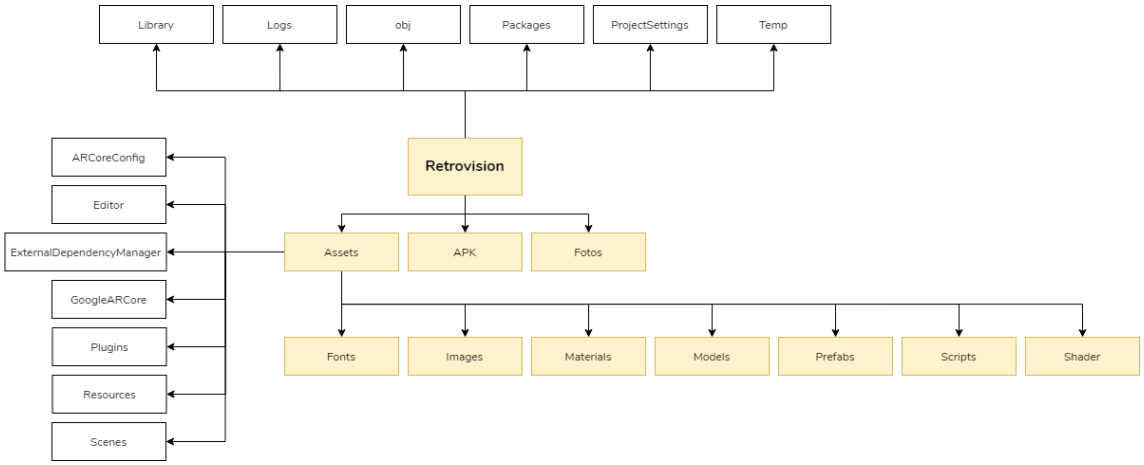


Ilustración 14. Estructura de carpetas.

Dentro de la carpeta de contenido principal, *Assets*, se encuentran varias carpetas, siendo las siguientes las que hemos creado/modificado:

- **Fonts.** Contiene la fuente usada tanto en la aplicación como en esta memoria: Nunito.
- **Images.** Contiene todas las imágenes que se han usado, tanto las publicitarias como las de los botones para seleccionar las gafas.
- **Materials.** Contiene los materiales usados en los elementos de Unity.
- **Models.** Contiene los modelos 3D de gafas usados para el probador virtual.
- **Prefabs.** Contiene los *prefabs*, objetos predefinidos y replicables, que hemos utilizado, como los botones tanto de las gafas como los de la galería.
- **Scripts.** Contiene todos los archivos de código que hemos creado a lo largo del desarrollo del proyecto.
- **Shader.** Contiene los *shaders*, archivos con instrucciones para la tarjeta gráfica, usados para crear los efectos de opacidad de la máscara respecto a las gafas comentado en el punto 4.2.

5.3 Estructura de elementos de Unity

Los elementos en Unity están ubicados en el apartado ***Hierarchy*** o Jerarquía, y en nuestro proyecto es la siguiente:

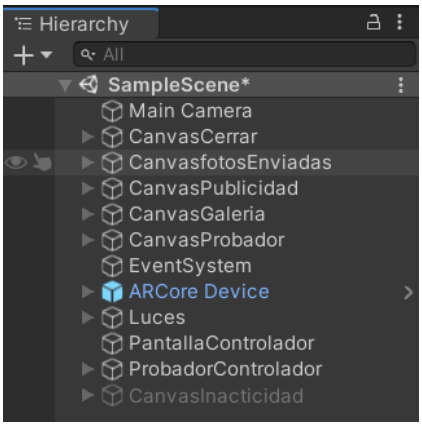


Ilustración 15. Jerarquía de elementos de Unity.

Los elementos son:

- **Main Camera.** Cámara principal de Unity gracias a la cual vemos el apartado Publicidad, ya que posteriormente la principal es la de ARCore (paquete para Unity explicado en el apartado 5.4) para permitir la realidad aumentada.
- **CanvasCerrar.** Contiene los elementos que permiten cerrar la sesión: el botón con forma de X y la pantalla con el texto y los botones para confirmar cerrar la sesión.
- **CanvasfotosEnviadas.** Contiene la imagen final que confirma que las fotografías están siendo enviadas.
- **CanvasPublicidad.** Contiene las imágenes publicitarias y las instrucciones para abrir el probador virtual, así como un elemento con las imágenes a tiempo real de la cámara del dispositivo [12, 13].
- **CanvasGaleria.** Contiene las instrucciones para enviar las imágenes, un campo de texto para escribir el correo electrónico al que se enviarán, el botón ENVIAR y las distintas imágenes que se hayan tomado durante la sesión para seleccionar únicamente las que se deseen enviar.
- **CanvasProbador.** Contiene el catálogo con las diferentes gafas disponibles y los filtros y el apartado para hacer fotografías.
- **EventSystem.** Controlador de eventos de Unity
- **ARCore Device.** Elemento del paquete de ARCore que contiene la cámara que permite ver las imágenes en tiempo real de la cámara del dispositivo en el que esté instalada la aplicación.
- **Luces.** Elementos lumínicos de Unity para que las gafas virtuales se vean adecuadamente como si estuvieran recibiendo luz desde arriba, imitando la luz del sol o de focos.
- **PantallaControlador.** Controla los toques en la pantalla para activar el CanvasInactividad cuando pasen 30 segundos sin ninguna actividad.

- **ProbadorControlador.** Es el elemento en el que se ubican los modelos de las gafas una vez se seleccionan. Contiene a su vez una máscara transparente para opacar las partes de la gafa que no deban verse (explicación más detallada en el siguiente apartado).
- **CanvasInactividad.** Contiene el texto de aviso de que ha pasado mucho tiempo desde el último toque en la pantalla (escribir el correo electrónico es lo único que no detecta como toques en la pantalla ya que lo que se toca es el teclado de Android).

5.4 Algoritmos

De entre todos los algoritmos (métodos de clases) de la aplicación podríamos destacar los que hacen posibles las funciones más importantes:

- **Gafas en Realidad Aumentada.** Para este algoritmo se ha usado **ARCore**, un paquete instalable que permite la implementación de **realidad aumentada** en un proyecto de Unity. Más que algo complicado, instalar ARCore es costoso, ya que requiere de una configuración inicial muy concreta para su correcto funcionamiento [14, 15]. La instalación consta de los siguientes pasos: instalar el SDK de ARCore, instalar el SDK de Android 7.0, instalar determinados paquetes dependiendo de la versión de Unity que utilicemos, importar el SDK de ARCore en el proyecto Android deseado, configurar los parámetros relacionados con el sistema operativo de Android y configurar los archivos *.gradle* dependiendo de la versión de Unity de nuestro proyecto.
  Con este paquete vienen ejemplos para distintos casos de uso de la realidad aumentada, uno de los cuales se llama *Augmented Faces* y es el que hemos utilizado para la detección de caras mediante la cámara frontal del dispositivo. Creamos el script (archivo de código) *ARCoreAugmentedFaceMeshFilterIvan* basado en el script *ARCoreAugmentedFaceMeshFilter* del ejemplo mencionado anteriormente para que se pudiera cambiar el modelo 3D de las gafas superpuestas a nuestro antojo, ya que en todo momento están todas las gafas sobre la cara [16], simplemente activamos la que queremos que se vea y dejamos el resto inactivas.

```
public GameObject gafas;

public void SetGafas(GameObject g)
{
    if (gafas)
    {
        gafas.SetActive(false);
    }
    gafas = g;
}
```

Ilustración 16. Código de ARCoreAugmentedFaceMeshFilterIvan (1).

```
public void Update()
{
    if (AutoBind)
    {
        _augmentedFaceList.Clear();
        Session.GetTrackables<AugmentedFace>(_augmentedFaceList, TrackableQueryFilter.All);
        if (_augmentedFaceList.Count != 0)
        {
            _augmentedFace = _augmentedFaceList[0];
            gafas.SetActive(true);
        }
    }

    if (_augmentedFace == null)
    {
        if (gafas)
        {
            gafas.SetActive(false);
        }
        return;
    }
```

Ilustración 17. Código de ARCoreAugmentedFaceMeshFilterIvan (2).

Conllevó mucho tiempo de configuración que las gafas tuvieran el tamaño adecuado y estuvieran correctamente centradas en la cara del usuario. Además, para crear una sensación más real, antes de colocar las gafas virtuales en la cara detectada, se crea una máscara transparente para dar la sensación de que ciertas partes de la cara del usuario opacan parte de las gafas que deberían dejarse de ver por la perspectiva, como las patillas cuando se gira la cara.

Ilustración 18. Máscara transparente.

- **Catálogo de gafas.** Para la creación del catálogo de gafas disponibles primero se obtienen los modelos, su información y una imagen. Al no disponer de base de datos de la óptica hemos usado modelos de gafas obtenidos de la página web Sketchfab [17] y creados con los objetos básicos de Unity, además de datos ficticios escritos a mano, los cuales sirven para el filtrado de gafas. Estos datos se almacenan en objetos de la clase Gafas y se almacenan en una lista.
  Dependiendo de las opciones seleccionadas en los 5 filtros [18] presentes dentro del catálogo, se creará otra lista con las gafas que coincidan con el filtrado y se crearán los botones correspondientes a cada gafa para que el usuario pueda seleccionarlas y probárselas.

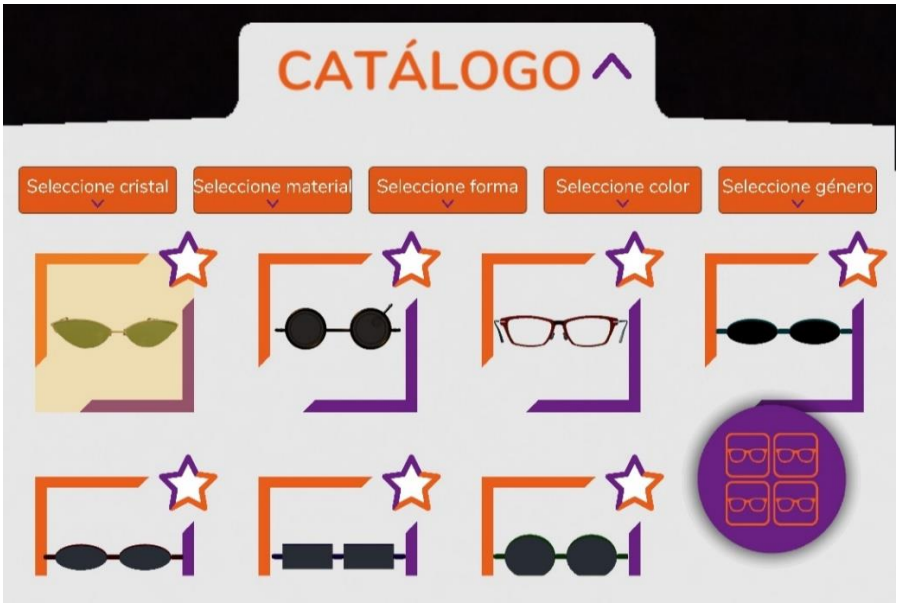


Ilustración 19. Catálogo de gafas de la aplicación comprimido.

Para rellenar el catálogo, dos de las funciones más importantes que se utilizan de la clase Catalogo son *RellenarCatalogo()* y *RellenarBotones()*. La primera rellena una lista de objetos de tipo Gafas con todas las gafas disponibles, mientras que la segunda rellena una parte de la interfaz con botones de las gafas que se deban mostrar, las cuales fueron previamente almacenadas en una lista de gafas que coinciden con las opciones de los filtros desde la función *Filtrar()*.

```
public void RellenarCatalogo()
{
    ObtenerGafas();

    for (int i=0; i<modelos.Count; i++)
    {
        Sprite img = imagenesGafas[0]; // si no existe se pone la foto de la primera gafa
        foreach (Sprite s in imagenesGafas)
        {
            if (modelos[i].name == s.name)
            {
                img = s;
                break;
            }
        }
        Gafas gafa = InstanciarModelo(modelos[i], cristales[i], materiales[i], formas[i], colores[i], generos[i], img);
        if (gafa.primera)
        {
            AsignarGafaVisible(gafa);
        }
        catalogo.Add(gafa);
        gafasAMostrar.Add(gafa);
    }

    if (!contenedorGafas.gafas) // Si ninguna tenia marcado el bool "primera"
    {
        AsignarGafaVisible(catalogo[0]);
    }

    AsignarOpcionesFitros();
}
```

Ilustración 20. Código de Catalogo, función RellenarCatalogo().

```
public void RellenarBotones()
{
    ConfigurarAlturas();

    int columnaActual = 0;
    int filaActual = 0;
    botones.Clear();
    foreach (Gafas g in gafasAMostrar)
    {
        if (columnaActual >= nColumnas)
        {
            columnaActual = 0;
            filaActual++;
        }
        GameObject nuevoBoton = Instantiate(botonGafa);
        nuevoBoton.transform.parent = contenedorBotones.transform;
        nuevoBoton.transform.localScale = new Vector3(1, 1, 1);
        nuevoBoton.GetComponent<RectTransform>().localPosition = new Vector3(columnaActual * anchoColumna + espacioEntreBotones,
            -(filaActual * anchoColumna + espacioEntreBotones), 0);
        nuevoBoton.GetComponent<BotonGafas>().gafa = g;
        nuevoBoton.GetComponent<BotonGafas>().SetSprite();
        nuevoBoton.GetComponent<RectTransform>().sizeDelta = new Vector2(anchoBoton, anchoBoton);

        if (nombresBotonesFav.Find((x) => x == g.name) == g.name)
        {
            nuevoBoton.GetComponent<BotonGafas>().SetFav();
        }

        botones.Add(nuevoBoton);

        if (g.imagen.name == nombreGafaVisible)
        {
            nuevoBoton.GetComponent<BotonGafas>().Seleccionar();
        }

        columnaActual++;
    }
}
```

Ilustración 21. Código de Catalogo, función RellenarBotones().

En esta última función, se usa la variable *nombreGafaVisible* para determinar la gafa que está activa y que su botón aparezca resaltado, gracias a la función *Seleccionar()* de la clase *BotonGafas*, para indicar al usuario que gafa se está probando.

- **Hacer capturas, almacenarlas y enviarlas.** Al darle al botón de la cámara de fotos, desaparecen todos los elementos de la interfaz y comienza una cuenta atrás para que el usuario pueda colocarse como quiera antes de la fotografía. Para la captura hemos utilizado una función de una clase de Unity (*UnityEngine.ScreenCapture.CaptureScreenshot*). Estas se guardan en un directorio que se crea al iniciarse la aplicación en el caso de que no exista [19, 20], y en el caso de que exista se borran las capturas que contenga para respetar la privacidad de los usuarios.
  Una vez hecha la captura, se recupera la imagen y se crea un botón en la galería para que el usuario pueda verla y seleccionarla si quiere enviársela a su correo electrónico, lo cual es posible mediante varias clases propias de Unity (*System.Net*), con las que se puede indicarle una dirección de correo electrónico del emisor y su contraseña para poder enviar correos electrónicos [21], así como la dirección del receptor, el mensaje y los adjuntos, que es la parte importante del algoritmo.
  La captura se realiza desde la clase ProbadorGestor() en la función HacerCaptura(), almacenándola en el dispositivo.

```
public IEnumerator HacerCaptura()
{
    string nuevaFoto = "Foto" + numeroFotos + ".png";
    string pathNuevaFoto = fotosPath + nuevaFoto;
    ScreenCapture.CaptureScreenshot(pathNuevaFoto);

    feedBackFoto.GetComponent<Image>().DOFade(1f, 0.2f).SetLoops(2, LoopType.Yoyo).OnComplete(() => {
        // Si es la primera foto aparece el boton de la galeria
        if (esPrimeraFoto)
        {
            botonGaleria.transform.localScale = new Vector3(0, 0, 0);
            botonGaleria.GetComponent<Button>().enabled = false;
            botonGaleria.transform.DOScale(1.2f, 0.3f).OnComplete(() => {
                botonGaleria.transform.DOScale(1f, 0.3f).OnComplete(() => {
                    botonGaleria.GetComponent<Button>().enabled = true;

                    AnimacionBotonAparecer(botonGaleria);
                    StartCoroutine(MostrarWizard(botonGaleria));
                });
            });
            esPrimeraFoto = false;
        }
        if (contenedorFavoritosOn)
        {
            contenedorFavoritos.SetActive(true);
        }
        botonModos.SetActive(true);
        botonFotos.SetActive(true);
        contenedorInfo.SetActive(true);
    });

    yield return new WaitForSeconds(1f);
    botonGaleria.SetActive(true);

    StartCoroutine(canvasGaleria.GetComponent<GaleriaGestor>().AddFoto(nuevaFoto, Screen.width, Screen.height));

    FindObjectOfType<PantallaGestor>().ActivarCanvasCerrar();
}
```

Ilustración 22. Código de ProbadorGestor, función HacerCaptura().

Después, se recupera la imagen y se muestra en el apartado galería de la aplicación en forma de botón desde la función *AddFoto()* de la clase *GaleriaGestor*.

```
public IEnumerator AddFoto(string file, int ancho, int alto)
{
    if (FindObjectOfType<PublicidadGestor>().inicializarCamara)
    {
        files = Directory.GetFiles(fotosPath, "*.png");
        string archivo = "file://" + Path.Combine(fotosPath, file);
        // Crear boton con la foto
        Texture2D tex = new Texture2D(ancho, alto, TextureFormat.DXT5, false);
        using (WWW www = new WWW(archivo))
        {
            yield return www;
            www.LoadImageIntoTexture(tex);

            // Crear y configurar boton
            if (columnaActual >= nColumnas)
            {
                columnaActual = 0;
                filaActual++;
            }
            galeria.GetComponent<RectTransform>().sizeDelta = new Vector2(galeria.GetComponent<RectTransform>().sizeDelta[0], anchoColumna * (filaActual+1));
            GameObject nuevoBoton = Instantiate(botonFotoPrefab);
            nuevoBoton.transform.parent = galeria.transform;
            nuevoBoton.transform.localScale = new Vector3(1, 1, 1);
            float nuevoAlto = (anchoBoton * alto) / ancho;
            nuevoBoton.GetComponent<RectTransform>().localPosition = new Vector3(columnaActual * anchoColumna + espacioEntreBotones,
                -((filaActual * nuevoAlto/*anchoColumna*/) + ((filaActual+1) * espacioEntreBotones)), 0);
            nuevoBoton.GetComponent<RectTransform>().sizeDelta = new Vector2(anchoBoton, nuevoAlto);

            nuevoBoton.GetComponent<BotonGaleria>().SetTexturaYurl(tex, ancho, alto, Path.Combine(fotosPath, file));

            columnaActual++;
        }
    }
```

Ilustración 23. Código de GaleriaGestor, parte de la función AddFoto() [22].

5.5 Problemas de implementación

5.5.1 Resueltos

- **Instalación y configuración de ARCore.** Este es un proceso que implica una gran cantidad de pasos que no deben saltarse y que hay que realizar con cuidado. Recomendamos que se sigan las instrucciones que ofrece Google [14, 15] directamente, ya que el software les pertenece. Además, es necesario preparar el SDK de Android en Unity añadiendo código en el *graddle* y configurando ciertos parámetros de Unity. Por último, se debe de instalar del *Package Manager* de Unity el paquete *XR Legacy Input Helpers* y configurar el API mínimo de Android en Unity al Android 7.
- **Encontrar la ubicación de las capturas de pantalla.** Tras hacer una captura de pantalla fue necesario encontrar su ubicación para mostrar las imágenes en la galería. La parte confusa fue que pese a ser el mismo proyecto, la ubicación de las fotografías en el ordenador y en un móvil Android no es igual, por lo que fue necesaria una búsqueda intensiva en Internet y un periodo de pruebas más largo de lo que pensamos. Al final encontramos que para acceder a la carpeta del proyecto en ordenador hay que usar *Directory.GetCurrentDirectory* mientras que en un dispositivo Android se usa *Application.persistentDataPath*. Con esto, seguido de la

carpeta en la que se haya guardado la imagen, si este es el caso, accederíamos a la ubicación de la carpeta, mientras que para acceder al archivo como tal, es necesario añadir el prefijo “file://” y al final el nombre del archivo, quedando algo del tipo “file://” + Aplication. *persistentDataPath* + “/Fotos/” + nombre del archivo.

- **Configurar botones en forma de tabla.** Esta funcionalidad está presente en las clases *Catalogo* y *GaleriaGestor*, en la primera en la lista de botones de las gafas y en la segunda en la lista de botones con las fotografías tomadas. De primeras, lograr que una serie de botones se colocasen en forma de tabla (distribuidos en filas y columnas) respetando los espacios entre ellos, no parecía una tarea compleja, pero tuvimos que usar por primera vez la clase *RectTransform* de Unity, encargada de posicionar elementos en un canvas, y sus métodos no son muy intuitivos. Pero después de informarnos y hacer pruebas, logramos crear tablas proceduralmente dependiendo de la cantidad de elementos que haya que se adaptan al tamaño de la pantalla del dispositivo en el que se halle.

5.5.2 No resueltos

- **Varias gafas a la vez.** Nuestra idea era que varios usuarios a la vez pudiesen probarse gafas, pero esto actualmente (septiembre 2021) no es posible. Buscamos en varios sitios de internet [23, 24, 25] y en todos decían que solo se podían colocar elementos en una cara a la vez. Nos metimos en el código de ARCore que hace esto posible, y aunque encontramos que sí que detecta varias caras a la vez, no podíamos superponer elementos en realidad aumentada a todas ellas. Por ello, optamos por dejar una cara sólo, explicándole al usuario con el botón información que para cambiar la cara detectada tienen que acercarse a la cámara de la pantalla digital, siendo la deseada la única cara visible.

Capítulo 6. Manuales

6.1 Manual de instalación

Para la instalación del proyecto como tal, sólo es necesario descargar la carpeta del proyecto tanto desde GitHub como desde cualquier otro medio posible y abrir dicha carpeta con Unity. Pero como el proyecto va a permanecer privado por el momento no habrá acceso público al mismo.

En cuanto a la instalación de la aplicación, el único archivo necesario será un *.apk*, el cual se entregará personalmente y bastará con abrir el archivo e instalarlo en un dispositivo Android para su funcionamiento.

6.2 Guía de uso

1. Al iniciar la aplicación podemos ver publicidad que pondría la óptica a su gusto y un apartado que indica que mediante un toque se puede iniciar el probador.
2. Al tocar la pantalla aparecerá una pantalla de carga
3. Seguidamente se abrirá el apartado probador, donde se encuentra el catálogo con todas las gafas disponibles y los filtros. Cada botón correspondiente a cada gafa en el catálogo dispondrá de un botón con forma de estrella que sirve para seleccionar favoritas y disponer de ellas en el apartado de las fotografías (**5**).
4. Al lado del título “CATÁLOGO” hay una flecha que despliega el catálogo para que ocupe toda la pantalla y se pueda navegar por él más fácilmente.
5. Si se clica el botón de bajo a la derecha dentro del catálogo (**3**) se alterna la visibilidad del catálogo con el botón para hacer fotografías.
6. En cualquier momento será posible cerrar la sesión y borrar tanto los favoritos como las fotografías que se hayan hecho con el botón con forma de X situado arriba a la derecha.
7. Una vez tocado el botón con forma de cámara, empezará una cuenta atrás para que el usuario se prepare para la fotografía y esta se hará a los 3 segundos.
8. A continuación, aparecerá otro botón abajo a la izquierda que nos permitirá entrar en la galería.
9. Una vez en la galería, se podrá volver a atrás con el mismo botón con el que se ha entrado (**8**). En el único campo de texto que hay se podrá escribir un correo electrónico, seleccionar las imágenes que nos hayan gustado y

tocar el botón ENVIAR para que dichas imágenes lleguen a la dirección que hemos escrito previamente.

10. Al completar los pasos anteriores (**9**), una pantalla de carga nos indicará que las imágenes se están enviando y al acabar nos llevará otra vez al apartado probador (**8**).

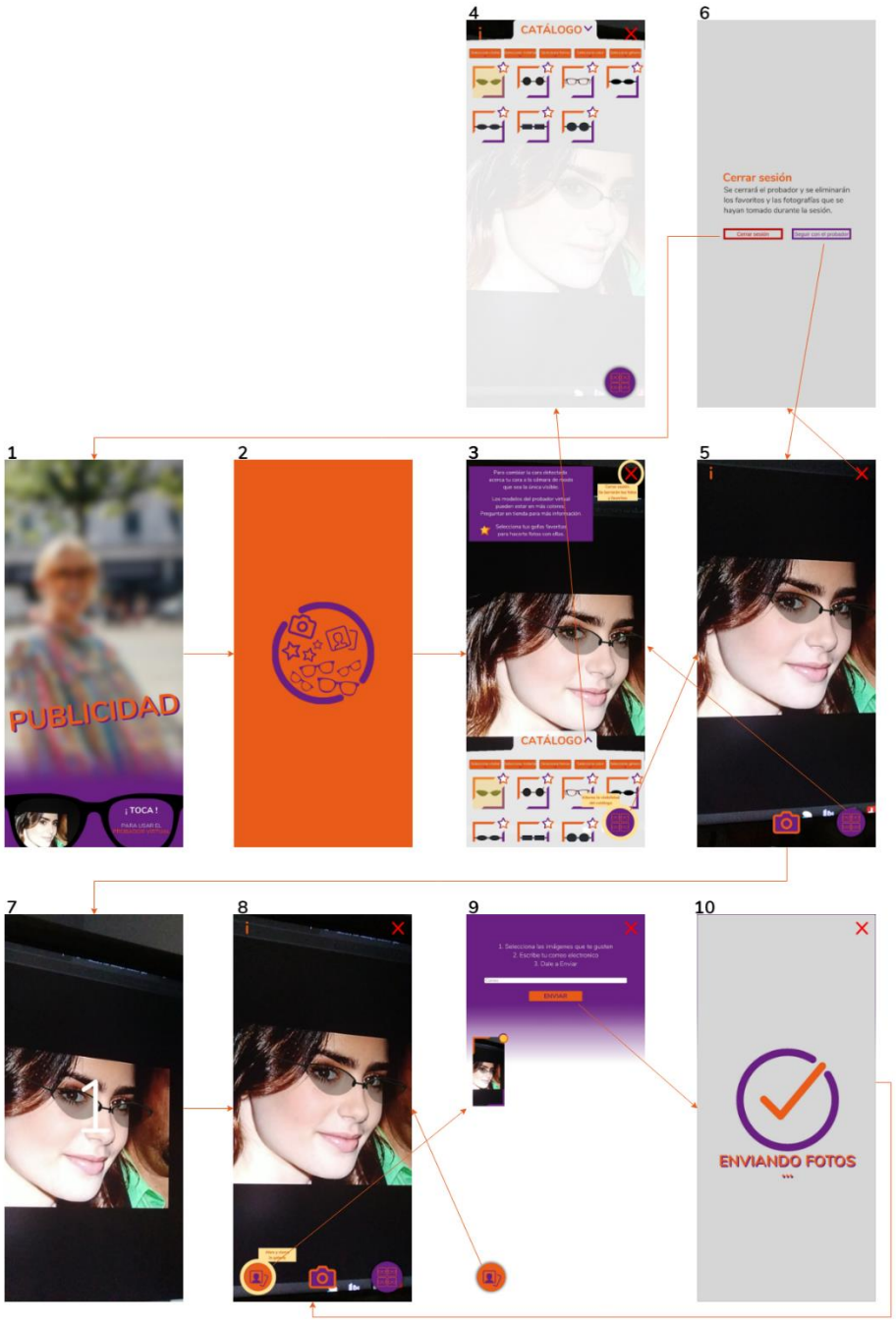


Ilustración 24. Pantallas de interacción.

Capítulo 7. Evaluación

En cuanto a la validación del código, usamos una herramienta de Unity que permite ejecutar la aplicación en una pantalla virtual de diferentes resoluciones para comprobar que la aplicación es ***responsive***, es decir, que las proporciones, tamaños y posiciones de todos los elementos en pantallas distintas se mantienen. El único inconveniente es que el código de ARCore, que hace posible la realidad aumentada, parece no funcionar correctamente en el ordenador, por lo que tuvimos que hacer pruebas constantemente en un móvil sin poder tener acceso a la consola para comprobar si había errores. Aunque no supone un gran problema en cuanto a si es *responsive*, ya que las imágenes en tiempo real que vemos de la cámara del dispositivo están configuradas para que ocupen totalmente la pantalla independientemente de su tamaño, y las cámaras suelen adaptar el tamaño de sus imágenes al tamaño de la pantalla del dispositivo.

Como ya se ha comentado anteriormente, para la evaluación objetiva de la aplicación, durante el desarrollo del proyecto la hemos dado a probar a distintos tipos de usuarios de distintas edades, tanto jóvenes de 20 años como adultos de más de 50, y ocupaciones, desde estudiantes universitarios a personas con pocos estudios, con el objetivo de crear una aplicación que sea fácilmente entendible y usable por cualquier tipo de persona. De las personas más mayores descubrimos que ciertas interacciones que la gente más joven puede dar por sentadas no son tan intuitivas como pensábamos. De profesionales del sector de las ópticas obtuvimos importante información sobre qué tipo de filtros son los que más usa la gente cuando va a elegir gafas. De la gente más joven aprendimos que a veces menos es más, y no hace falta que haya una gran cantidad de interacciones para que la aplicación deje un buen sabor de boca en los usuarios.

Una vez finalizada la aplicación realizamos un estudio para comprobar las tareas, listadas en la ilustración 26, que realizan los usuarios (7 de alrededor de 20 años y 2 de entre 40 y 60 años) entre todas las posibles y los tiempos, además de una serie de preguntas posteriores al testeo para identificar los puntos fuertes y los débiles. Más del 50% de los usuarios que accedieron al test fueron estudiantes de entre 20 y 25 años, aunque también la testearon personas de más de 40 años, obteniendo resultados similares a los jóvenes.

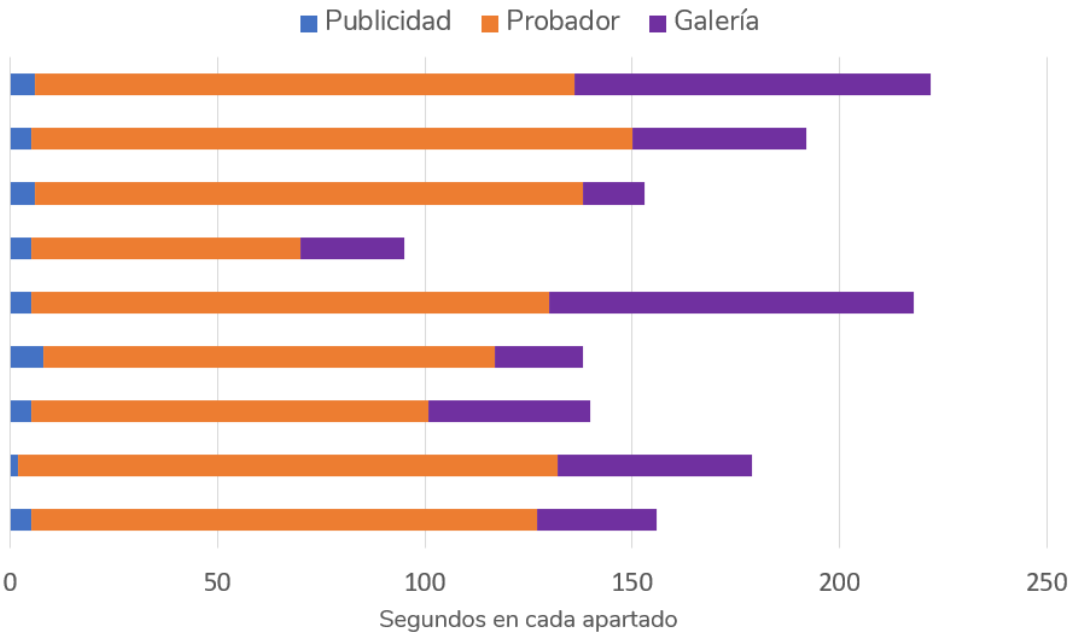


Ilustración 25. Gráfico de tiempo que pasan los usuarios en cada apartado.

Cada una de las barras del gráfico representa los tiempos en segundos que ha estado cada usuario en cada uno de los 3 apartados, siendo la parte de color azul el apartado publicidad, el naranja el probador y el morado la galería. Aunque hemos obtenido los datos de usuarios de diferentes edades no hemos encontrado ninguna diferencia notoria en el uso de los apartados respecto al tiempo.

Hay un patrón que es que no pasan mucho tiempo en publicidad, y esto es obvio ya que solo hay imágenes de prueba y por tanto de poco interés para los usuarios. En cuanto al probador, este es el apartado en el que más tiempo pasan en todos los casos, cosa bastante normal porqué es donde se encuentran la mayoría de las interacciones, permaneciendo entre 65 y 132 segundos. Por último, en la galería parece ser donde vas variación hay, oscilando entre los 15 y los 88 segundos.

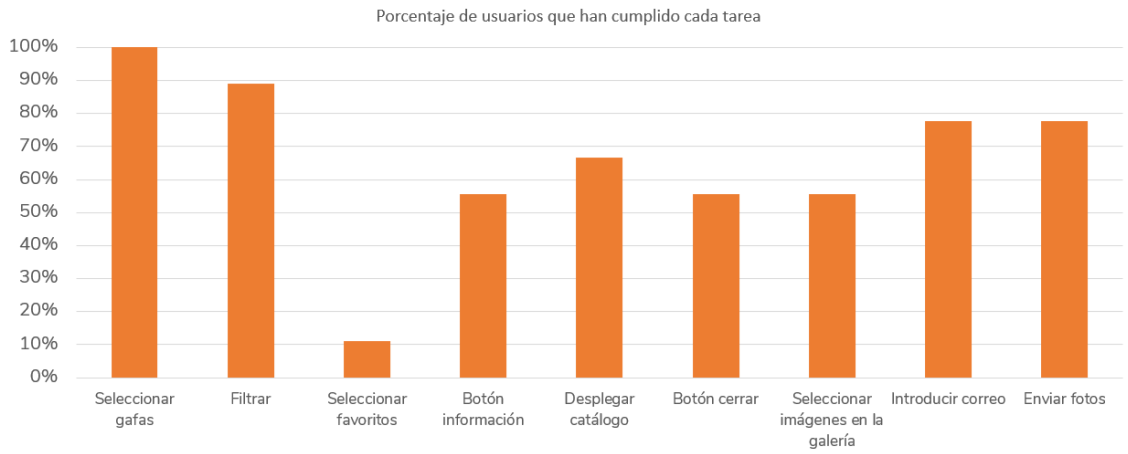


Ilustración 26. Gráfico de porcentaje de usuarios que realizaron cada tarea.

El anterior es el gráfico de tareas, que muestra mediante barras el porcentaje de usuarios que han realizado cada tarea, por supuesto sin nuestra intervención. Podríamos separarlas por 3 grupos:

- Las más realizadas (más del 75%) son seleccionar gafas, con un 100% de uso, filtrar, introducir correo y enviar fotos. Son de las funciones más importantes que definimos en los requisitos en el Capítulo 3, lo que significa que hemos implementado correctamente estas interacciones.
- Las realizadas normalmente (entre el 75% y el 40%) son tocar el botón información, desplegar el catálogo, tocar el botón cerrar y seleccionar las imágenes de la galería. Esto tiene sentido debido a que estas son funciones opcionales para apoyar al usuario. Desplegar el catálogo deducimos que no ha sido realizado por tantos usuarios porque durante el desarrollo del proyecto y durante su testeo dispusimos de pocas gafas, por tanto, los usuarios no se verían en la necesidad de desplegar el catálogo para ver mejor todos los modelos. Y en cuanto a seleccionar las imágenes de la galería puede tener que ver también con lo que acabamos de mencionar, al disponer de pocas gafas, todos los usuarios que se hicieron fotos solo hicieron una, por lo que no se veían con la necesidad de seleccionar fotos de la galería para enviarlas porque solo había una.
- Las poco realizadas (menos del 40%). Sólo hay una tarea en este grupo, y es la de seleccionar favoritos, realizada únicamente por un usuario. De nuevo puede ser debido a la poca cantidad de modelos disponibles en el catálogo, de manera que era innecesario seleccionar los favoritos de entre tan pocos. Aunque, por otra parte, también es posible que los botones de estrella de cada gafa, pensados para seleccionar favoritos, no sean intuitivos y quizá son vistos como el diseño del botón, es decir, como un

elemento no interactivo. Si fuera este el caso habría que rediseñar esta función o hacerla más llamativa si queremos que sea más usable, aunque no deja de ser una función opcional.

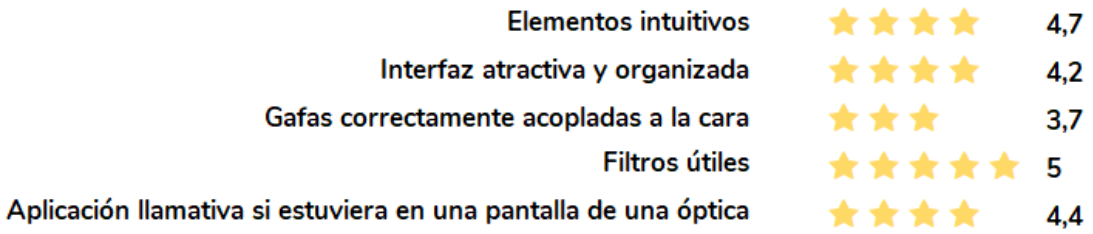


Ilustración 27. Valoración de los usuarios de ciertos aspectos de la aplicación.

Por último, hemos preguntado por ciertos aspectos de la aplicación que nos parecen importantes para comprobar si hemos conseguido cumplir con nuestros requisitos iniciales.

A todos los usuarios les ha parecido que los filtros han sido útiles, tanto por las categorías a filtrar como por su funcionalidad, puntuándolos con un 5 de 5. Otros 3 aspectos han obtenido una nota de más de 4 sobre 5 de media, cosa que indica que, aunque no son perfectos, están bien implementados, siendo estos elementos intuitivos, interfaz atractiva y organizada y cuan llamativa les parece la aplicación si la viesen en una pantalla digital en una óptica como para acercarse e interactuar con ella. Finalmente, ha habido un aspecto con una media inferior al 4, que es si las gafas se acoplaban correctamente a la cara. Esto tiene su razón de ser, y es que las gafas que hemos descargado son de diferentes autores, por lo que hemos tenido que acoplarlas nosotros manualmente una a una, quedando ligeramente a distintas alturas. Pero cuando dispongamos de las gafas en 3D de la óptica, las cuales deberían de tener el mismo formato, estando todas centradas, de tamaños similares y con las patillas abiertas de la misma forma; con colocar una perfectamente sobre la cara será suficiente para usar esa configuración para todas las demás.

Capítulo 8. Modelo comercial

Nuestro plan de negocio es vender la aplicación a ClaraVisión manteniendo los derechos de explotación y distribución para una posible futura expansión.

Hablando con el Gerente de ClaraVisión planteamos la idea de, en el caso de que la aplicación tenga una buena recepción por parte de los usuarios, presentarles la aplicación a más ópticas que puedan llegar a comprarla, siendo esta la principal fuente de ingresos de este proyecto.

Capítulo 9. Conclusiones

Hemos desarrollado una aplicación destinada a implementarse en pantallas digitales en ópticas con la finalidad de que los usuarios que interactúen con ella puedan probarse gafas de manera virtual sin necesidad de que la óptica disponga de los modelos físicos en ese momento. Siendo así una forma más higiénica de decidir qué gafas comprar, algo de gran importancia en tiempos de pandemia.

El proyecto se finalizó con éxito consiguiendo todos los hitos propuestos desde el principio: al iniciar la aplicación se muestra publicidad de la óptica y la posibilidad de interactuar con la pantalla, después se pueden filtrar y elegir gafas entre todas las del catálogo para probárselas mediante realidad aumentada, y finalmente se pueden hacer fotografías, seleccionar las que más gusten y enviarlas al correo electrónico que se especifique durante la sesión.

Hemos conseguido crear una interfaz de usuario simple e intuitiva testada por diferentes usuarios que han disfrutado de la experiencia, ayudando a su vez al desarrollo y mejora de distintos aspectos de la aplicación.

En cuanto a la relación de este proyecto con los estudios previos, en uno de los años del Grado en Tecnologías Interactivas hicimos varios proyectos en Unity de realidad aumentada en las asignaturas “Tecnologías realidad virtual/realidad aumentada” y “Proyecto entornos interactivos avanzados”, aprendiendo las tecnologías más usadas en este campo. En cuanto a la programación, tanto los conocimientos básicos como los más específicos, fueron aprendidos en las mencionadas anteriormente y en “Programación 1”, “Programación 2” y “Proyecto Aplicaciones Multimedia Interactivas. Videojuegos”, conociendo en esta última cómo funciona el programa Unity. Y en cuanto a la interfaz, aparte de las 2 primeras asignaturas mencionadas previamente, también aprendimos las bases en “Diseño de interfaces y experiencia de usuario”, descubriendo la importancia de la distribución, tamaños y colores de elementos en una aplicación.

Además, en las prácticas en empresa también trabajamos con Unity, haciendo el frontend de dos experiencias en realidad virtual.

Los requerimientos propuestos en el Capítulo 3 se han cumplido de la siguiente forma:

- **RF1.** Mediante ARCore y los modelos encontrados tanto en internet como los que hemos creado nosotros, los usuarios se pueden ver con gafas superpuestas en realidad aumentada.

- **RF2.** Mediante un botón con una cuenta atrás, los usuarios pueden hacerse fotografías.
- **RF3.** Mediante el apartado galería, los usuarios pueden seleccionar las fotos que le interesen y enviarlas a su correo electrónico.
- **RF4.** Mediante el apartado probador, los usuarios pueden seleccionar las gafas que quieran de entre todas las disponibles, además de filtrarlas, para probárselas en realidad aumentada.
- **RF5.** Mediante la estrella situada arriba a la derecha de cada botón del catálogo, los usuarios pueden seleccionar sus gafas favoritas para disponer de ellas en el apartado fotografías.
- **RF6.** Mediante una especie de carrusel situado en el apartado fotografías, los usuarios pueden ver publicidad de la óptica (aunque en la versión actual son imágenes de prueba) sin interaccionar con la pantalla.
- **RNF1.** Gracias a la disposición y usabilidad de todos los elementos, los usuarios puntuaron cuan intuitiva es la aplicación con un 4.2 sobre 5, además de realizar la mayoría de acciones importantes durante el testeo.

9.1 Trabajo futuro

- Cuando la aplicación esté en marcha en la óptica sería muy costoso estar metiendo a mano las imágenes publicitarias y los nuevos modelos de gafas, por lo que lo primero que tendríamos que hacer sería programar algoritmos que se conectasen con la base de datos de la óptica para obtener esta información de manera automática cada vez que se iniciase la aplicación. No nos propusimos hacer una base de datos de prueba porque priorizamos tener una demo con buen diseño, funcional y que gustase a los usuarios y así tener más posibilidades de que la óptica se interesase por seguir el proyecto y entonces enfocarnos más en el backend.
- Uno de los problemas que no se han podido corregir es la detección de varias caras simultáneamente para que varios usuarios puedan probarse las mismas gafas a la vez. Intentamos cambiar el código del script (archivo de código) de ARCore que hace posible la detección de caras, pero no conseguimos que funcionase correctamente. También buscamos por Internet, pero todos los comentarios decían que no se podía hacer con la versión actual. Una posible solución sería buscar un paquete distinto a ARCore en el que esto se pudiera hacer, pero quizá sea difícil encontrar uno tan preciso, ya que ARCore es de Google y funciona bastante bien.

- Si se sale y se vuelve a entrar en la aplicación, la cámara se ve en negro, por lo que hay que reiniciar la aplicación. Como este problema no tiene ninguna importancia en el escaparate ya que los usuarios no podrán salir de la aplicación en ningún momento, no hemos buscado ninguna solución y nos hemos enfocado en avanzar adecuadamente. Pero para un futuro sí que sería buena idea solucionarlo.

Bibliografía

Daniel Pareja Valle, 2017-2018 Diseño e implementación de una aplicación móvil docente.

Pedro Pérez Palazón, 2016 Realidad Aumentada, orientada a la gestión de servicios e información en un Smart Place.

1. Página web de Afflelou (visitada el 7 de junio de 2021)
   https://www.afflelou.es/

2. Página web de VisionLab (visitada el 7 de junio de 2021)
   https://www.visionlab.es/

3. Nice2SeeU: proyecto de TimiakTech (visitada el 7 de junio de 2021)
   https://www.nice2seeu.com/

4. Unity: motor de desarrollo de aplicaciones y videojuegos
   https://unity.com/es

5. DeVuego: base de datos de la industria española del videojuego, uso de motores de juego en 2019 (visitada el 10 de junio de 2021)
   https://www.devuego.es/bd/estadisticas/distribucion-motores/?ano=2019

6. Visual Studio: editor de código
   https://visualstudio.microsoft.com/es/

7. DOTween: plugin de animaciones para Unity
   https://www.programmersought.com/article/42621159944

8. Repositorio de GitHub del proyecto (puede ser privado)
   https://github.com/IviRome/Retrovision1

9. Google Imágenes (visitada en julio y agosto de 2021)
   https://www.google.com/imghp?hl=en

10. Iconfinder: página web con iconos descargables y algunos gratuitos (visitada en julio y agosto de 2021)
    https://www.iconfinder.com.

11. GIMP: programa de edición de imágenes gratuito
https://www.gimp.org/

12. YouTube, usar las imágenes de la cámara de un dispositivo como textura de un objeto de Unity (visitadas el 21 de junio de 2021)
https://www.youtube.com/watch?v=tEXEe_ikfLw&ab_channel=LearnEverythingFast

13. Unity Answers, acceder a la cámara frontal de un dispositivo (visitadas el 21 de junio de 2021)
https://answers.unity.com/questions/633097/cant-access-front-device-camera.html

14. Google Developers, descargar e instalar el paquete de ARCore (visitada el 10 de junio de 2021)
https://developers.google.com/ar/develop/unity/quickstart-android

15. Google Developers, preparar el SDK de Android para usar ARCore en Unity 2019.4 (visitada el 10 de junio de 2021)
https://developers.google.com/ar/develop/unity/android-11-build

16. YouTube, ubicar modelos 3D sobre caras con realidad aumentada (visitada el 10 de junio de 2021)
https://www.youtube.com/watch?v=X6xqmsivAHU&ab_channel=FuturoAumentadoFuturoAumentado

17. Sketchfab: página con contenido de modelos 3D, algunos gratuitos (visitada en julio y agosto de 2021)
https://sketchfab.com/3d-models

18. YouTube, filtros del catálogo (visitada el 3 de julio de 2021)
https://www.youtube.com/watch?v=FteXwEdED0Y&ab_channel=MohammadFaizanKhan

19. Unity Answers, crear un directorio (visitada el 7 de julio del 2021)
https://answers.unity.com/questions/528641/how-do-you-create-a-folder-in-c.html

20. Unity Answers, comprobar si existe un directorio (visitada el 7 de julio del 2021)
https://answers.unity.com/questions/529327/how-do-you-check-if-a-directory-exists-c.html

21. YouTube, enviar un correo electrónico en Unity (visitada el 3 de agosto del 2021)
https://www.youtube.com/watch?v=qRytzVzruCQ&ab_channel=Gamad

22. Unity Answers, acceder a imágenes del dispositivo sabiendo su ubicación (visitada el 7 de julio del 2021)
https://answers.unity.com/questions/25271/how-to-load-images-from-given-folder.html

23. Documentación de Unity (visitada en junio, julio y agosto de 2021)
https://docs.unity3d.com

24. Unity Answers: preguntas sobre Unity (visitada en junio, julio y agosto de 2021)
https://answers.unity.com

25. Stack Overflow: foro de programadores (visitada en junio, julio y agosto de 2021)
https://stackoverflow.com