

2022

CIFP Virge de Gracia

Desarrollo de Aplicaciones
Multiplataforma

BUSCO TINTA

App BUSCADOR DE TATUAORES

José María Ruiz León

1. PRESENTACIÓN DEL PROYECTO	3
1. 1. Explicación resumida.	3
1. 2. Estudio de mercado	4
1. 3. Valor del producto	7
2. ANÁLISIS DE LA SOLUCIÓN.....	7
2. 1. Análisis y especificación de requisitos	7
2. 2. Análisis de escenarios (casos de uso)	10
3. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES	21
3. 1. Planificación y organización de tareas.....	21
3. 2. Estimación de costes y recursos: hardware, software y humanos.....	25
3. 3. Herramientas usadas	27
3. 4. Gestión de riesgos.....	28
4. DISEÑO DE LA SOLUCIÓN.....	29
4. 1. Diseño de la interfaz de usuario y prototipos.....	29
4. 2. Diagrama de clases	36
4. 2. Diagrama de persistencia de la información	40
4. 4. Arquitectura del sistema.....	42
5. IMPLEMENTACIÓN DE LA SOLUCIÓN	43
5. 1. Análisis tecnológico.....	43
5. 2. Elementos a implementar.....	45
6. PRUEBAS	60
6. 1. Realización de pruebas	60
BIBLIOGRAFIA	65

1. PRESENTACIÓN DEL PROYECTO

1. 1. Explicación resumida.

OBJETIVO:

Realizar una aplicación de Android que consista en una red social enfocada a los tatuajes, para conectar tatuadores y clientes.

RESUMEN:

La intención principal de la aplicación es la doble:

- Ofrecer a tatuadores y artistas un espacio donde darle más visibilidad a su trabajo, darse a conocer y poder atraer nuevos clientes.
- Facilitar a posibles clientes una novedosa herramienta de búsqueda de tatuadores, basada en filtros o en requisitos que este especifique.

Esta aplicación consta de dos tipos de perfiles de usuario: **Tatuador y cliente**.

Habrá un tercer tipo de usuario **Administrador**, el cual está pensado para gestionar ciertos aspectos de la aplicación, en especial del tratamiento de datos de los otros dos tipos de usuarios.

Se pretende dar visibilidad al usuario que se registre como **tatuador**. Tendrá una página principal de perfil, donde aparecerá su nombre, una foto de perfil o logo personal, ubicación geográfica de su estudio de tatuajes, margen de precios según tamaños, datos de contacto (opcionales) y un muro de publicaciones. Cada publicación es una foto de un tatuaje realizado, además consta de un pie de foto y la posibilidad de poner *etiquetas de estilos** a esa foto (las etiquetas de estilos son un punto muy importante para el motor de búsqueda).

El usuario que se registre como **cliente**, tendrá un perfil más enfocado a la búsqueda de tatuadores. Tendrá una página de perfil más básica, que el tatuador, ya que el objetivo principal es que haya un buscador que ofrezca diferentes opciones de búsqueda pudiendo aplicarle diversos filtros en caso de exigir ciertos requisitos. El primero (y que mayor atractivo aporta a la aplicación) es la posibilidad de filtrar la búsqueda por proximidad a una ubicación geográfica. El segundo filtro más importante es el de margen de precios según tamaños. Otro filtro que se puede aplicar al buscador es la posibilidad de buscar por etiquetas de estilos. Y por supuesto, también está la posibilidad de buscar por el nombre.

Un ejemplo de una búsqueda sería aplicar una proximidad de **50km** desde mi ubicación, aplicar un margen de precios según tamaños de **50€ en 10cmX10cm** y además aplicaremos tres etiquetas de estilos **#realismo #animal #fantasia**. El resultado de la búsqueda se puede mostrar de dos formatos de vistas posibles **Listado o Mapa** (pudiendo alternar el usuario entre una vista u otra).

En el formato de vista Mapa, se verá una ventana de googleMaps con chinchetas/marcadores de todos los estudios de tatuaje que cumplen los requisitos de la búsqueda. Y en caso de pinchar en un marcador, nos llevaría al perfil del tatuador, donde podemos ver su mural de fotos, contactar con él para concertar una cita, etc.

En el formato de vista Listado, se mostrarán en una lista deslizable el nombre y foto de perfil de todos los tatuadores que cumplen con esos requisitos de búsqueda. Al hacer click en un tatuador de la lista, nos llevaría directamente a su perfil, donde podemos ver su mural de fotos, contactar con él para concertar una cita, etc. El formato listado también cuenta con posibilidades de aplicar criterios de ordenación (Ordenar alfabéticamente por nombre, ordenar de menor a mayor distancia de proximidad geográfica, ordenar de menor a mayor precio X tamaño, etc.).

OTRAS FUNCIONALIDADES

La app también ofrece la posibilidad de contactar con el tatuador. El tatuador puede escoger si poner datos de contacto en su perfil, como su número de teléfono o su correo. Pero esta opción es opcional. Por tanto, la forma de poner en contacto a potenciales clientes con tatuadores es incorporando la posibilidad de abrir un **chat de conversación** entre ambos.

La posibilidad de **dar “Like” o “Me gusta” a las publicaciones** en los murales de cada tatuador.

La posibilidad de **guardar en favoritos** los perfiles de tatuadores que más le interese al usuario.

El **perfil del cliente** contará también con foto de perfil y nombre, además de un espacio de “favoritos”.

POSIBLES IMPLEMENTACIONES (Se estudia su incorporación a largo plazo)

Investigación e implementación de **Lucene** (indexador de contenidos) como herramienta sustitutiva o complementaria a la base de datos. Esta idea podría dar una aportar un gran potencial al sistema de búsqueda.

Possible implementación de otra opción de búsqueda para el usuario cliente, pudiendo hacer una foto y que el sistema reconozca si hay un tatuaje en la foto, y si ese tatuaje es de un estilo ya registrado. Consiste en implementar una **red neuronal de reconocimiento de imágenes basada en MachineLearning**. Este sistema de reconocimiento se le pasa una foto o captura de pantalla y reconoce los posibles estilos de tatuaje que pueda haber en la imagen. (Sugerencia de Juan Ramón Vallejo, mi coordinador de prácticas). La complejidad de intentar implementar esto, es que este tipo de sistemas hay que “entrenarlos” pasándole una gran cantidad de imágenes de cada estilo, para que una vez entrenado sepa procesar y reconocer y clasificar las imágenes. (Esta implementación dependerá del tiempo, ya que requiere mucho tiempo para “entrenar” al sistema).

Migración a iOS. Se plantea poder utilizar también esta aplicación en dispositivos iOS mediante un sistema emulador del código fuente.

1. 2. Estudio de mercado

Un estudio de mercado consiste en una investigación con el fin de saber si una iniciativa emprendedora puede resultar viable económicamente o no. Esta investigación siempre se realiza de forma previa al desarrollo de cualquier iniciativa emprendedora o actividad económica.

El objetivo es conocer el impacto real que pueda tener ese proyecto en el mercado, la posible competencia, la aceptación que pueda llegar a tener por parte de los clientes y la viabilidad en la implantación de la idea.

Estudio de mercado:

- Información del sector.
- Público objetivo
- Competencia en el mercado
- Análisis DAFO

INFORMACIÓN DEL SECTOR

El sector en el que se va a desarrollar esta aplicación es algo difuso e indefinido todavía. Se podría considerar una mezcla entre el sector de las redes sociales y el sector de los servicios de poner en contacto clientes con artistas (algo similar a la aplicación Wallapop que pone en contacto clientes con vendedores particulares).

Esta aplicación puede ser considerada innovadora al afirmar que el sector es un nicho de mercado.

Podemos considerarlo un nicho de mercado ya que se ha observado una necesidad insatisfecha por parte de potenciales clientes. La necesidad de una herramienta o motor de búsqueda de artistas de tatuajes, que facilite a los potenciales clientes la búsqueda de tatuadores mediante el uso de filtros.

Por tanto, este nicho de mercado con una necesidad insatisfecha es una oportunidad que se debe aprovechar.

PÚBLICO OBJETIVO

El público objetivo son los potenciales clientes que pueden llegar a consumir nuestro producto o servicio. No se hacen apenas distinciones en las características del público objetivo considerado como clientes, ya que esta iniciativa está enfocada a todo tipo de usuarios, aunque si podemos decir que el público objetivo en el que mejor encaja son todas aquellas personas que tienen tatuajes o que buscan hacerse uno. Pero si podemos hacer un matiz en el público objetivo considerado artista, ya que esta aplicación está fuertemente enfocada a los artistas de tatuajes y anilladores.

Pequeño estudio realizado por el desarrollador con entrevistas reales a personas con ningún tatuaje, con pocos y con gran cantidad de tatuajes, y siempre usando el mismo discurso introductorio:

“¿Qué opinas de una aplicación para conectar tatuadores con gente que busca tatuajes con ciertos requisitos? Una persona se puede registrar como tatuador y poner la ubicación de su estudio, un muro de fotos con los tatuajes y los estilos que trabaja y una lista de precios. O también se pueden registrar como cliente y buscar tatuadores con un margen geográfico y un estilo concreto.”

En dicho estudio hemos obtenido estos datos:

-Noelia (26/11/2021): Observación de una necesidad insatisfecha. Búsqueda de un tatuador no muy lejos de su ubicación y dentro de un presupuesto limitado. Además, quería ver que ese tatuador tuviera imágenes de trabajos previos de tatuajes de “anime”.

-Toni (01/04/2022): Le gusta mucho y dice que usaría este tipo de aplicación.

-Darío (01/04/2022): Le parece una gran idea y cree que, a pesar de tener tatuadores de confianza, si usa la aplicación y ve tatuadores nuevos con buenas valoraciones o comentarios positivos, los probaría.

-Jorge (01/04/2022): Nunca había oído de una aplicación que hiciera eso y le encanta la idea.

-Sandra (09/04/2022): Le gusta la idea y además aporta la idea de añadir “piercing” a la aplicación. Por lo tanto, los usuarios que se registren como artistas podrán ser tatuadores o anilladores.

-Edu (09/04/2022): Le parece buena idea, pero es un poco reacio a creer que los clientes que tengan tatuadores de confianza probaran nuevos tatuadores.

COMPETENCIA EN EL MERCADO

No se ha encontrado mucha competencia que merezca la pena destacar. Como hemos dicho antes, es un sector en nicho de mercado, por tanto, cada búsqueda que se realiza nos da como resultados buscadores de diseños de tatuajes o dibujos de tatuajes. Lo que no encontramos apenas son buscadores de tatuadores. Tras una profunda búsqueda, solo hemos encontrado dos webs dedicadas a ello y una app Android

<https://tutatuador.com/>

<https://tattooalia.com/>

Tattoodo (PlayStore)

Por desgracia no somos los primeros, pero por suerte no hay apenas competencia. En el desarrollo de este proyecto vamos a tratar de implementar soluciones que solventen los problemas, críticas y comentarios negativos que tiene esta app y estas webs.

ANÁLISIS DAFO

-Debilidades:

Poco conocimiento de la gestión y mantenimiento diario de una aplicación Android que es considerada como una red social.

Pocos recursos para dar a conocer la app mediante una campaña de publicidad.

-Amenazas:

Possible amenaza de que la app no tenga la cantidad de registros de tatuadores necesaria como para que sea atractiva geográficamente hablando.

-Oportunidades:

El sector como tal es una oportunidad, ya que hay muy baja competencia.

La oportunidad de que el boca a boca dé a conocer la aplicación.

El hecho de que todavía no se conozca una aplicación “predominante” en este sector es una oportunidad de “llegar pisando fuerte”.

-Fortalezas:

Nos diferenciamos de nuestros competidores por enfocar la app al motor de búsqueda para los clientes, dando la posibilidad de poner requisitos o condiciones a su búsqueda.

Por otro lado, nos diferenciamos de nuestros competidores al darle más facilidades a los usuarios que se registren como tatuador (ya que por lo que se ha investigado, en la competencia es complicado para los tatuadores darse a conocer si no pagan una cuenta premium). Por tanto, ofreceremos más facilidades y visibilidad a los artistas.

1. 3. Valor del producto

Considerado un producto de alto valor debido al sector en el que se encuentra, y a su bajo nivel de competencia. Al ser considerado un nicho de mercado, aumenta el valor del proyecto, convirtiendo esta iniciativa emprendedora en una idea innovadora que vale la pena desarrollar. El estudio de mercado realizado ha reflejado un gran atractivo hacia esta idea por parte de los jóvenes que tienen tatuajes. Por tanto, tras las impresiones obtenidos por parte de potenciales clientes, se deduce que esta app puede tener una gran acogida siempre que se dé a conocer.

A título personal considero lo considero un proyecto de alto valor debido a que tiene posibilidades de irrumpir en un nicho de mercado inexplotado. Ya que no hay ninguna aplicación “predominante” de este tipo.

2. ANÁLISIS DE LA SOLUCIÓN

2. 1. Análisis y especificación de requisitos

REQUISITOS FUNCIONALES

RF1 -> La aplicación debe permitir la posibilidad de que un nuevo usuario pueda registrarse con uno de los dos tipos de perfil disponibles (tatuador o cliente).

RF2 -> La aplicación debe permitir la posibilidad de registrarse con Google

RF3 -> La aplicación debe permitir la posibilidad de registrarse con Facebook

RF4 -> La aplicación debe permitir la posibilidad de que un usuario registrado como tatuador pueda vincular su perfil de Instagram o vincular otros enlaces a su sitio web.

RF5 -> El usuario registrado como tatuador deberá establecer la ubicación de su estudio de tatuajes al registrar su perfil.

RF6 -> El usuario registrado como tatuador deberá establecer su número CIF y ciertos datos de importancia (para evitar registros fraudulentos).

RF7-> El usuario registrado como tatuador tendrá un apartado de perfil de usuario, con su nombre, sus datos de contacto como teléfono o correo, su ubicación. Además de un muro, una tabla de precios, la opción de contactar por chat y la opción de reservar una cita. También debe poder editar datos de su perfil o borrarlo cuando quiera.

RF8 -> El usuario registrado como tatuador podrá tener un muro donde poder publicar fotos de sus trabajos y añadir etiquetas de estilos a esas publicaciones. Tendrá la posibilidad de añadir, editar o borrar publicaciones cuando quiera.

RF9 -> El usuario registrado como tatuador podrá tener una tabla de precios X tamaño (por ejemplo: 50€ para 10cmX10cm)

RF10 -> El usuario registrado como tatuador tendrá habilitada una sección de chat de mensajes de texto, mediante el cual los clientes podrán contactar con el tatuador.

RF11 -> El usuario que se registre como cliente, tendrá un muro a modo lista, con los perfiles de los tatuadores que ha guardado en favoritos. (OPCIÓN que está en el perfil público del tatuador y solo es visible para el cliente)

RF12 -> El usuario que se registre como cliente, tendrá disponible un apartado de búsqueda para poder ver los perfiles de los tatuadores registrados en la aplicación. Este apartado de búsqueda podrá buscar mediante texto, aplicando filtros para la ubicación, aplicando filtros para el precio y/o aplicando filtros de etiquetas de estilos.

RF13 -> El usuario que se registre como cliente, podrá ver los perfiles públicos de los tatuadores, abrirles conversación de chat y reservar una cita con fecha y hora. (Esta opción está desactivada hasta que el tatuador la habilite.)

RF14 -> Al abrirse una conversación de chat por parte del cliente, le llega al tatuador una notificación de nuevo chat. En la ventana del chat del tatuador está el botón de “habilitar opción de solicitar cita” (esta opción no es visible para el cliente). Si el tatuador habilita la opción de solicitar cita, el cliente se le desbloquea en su chat la opción de “solicitar cita”. Esto le permitirá concretar una cita de fecha y hora para una sesión de tatuaje.

RF15 -> Usuario Administrador: La aplicación podrá permitir que un usuario con rol administrador pueda acceder a ella (solo Login, no habrá forma de que se pueda hacerse un nuevo registro de usuario con este tipo de rol.)

RF15 -> El usuario Administrador puede gestionar ciertos aspectos como eliminar o modificar usuario Tatuadores y clientes

REQUISITOS NO FUNCIONALES

RNF1-> El cliente móvil del sistema será desarrollado en AndroidStudio mediante lenguaje de programación kotlin

RNF2 -> Todos los datos que usará la aplicación deben almacenarse remotamente usando los servicios de Firebase de Google (no guardando ningún dato en local)

RNF3 -> Los métodos de autenticación del usuario son su email y contraseña, número de teléfono, Google, Facebook y Twitter.

RNF4 -> La gama de colores de la aplicación constará de una combinación entre colores azul marino y grises, sobre fondos blancos o de colores claros.

REQUISITOS DE INFORMACIÓN

RI1 -> Usuario tatuador -> El usuario que se registra con perfil de tatuador

- Id Usuario
- Nombre De Usuario
- Imagen
- CIF
- Correo electrónico
- Teléfono
- Ubicación geográfica del estudio de tatuajes
- Lista de precios X tamaño
- Lista de tamaños
- Lista Favoritos (perfiles de tatuadores guardados en favoritos)
- Lista roles

RI2 -> Usuario cliente -> Es el usuario que se registra buscando tatuadores

- Id Usuario
- Nombre De Usuario
- Imagen
- Correo electrónico
- Teléfono
- Lista Favoritos (perfiles de tatuadores guardados en favoritos)
- Lista roles

RI3 -> Publicación -> Son las “fotos” que tiene el usuario tatuador publicadas en su muro. Cada publicación consta de:

- Id Publicación
- id Imagen (Storage)
- Id Artista propietario
- Lista de etiquetas

-Lista roles

RI4 -> Chat -> Son los chats que se abren entre cliente y tatuador para gestionar una fecha de cita para tatuaje, y consta de:

-Id chat

-Id Tatuador

-Nombre tatuador

-Id Cliente

-Nombre cliente

-Fecha de la cita

RI5 -> Comentario -> Son los mensajes de texto que se envían a través del chat entre cliente y tatuador:

-Id mensaje

-Id tatuador

-Id Cliente

-Fecha/ Hora

-Texto del mensaje

RI6 -> Administrador -> Son los usuarios con rol de Administrador que realizarán ciertas funciones de gestión.

-Id Usuario

-Nombre De Usuario

-Imagen

- Correo electrónico

-Teléfono

-Lista Favoritos (perfiles de tatuadores guardados en favoritos)

-Lista roles

2. 2. Análisis de escenarios (casos de uso)

Un diagrama de casos de uso es una forma de diagrama de comportamiento UML mejorado.

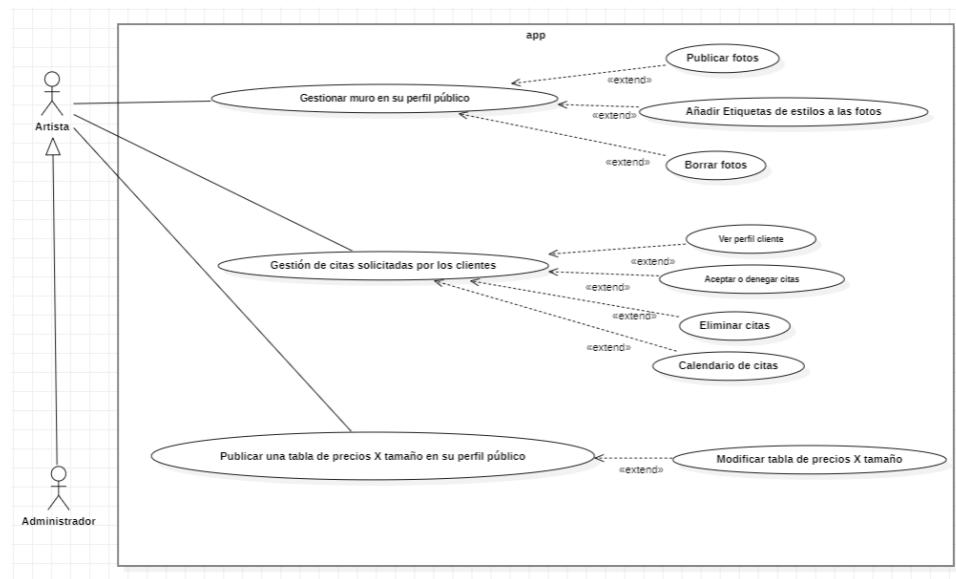
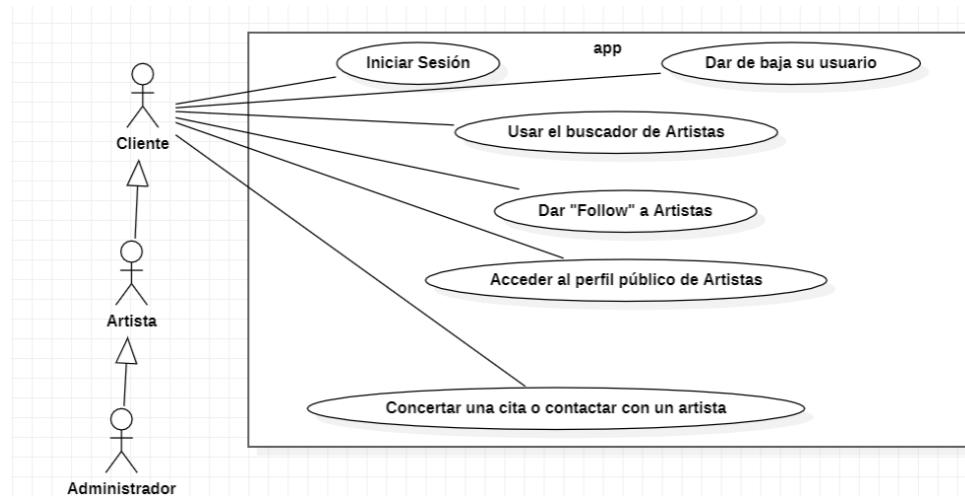
Ofrecen una visión general de los actores involucrados en un sistema, las diferentes funciones que necesitan esos actores y cómo interactúan estas diferentes funciones.

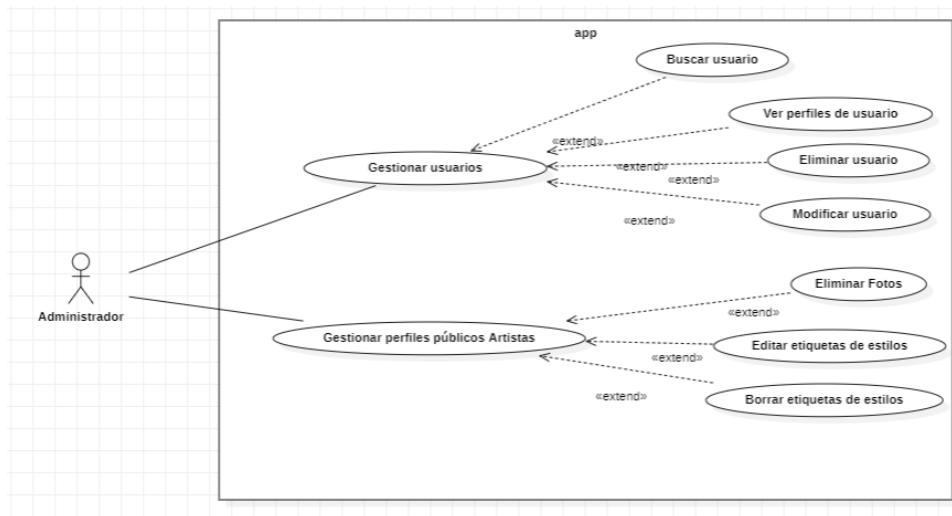
ACTORES

-Artista (Usuario Tatuador): Son aquellos usuarios que se registran en la aplicación con la intención de crearse una cuenta de artista y publicitar su negocio de tatuador.

-Cliente (Usuario Básico): Son aquellos usuarios que se registran en la aplicación con intención de buscar tatuadores.

-Administrador: Tendrá acceso a todas las gestiones de la aplicación.





CU-1.01: Iniciar Sesión

-Actores: **Cliente, Artista, Administrador**

-Descripción: Iniciar sesión en la aplicación

-Precondición: Tener un usuario registrado

-Flujo de Eventos (Secuencia Normal):

1- El usuario indica su correo y contraseña

2- El sistema verifica que sea válido

-Excepciones:

1-El usuario no está registrado

2-Credenciales incorrectas

3-Rol incorrecto

-Postcondición: El usuario accede a la aplicación (al perfil que le corresponda según su rol)

-Comentarios:

CU-1.02: Dar de baja su usuario

-Actores: **Cliente, Artista, Administrador**

-Descripción: Dar de baja su usuario, eliminándolo su perfil de la aplicación.

-Precondición: Haber iniciado sesión con un usuario registrado.

-Flujo de Eventos (Secuencia Normal):

1-El usuario selecciona el panel de ajustes

2-Selecciona la opción “Dar de baja usuario”

3-Se abrirá una ventana emergente pidiendo confirmación

4-En caso correcto, el sistema dará de baja el usuario

-Excepciones:

-Postcondición: El usuario volvería a la ventana de registro.

-Comentarios: Se plantea la posibilidad de escoger un sistema de borrado lógico, por el cual no se eliminaría de la base de datos. El usuario sé da de baja con la posibilidad de retomar su cuenta en un futuro.

CU-1.03: Usar el buscador de Artistas

-Actores: **Cliente, Artista, Administrador**

-Descripción: Usar el buscador de la app para encontrar artistas. Pudiendo aplicar filtros.

-Precondición: Haber iniciado sesión con un usuario registrado.

-Flujo de Eventos (Secuencia Normal):

1-El usuario selecciona la opción de búsqueda en el panel desplegable lateral

2-El usuario puede buscar Artistas sin aplicar ningún tipo de filtro

3-Puede Aplicar filtros como el nombre, la distancia en KM, precios X tamaño o estilos.

4-El usuario pulsa el botón buscar.

5-Aparece un listado con los perfiles de Artistas que cumplan esos filtros

-Excepciones:

1-No hay artistas que mostrar

-Postcondición: En el listado se puede navegar hasta el perfil público de un Artista haciendo click en su nombre.

-Comentarios: Se plantea la opción de implementar también criterios de ordenación (alfabéticamente y por precio más barato o más caro) al mostrar los resultados de la búsqueda

CU-1.04: Acceder al perfil público de un Artista

-Actores: **Cliente, Artista, Administrador**

-Descripción: Un usuario puede visitar un perfil público de un Artista.

-Precondición: Haber navegado desde el buscador hasta el perfil público de un Artista.

-Flujo de Eventos (Secuencia Normal):

1-Tras la búsqueda, en la lista de resultados de Artistas, el usuario puede hacer click en un Artista

2-Tras hacer click, se navega al perfil público del Artista

-Excepciones:

-Postcondición: Tras navegar, el usuario se encuentra en la ventana del perfil público de un Artista, donde puede ver su nombre, su muro de fotos, su tabla de precios, darle a follow y concertar una cita.

-Comentarios:

CU-1.05: Dar “Follow” a Artistas

-Actores: **Cliente, Artista, Administrador**

-Descripción: Al visitar un perfil público, se puede dar al botón de “Follow” para que aparezca entre tus favoritos.

-Precondición: Haber navegado desde el buscador hasta el perfil público de un Artista.

-Flujo de Eventos (Secuencia Normal):

1-Navegar hasta el perfil público de un Artista

2-En la ventana del perfil, el usuario puede hacer click en el botón de “Follow” debajo del nombre y la foto del Artista

3-Cambia la apariencia del botón, ahora pone “Unfollow”

-Excepciones:

-Postcondición: Cuando un usuario le da a seguir a un Artista, aparecerá el nombre y foto del Artista en una lista de Favoritos en el panel desplegable lateral.

-Comentarios:

CU-1.05: Contactar con el Artista

-Actores: **Cliente, Artista, Administrador**

-Descripción: Al visitar un perfil público, se puede dar al botón de “Contactar” para que se despliegue una ventana de chat entre el usuario y el Artista

-Precondición: Haber navegado desde el buscador hasta el perfil público de un Artista.

-Flujo de Eventos (Secuencia Normal):

1-Navegar hasta el perfil público de un Artista

2-En la ventana del perfil, el usuario puede hacer click en el botón de “Concertar cita/contactar” debajo del nombre y la foto del Artista.

3-Se abre una ventana de chat de conversación entre el Artista y el Usuario

4-En caso de que el Artista le confirme una cita (tras una conversación de chat), el usuario verá cómo se ha actualizado la fecha(cita) en la sección de chats

-Excepciones:

-Postcondición: Una vez abierto un chat, esta conversación se queda de forma permanente en caso de querer volver a comunicarse. Se vuelve a acceder desde el botón “Concertar cita/ Contactar”

-Comentarios: En el perfil de usuario, en el panel desplegable lateral, hay un apartado de “Citas confirmadas” con un calendario.

CU-2.01: Gestionar muro en su perfil público

-Actores: **Artista, Administrador**

-Descripción: El Artista puede editar su perfil y su muro de fotos

-Precondición: Haberse registrado como usuario Artista e iniciar sesión en la aplicación

-Flujo de Eventos (Secuencia Normal):

1- Tras iniciar sesión en la aplicación, la ventana de perfil.

2- En la primera ventana, la del perfil público, puede hacer click en el botón “Editar”

3-Puede cambiar su foto de perfil, su nombre, sus datos de contacto.

-Excepciones:

-Postcondición: Modifica su muro de perfil.

-Comentarios:

CU-2.01.01: Publicar fotos en su muro del perfil público

-Actores: **Artista, Administrador**

-Descripción: En su muro de fotos, puede publicar fotos nuevas

-Precondición: Haberse registrado como usuario Artista e iniciar sesión en la aplicación

-Flujo de Eventos (Secuencia Normal):

1- Debajo del nombre del perfil, el usuario hace click en el botón “Publicar”

2- Se despliega un mensaje mostrando dos opciones “Cámara” o “Galería”

3-El usuario puede escoger una foto de su galería o tomar una instantánea con su cámara, y seleccionarla como foto para publicar.

4-Se verá la foto en una ventana de vista previa, con un botón de “Confirmar” y otro de “Cancelar”.

5-Si el usuario pulsa “Confirmar” se publicará la foto en su muro

6-Se refrescará la ventana para ver la foto publicada

-Excepciones:

-Postcondición: Se verá la foto publicada en el muro en cuadricula de 3 columnas, siendo esta foto la primera de la izquierda.

-Comentarios: Se plantea la opción de implementar alguna opción de “Denunciar publicación” libre para todos los usuarios, con el fin de prevenir el posible mal uso por parte de los usuarios Artistas.

CU-2.01.02: Añadir etiquetas a las fotos publicadas en su muro

-Actores: **Artista, Administrador**

-Descripción: En su muro de fotos, puede añadir etiquetas a las fotos publicadas.

-Precondición: Haberse registrado como usuario Artista, iniciar sesión en la aplicación y haber publicado alguna foto en el muro

-Flujo de Eventos (Secuencia Normal):

- 1- Se selecciona una de las fotos publicadas en el muro haciendo click en ella
- 2- Se despliega un mensaje preguntando al usuario si desea añadir etiquetas de estilos a esa foto, junto a un botón de “Confirmar” y otro de “Cancelar”
- 3-Si confirma, se despliega una ventana con una caja de texto donde poder escribir y un botón de “Confirmar” y otro de “Cancelar”.
- 4-Tras escribir las etiquetas de estilos oportunas, el usuario le da a Confirmar.
- 5-Se añaden las etiquetas como pie de foto

-Excepciones:

-Postcondición: Solo al hacer click en una foto se puede ver en su pie de foto las etiquetas de estilos

-Comentarios: Cada etiqueta será una sola palabra que irá precedida de un símbolo #.

CU-2.01.03: Borrar fotos publicadas en su muro

-Actores: **Artista, Administrador**

-Descripción: En su muro de fotos, puede eliminar las fotos publicadas.

-Precondición: Haberse registrado como usuario Artista, iniciar sesión en la aplicación y haber publicado alguna foto en el muro

-Flujo de Eventos (Secuencia Normal):

- 1- El usuario mantiene presionado el click sobre una imagen del muro
- 2- Se despliega un mensaje preguntando al usuario si desea eliminar esa foto, junto a un botón de “Confirmar” y otro de “Cancelar”
- 3-Si confirma, se elimina permanentemente esa foto de su muro.

4-Se refresca la ventana de perfil y se actualiza el muro sin la foto

-Excepciones:

-Postcondición: Las fotos eliminadas no se pueden recuperar

-Comentarios:

CU-2.02: Proponer citas a los clientes

-Actores: **Artista, Administrador**

-Descripción: El artista puede ofrecer fechas de citas a los clientes que se pongan en contacto con ellos.

-Precondición:

1-Haberse registrado como usuario Artista e iniciar sesión en la aplicación

2-Un usuario ha pulsado el botón “Concertar cita/Contactar” en el perfil del Artista

-Flujo de Eventos (Secuencia Normal):

1- Tras haber pulsado el usuario el botón “Concertar cita/Contactar”, le aparece una notificación al Artista.

2- En su panel lateral desplegable (no visible para usuarios Clientes), le saldrá un apartado de “Clientes”.

3-Tras hacer click en el apartado Chats/Citas, se abre una ventana con una lista con los nombres de los clientes que le hayan contactado y hayan abierto un chat

4-Selecciona el cliente que le ha escrito, y se abre una ventana de chat.

5-La ventana tiene la opción de “Crear cita”. Al pulsarla se despliega un calendario y un reloj.

6-Tras seleccionar la fecha, al usuario le llega una se le actualizará la fecha en el apartado Chats/Citas, al igual que al Artista.

-Excepciones:

-Postcondición: Desde el apartado de “Mis Citas”, en el panel desplegable lateral, se pueden ver un calendario con las citas programadas.

-Comentarios:

CU-2.02.03: Eliminar una Chat

-Actores: **Artista, Administrador**

-Descripción: El artista puede ver el aceptar o denegar una cita a un usuario

-Precondición:

1-Haberse registrado como usuario Artista e iniciar sesión en la aplicación

2-Un usuario ha pulsado el botón “Concertar cita/Contactar” en el perfil del Artista

3-Tener una cita programada en su calendario del apartado “Mis citas”

-Flujo de Eventos (Secuencia Normal):

1- Tras abrir el calendario en el apartado “Mis citas”, el usuario puede seleccionar una cita.

2-Al hacer click en la cita, se despliega mensaje preguntando al usuario si desea eliminar esa cita foto, junto a un botón de “Confirmar” y otro de “Cancelar”

3-Si se confirma, el Artista elimina esa cita de su calendario

-Excepciones:

-Postcondición: Al cliente también se le elimina la cita de su apartado “Mis citas” y se le envía una notificación al correo.

-Comentarios:

CU-2.01.03: Publicar una tabla de precios X tamaño

-Actores: **Artista, Administrador**

-Descripción: En su perfil público, los usuarios pueden ver los precios que ofrece el tatuador.

-Precondición: Haberse registrado como usuario Artista, iniciar sesión en la aplicación y haber publicado alguna foto en el muro

-Flujo de Eventos (Secuencia Normal):

1- Debajo del nombre de usuario, habrá un botón “Tabla de precios X tamaño”

2- Se despliega una ventana con una donde se pueden ver los precios X tamaño

3-Un botón de “Editar” para que el artista pueda modificar las cifras. Al darle a Editar, el botón cambia su apariencia a “Guardar”

4-Al darle a “Guardar” se establecen los cambios que se hayan realizado en las cifras.

-Excepciones: El usuario no podrá darle a cancelar y recuperar lo que había previo a editar.

-Postcondición: Se establecen los cambios de los precios X tamaño

-Comentarios: Desde el punto de vista de los usuarios solo se puede ver esta tabla, no editar.

CU-3.01: Gestionar usuarios

-Actores: **Administrador**

-Descripción: Gestión completa de los usuarios por parte del Administrador

-Precondición: Tener un usuario registrado con rol de Administrador

-Flujo de Eventos (Secuencia Normal):

- 1- El usuario Administrador tiene en su panel lateral desplegable la opción de “Gestión de usuarios”
- 2-Tras hacer click en esa opción se abrirá una ventana con una lista con todos los usuarios, y una caja de texto en la parte superior junto a un botón de “Buscar”.

-Excepciones:

- Postcondición: El Administrador contempla una ventana con la lista de usuarios
- Comentarios: El Administrador tiene acceso además de esta, a todas las funciones que tienen usuarios Clientes y usuarios Artistas, con la diferencia de que su perfil no es visible para ningún otro usuario.

CU-3.01.01: Buscar usuario

-Actores: **Administrador**

-Descripción: Búsqueda de cualquier usuario registrado en la aplicación

-Precondición:

- 1-Tener un usuario registrado con rol de Administrador
- 2-Acceder al apartado “Gestión de Usuarios”

-Flujo de Eventos (Secuencia Normal):

- 1-En la ventana con una lista con todos los usuarios, en la caja de texto en la parte superior puede escribir el correo o nombre de usuario, para poder buscar a cualquier usuario registrado.
- 2-Se actualizará la lista con los resultados obtenidos de la búsqueda

-Excepciones:

-Postcondición: El Administrador contempla una ventana con la lista de usuarios que cumplen la búsqueda

-Comentarios:

CU-3.01.02: Ver perfil de un usuario

-Actores: **Administrador**

-Descripción: Búsqueda de cualquier usuario registrado en la aplicación

-Precondición:

- 1-Tener un usuario registrado con rol de Administrador
- 2-Acceder al apartado “Gestión de Usuarios”

-Flujo de Eventos (Secuencia Normal):

- 1-En la ventana con una lista con todos los usuarios, hacer click en un usuario de la lista.
- 2-Tras hacer click en un usuario, se navega al perfil de ese usuario

-Excepciones:

-Postcondición: El Administrador contempla el perfil de un usuario en concreto

-Comentarios:

CU-3.01.03: Eliminar perfil de un usuario

-Actores: **Administrador**

-Descripción: Eliminar cualquier usuario seleccionado de la lista

-Precondición:

- 1-Tener un usuario registrado con rol de Administrador
- 2-Acceder al apartado “Gestión de Usuarios”

-Flujo de Eventos (Secuencia Normal):

- 1-En la ventana con una lista con todos los usuarios, mantener click en un usuario de la lista.
- 2-Tras mantener click en un usuario, se navega al perfil de ese usuario despliega mensaje preguntando al Administrador si desea “Eliminar” ese usuario o si desea “Modificar”.
- 3- Si hace click en “Eliminar” se despliega mensaje preguntando al usuario si desea “Confirmar” o “Cancelar”
- 4-Si Confirma, se da de baja ese usuario de la aplicación

-Excepciones:

-Postcondición: El usuario se elimina y por tanto se eliminan las citas que tuviera.

-Comentarios:

CU-3.01.04: Modificar perfil de un usuario

-Actores: **Administrador**

-Descripción: Modificar de cualquier usuario seleccionado de la lista

-Precondición:

- 1-Tener un usuario registrado con rol de Administrador
- 2-Acceder al apartado “Gestión de Usuarios”

-Flujo de Eventos (Secuencia Normal):

- 1-En la ventana con una lista con todos los usuarios, mantener click en un usuario de la lista.
- 2-Tras mantener click en un usuario, se navega al perfil de ese usuario despliega mensaje preguntando al Administrador si desea "Eliminar" ese usuario o si desea "Modificar".
- 3- Si hace click en "Modificar" se navega hasta el perfil del usuario seleccionado.
- 4-El Administrador puede editar todos los campos o cambiar la foto de perfil, solo haciendo click sobre ellos.
- 5-Al salir, preguntará al Administrador si desea confirmar los cambios guardados.
- 6- Si confirma, volverá a la ventana con el listado de usuarios, habiéndose modificado el usuario anterior.

-Excepciones:

-Postcondición: El usuario se modifica y se le comunica sus cambios por correo.

-Comentarios:

CU-3.02: Gestionar perfil público de los Artistas

-Actores: **Administrador**

-Descripción: Gestión completa de los perfiles públicos de los usuarios Artistas por parte del Administrador

-Precondición: Tener un usuario registrado con rol de Administrador

-Flujo de Eventos (Secuencia Normal):

- 1- El usuario Administrador tiene en su panel lateral desplegable la opción de "Gestión de Artistas"
- 2-Tras hacer click en esa opción se abrirá una ventana con una lista con todos los usuarios registrados como artistas, y una caja de texto en la parte superior junto a un botón de "Buscar".

-Excepciones:

-Postcondición: El Administrador contempla una ventana con la lista de usuarios que son Artistas

-Comentarios: El Administrador tiene acceso además de esta, a todas las funciones que tienen usuarios Clientes y usuarios Artistas, con la diferencia de que su perfil no es visible para ningún otro usuario.

3. PLANIFICACIÓN DE TAREAS Y ESTIMACIÓN DE COSTES

3. 1. Planificación y organización de tareas

La planificación de tareas es una parte fundamental en los pasos previos al desarrollo de un proyecto. La función principal de la aplicación se fragmenta en una división de tareas clara y simple, que facilite su desarrollo.

PLANIFICACIÓN DE TAREAS

03/04/2022

Presentación del proyecto

-Explicación resumida

10/04/2022

Análisis de la solución

-Análisis de la solución (requisitos, mapa de historia de usuario...)

15/04/2022

Presentación del proyecto

-Estudio de mercado

-Valor del producto

22/04/2022

Análisis de la solución

- Análisis de escenarios (casos de uso, mapa de interacción...)

29/04/2022

Planificación de tareas y estimación de costes

-Planificación y organización de tareas

06/05/2022

Planificación de tareas y estimación de costes

-Estimación de costes y recursos: hardware, software y humanos

-Herramientas usadas

-Gestión de riesgos

13/05/2022

Diseño de la solución

-Diseño de la interfaz de usuario y prototipos

18/05/2022

Diseño de la solución

-Diagrama de clases

-Diseño de la persistencia de la información

23/05/2022

Diseño de la solución

- Arquitectura del sistema

29/05/2022

Implementación de la solución

- Análisis tecnológico

12/06/2022

Implementación de la solución

- Elementos a implementar

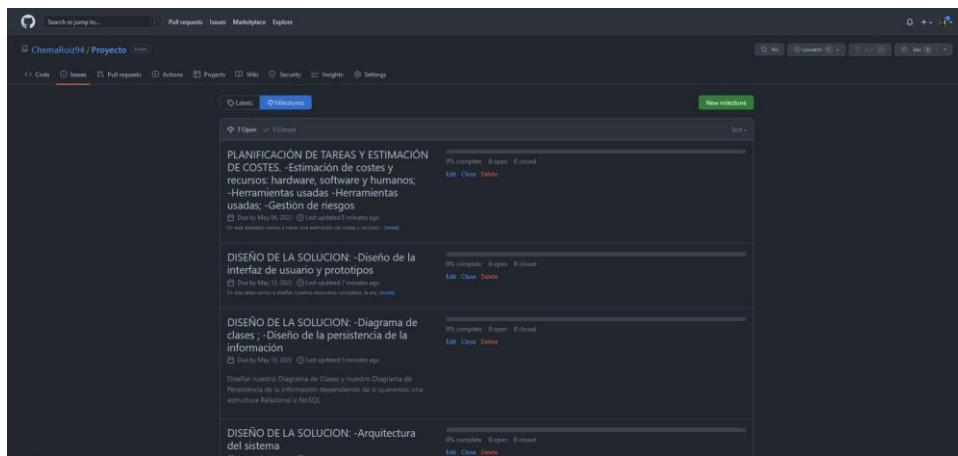
19/06/2022

Entrega de proyecto, documentación y repositorio

HERRAMIENTAS UTILIZADAS PARA LA PLANIFICACIÓN DE TAREAS

Las herramientas utilizadas son **GitHub** y **GitKraken**.

Para la gestión de las tareas usaremos GitHub. Para ello, iremos a la pestaña “Issues” y en “miletstones” crearemos todas las tareas pendientes, con su fecha de entrega y descripción.



Creamos las Issues.

The screenshot shows the GitHub Issues page for a project named 'ChemaRuiz94 / Proyecto'. At the top, there's a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. Below the header, a banner says 'Label issues and pull requests for new contributors' with a 'Dismiss' button. A 'New issue' button is visible. The main area displays a list of issues under the 'New issue' tab, with filters for Open and Closed status. The issues are categorized into several columns:

- Entrega de proyecto, documentación y repositorio
- Implementación de la solución
- Implementación de la solución
- Desarrollo de la solución
- Planificación de tareas y estimación de costes

Each issue has a small green circular icon next to it. At the bottom right of the list, there's a note: 'Last updated in the last three days ago • 2022-04-30'.

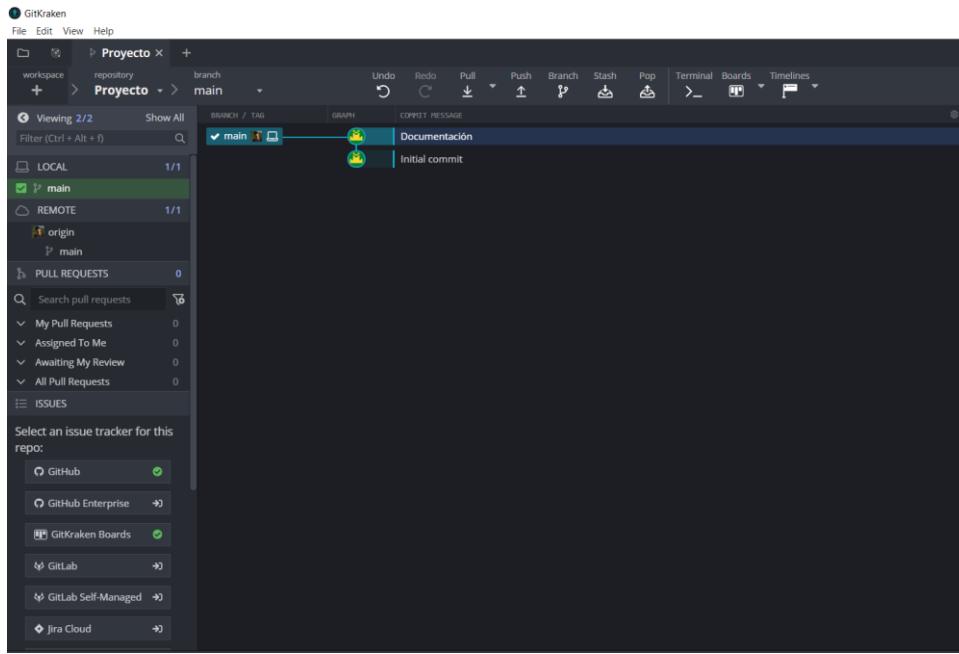
En la pestaña “projects” creamos un proyecto, y añadimos las Issues que hemos creado previamente a la columna que le corresponda a cada tarea.

The screenshot shows the GitHub Projects board for the same project. The board has three columns: 'To do', 'In progress', and 'Done'. Each column contains cards representing the issues from the previous screenshot, grouped by category. The 'To do' column has cards for 'Entrega de proyecto, documentación y repositorio' and 'Implementación de la solución'. The 'In progress' column has cards for 'Implementación de la solución', 'Desarrollo de la solución', 'Desarrollo de la solución', 'Desarrollo de la solución', and 'Desarrollo de la solución'. The 'Done' column has a card for 'Planificación de tareas y estimación de costes'. At the bottom of the board, there are buttons for 'Automated to: To do', 'Manage', 'Automated to: In progress', 'Manage', 'Automated to: Done', and 'Manage'.

Podemos sincronizar con GitKraken para verlo en Timeline.

GESTION DEL FLUJO DE TRABAJO

Git es uno de los sistemas de control de versiones más usados. GitFlow es el flujo de trabajo basado en Git, con el podemos dividir y ramificar el trabajo para facilitar su desarrollo.



DOCUMENTACIÓN

Como herramienta para la documentación sé usará GitHub, ya que en su pestaña “Wiki” nos permite gestionar la documentación de nuestro proyecto.

3. 2. Estimación de costes y recursos: hardware, software y humanos.

En este apartado vamos a realizar una aproximación de los costes monetarios necesarios para realizar las funciones necesarias para el proyecto. Los costes se estiman para todos los recursos asignados al proyecto, es decir, recursos de trabajo, recursos materiales, costes de servicios e instalaciones y posibles costes de contingencias.

Esta no es una estimación fija, ya que se ha de ir modificando junto con la evolución de la aplicación. No solo hablamos de la evolución en el desarrollo del proyecto, también tendremos en cuenta la posible evolución tras su publicación.

Por tanto, vamos a realizar una estimación de costes para la situación actual, y se incluirá una previsión para una posible situación a largo plazo.

En costes de desarrollo:

Desarrollo y mantenimiento por cuenta del desarrollador principal. (No sé prevén ingresos en corto plazo).

En costes de licencias incluiremos:

-Despliegue de la aplicación en Google Play Store: Esto incluye 25 dólares como licencia de desarrollador para poder subir aplicaciones al Play Store. Además del 30% de los ingresos que genere la aplicación.

-El despliegue en Firebase es difícil de cuantificar ya que el coste varía en función de la cantidad de descargas de la aplicación y la cantidad de peticiones, lecturas, escrituras y autentificaciones.

Incorpora un despliegue gratuito hasta cierto límite, por lo que se incluye un ejemplo de la previsión posible para un desglose pequeño (50.000 descargas)

	Pequeña (50,000 instalaciones)	Media (1 millón de instalaciones)	Grande (10 millones de instalaciones)
Por 50,000 instalaciones de la app (5,000 usuarios activos por día): \$ 12.14 por mes			
Costos de lecturas y escrituras			
	Total de 400,000 lecturas diarias	= 50,000 lecturas sin costo + (350,000 lecturas a \$0.06 por 100,000)	= 3.5 * \$0.06
		\$ 0.21 por día * 30 = \$ 6.30	
Costo total mensual = \$ 11.10	Total de 100,000 escrituras diarias	= 20,000 escrituras sin costo + (80,000 escrituras a \$0.18 por 100,000)	= 0.8 * \$0.18
		\$ 0.14 por día * 30 = \$ 4.20	
	Total de 100,000 eliminaciones diarias	= 20,000 eliminaciones sin costo + (80,000 eliminaciones a \$0.02 por 100,000)	= 0.8 * \$0.02
		\$ 0.02 por día * 30 = \$ 0.60	
Costos de almacenamiento y red			
	20 KB por DAU de salida diaria * 5,000 DAU	= 100 MB de salida diaria * 30	= 3 GB de salida de red mensual
		3 GB de salida sin costo = sin costo ¹	
Costo total mensual = \$ 1.04 por mes	15 KB de almacenamiento diario de mensajes por DAU + 3 KB de almacenamiento por instalación ²	= 45 KB de almacenamiento por DAU * 5,000 DAU	= 225 MB de almacenamiento diario por DAU * 30
			= 6.75 GB de uso de almacenamiento mensual
	¹ 1 GB de almacenamiento sin costo + (5.75 * \$0.18) = \$1.04 por mes		

PRESUPUESTO

Voy a realizar una aproximación monetaria al presupuesto que va a requerir el desarrollo de este proyecto (puede no ser muy precisa ya que, como he dicho antes, irá modificándose junto a la evolución en el desarrollo del proyecto.):

Nº	Descripción	Cantidad Horas aprox.	Precio X Hora	Importe

1	Análisis de la solución y Documentación	100 h	(Desarrollador no ingresa de momento)	0€
2	Herramientas de desarrollo (instalación y configuración)	1h	0€	0€
3	Desarrollo de la aplicación	200 h	(Desarrollador no ingresa de momento)	0€
4	Alta como desarrollador en Google Play Store	1h	25€	25€
5	Mantenimiento de la aplicación	anual	30% de ingresos de la app	30% mensual
6	Servicios Firebase	anual	Gratis/12,14\$	145,56\$

3. 3. Herramientas usadas

Las Herramientas que voy a utilizar son las siguientes:

Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan tu productividad para desarrollar apps para Android. Por ejemplo, la integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra.

Se ha optado por este IDE para el desarrollo de la aplicación.



Git, GitHub y GitKraken

Git es un **sistema de control específico de versión de fuente abierta** creada por Linus Torvalds en el 2005.

Específicamente, Git es **un sistema de control de versión distribuida**, lo que quiere decir que la base del código entero y su historial se encuentran disponibles en la computadora de todo desarrollador, lo cual permite un fácil acceso a las bifurcaciones y fusiones.

GitHub es una compañía sin fines de lucro que ofrece un servicio de hosting de repositorios almacenados en la nube. Esencialmente, hace que sea más fácil para individuos y equipos usar Git como la versión de control y colaboración.

GitKraken (en su versión gratuita), como herramienta que facilita el uso de Git.



Firebase

Firebase es una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles ubicada en la nube, integrada con Google Cloud Platform, que usa un conjunto de herramientas para la creación y sincronización de proyectos.



OTRAS HERRAMIENTAS:



Office 365 para las tareas de documentación y ofimática. Office 365 incluye un conjunto de programas como editor de texto, de hojas de cálculo, de correo (Word, Excel, Outlook)



Mozilla Firefox es un navegador web multiplataforma desarrollado por la compañía Corporación Mozilla. Para la búsqueda de recursos y la gestión de Firebase, GitHub y Office 365



StarUML **StarUML** es una herramienta de modelado para creación de diagramas UML

3. 4. Gestión de riesgos

La gestión de riesgos es un proceso dentro del desarrollo del proyecto, que consiste en identificar y eliminar o reducir los posibles riesgos que puedan afectar al correcto desarrollo y ejecución de las funciones del proyecto.

Riesgos que puedan provocar retrasos en la planificación / Soluciones:

-Falta de experiencia con algunas funciones a implementar en la aplicación.	Investigación en web y foros y formación mediante tutoriales o cursos
-Riesgos derivados de la planificación-	Agilizar en la medida de lo posible el desarrollo de las tareas para acelerar el proceso de producción.
-Subtareas no planificadas.	Disponer del máximo tiempo posible, para poder hacer frente al desarrollo de estas subfunciones.
-Nuevos requisitos a implementar	Prever los posibles y futuros requisitos a implementar, por si surge la necesidad de desarrollarlos, saber cómo enfocarlos.
-Cambios que puedan afectar a la estructura del proyecto.	Prever las necesidades en la estructura del proyecto, para no tener que realizar cambios que puedan afectar a toda la estructura, o que los cambios sean mínimos.
-Fuerzas mayores	Disponer del máximo tiempo posible, para poder hacer frente a estos imprevistos de fuerza mayor.

4. DISEÑO DE LA SOLUCIÓN

4. 1. Diseño de la interfaz de usuario y prototipos

La **interfaz de usuario** es el medio a través del cual el usuario interactúa con el sistema o dispositivo. El objetivo de esta interacción es permitir el funcionamiento y control más efectivo de la máquina desde la interacción con el usuario. El objetivo del diseño de una interfaz es producir una interfaz que sea fácil de usar, eficiente y agradable para cualquier usuario.

En pocas palabras, es el diseño del aspecto visual que tendrá nuestra aplicación y de cómo el usuario interactuará con el sistema.

El diseño de **prototipos** nos será útil para la representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final.

El prototipo ha sido realizado mediante Figma.



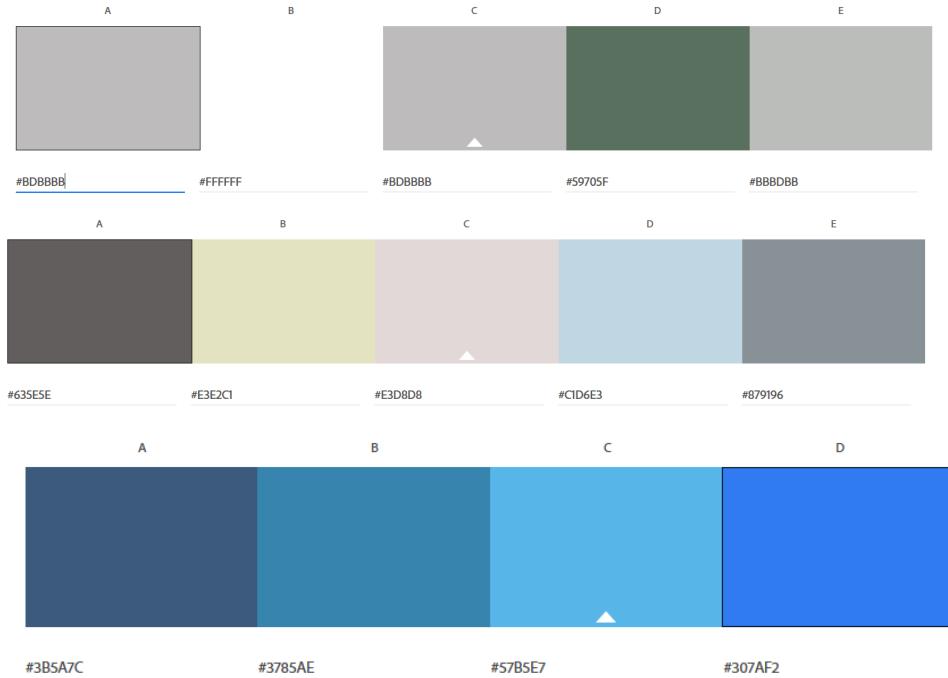
Enlace a YouTube con el video del prototipo:

<https://youtu.be/GgLhKMeROOY>

ASPECTO DE LA APLICACIÓN

En este apartado se expondrá la paleta de colores que se ha escogido para esta aplicación.

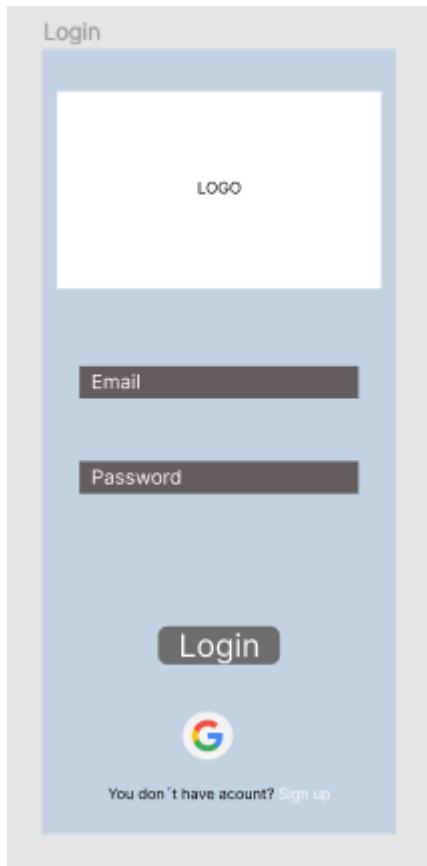
La aplicación pretende tener un estilo “Urban”, por ese motivo la paleta de colores escogida para esta aplicación, es principalmente de tonos grises y secundariamente de tonos azules y azul marino.



Colores Auxiliares



Capturas del prototipo



Al acceder a la aplicación el usuario tendrá que hacer login o registrarse como nuevo usuario

The image displays two side-by-side wireframe mockups of a sign-up interface. Both versions are titled "Sign up".

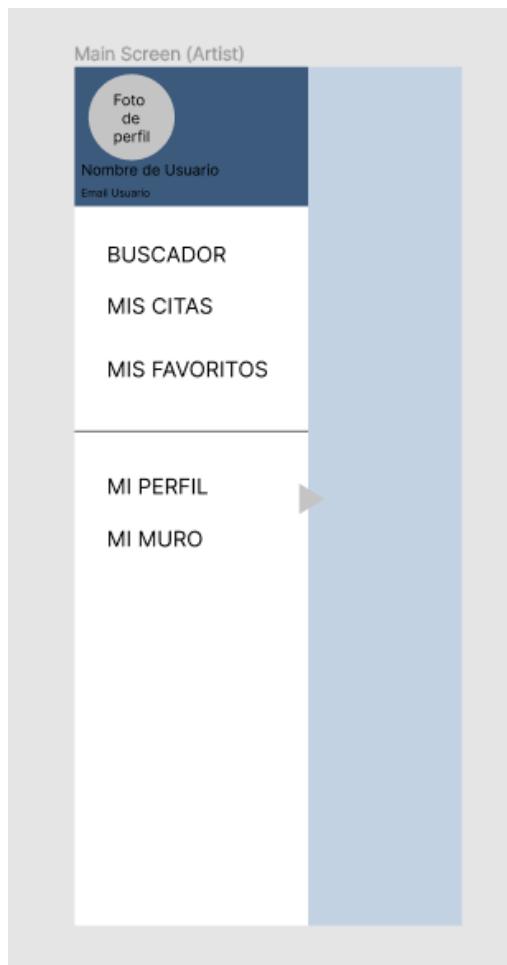
Left Mockup (Artist Role):

- Profile Photo placeholder: "foto de perfil"
- User Name input field
- Email input field
- Password input field
- Repeat Password input field
- Phone input field
- CIF input field
- Location input field: "Ubicación"
- Sign Up button

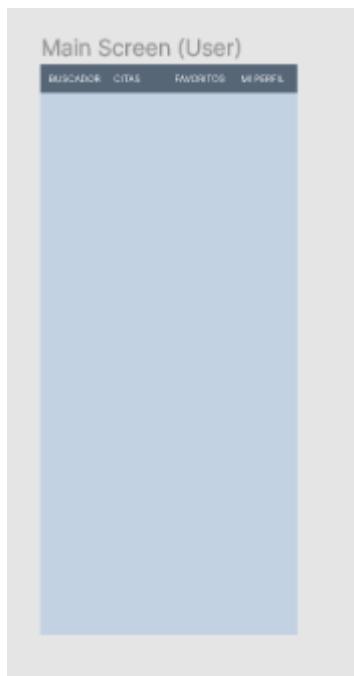
Right Mockup (General Role):

- Profile Photo placeholder: "foto de perfil"
- User Name input field
- Email input field
- Password input field
- Repeat Password input field
- Phone input field
- Sign Up button

Puede escoger el rol de usuario que con el que se va a registrar, y cada opción navega a una ventana diferente.

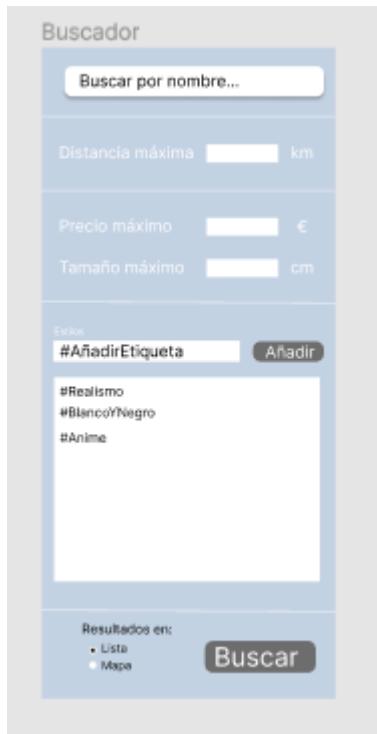


Una vez hecho el registro o el login, se accederá (desde el usuario artista) a un NavigationDrawer.



En el caso del usuario artista se accederá a un TabActivity.

Ambos casos constan de formas de navegación con la intención de simplificar el usuario básico.



En ambos casos se mostrará principalmente la ventana del buscador, ya que es la principal función de la aplicación.

Con el buscador se pueden realizar diversas búsquedas aplicando diferentes filtros. El usuario puede seleccionar también si quiere ver las resultados en modo lista o en modo mapa.

Resultados (Lista)

Foto de perfil

Nombre Usuario Artista
A 2 km de tu ubicación

Foto de perfil

Nombre Usuario Artista
A 5 km de tu ubicación

Foto de perfil

Nombre Usuario Artista
A 10 km de tu ubicación

Foto de perfil

Nombre Usuario Artista
A 15 km de tu ubicación

Foto de perfil

Nombre Usuario Artista
A 20 km de tu ubicación

(Modo lista)

Tras hacer click en uno de los artistas mostrados en la lista de resultados, se navega hasta la ventana del muro del artista.

Perfil Artista

foto de perfil

Nombre Usuario Artista

Ubicación del artista

Sitio web

favorito

SOLICITAR CITA

TABLA DE PRECIOS

10cm X 10 cm → 50€	COLOR 10cm X 10 cm → 70€
20cm X 20 cm → 100€	COLOR 20cm X 20 cm → 110€
30cm X 30 cm → 140€	COLOR 30cm X 30 cm → 160€

FOTO FOTO FOTO

FOTO FOTO FOTO

FOTO FOTO FOTO

FOTO FOTO FOTO

Desde este muro se puede dar al botón de favoritos (se añade a favoritos y aparecerá en su apartado). También se pueden ampliar las imágenes subidas por el artista. Y por último se puede contactar con el artista abriendo una ventana de chat y añadiéndolo al apartado de citas/chat.

En caso de ser artista usuario, contara con un apartado para su muro y poder editarlo o añadir fotos

Originalmente se planteó escoger alguna **tipografía** que tuviera que ver con la temática de la aplicación y resaltase, pero tras plantearse varias opciones se escogió utilizar la tipografía por defecto de Android. Esto principalmente es a causa de un motivo, ya que tras comparar varias aplicaciones profesionales se ha llegado a la conclusión de que cada vez se usa menos las tipografías que destacan mucho pudiendo parecer “orteras”. Por tanto, se ha escogido la opción más simple y agradable para el usuario medio.

4. 2. Diagrama de clases

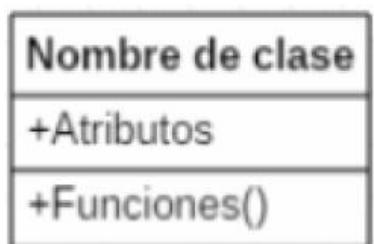
Es un tipo de diagrama de estructura que describe los elementos que deben estar presentes en el sistema que se está desarrollando. Se compone de clases, instancias e interfaces, y de las relaciones y asociaciones que hay entre estos elementos.

CLASES

Son el elemento principal de los diagramas de clases y representan una clase dentro del paradigma de la orientación a objetos. Cada clase es una plantilla para la creación de objetos. Cada objeto creado a partir de la clase se conoce como instancia de la clase.

Las clases están compuestas por nombre de la clase, atributos y funciones. Se representan con una caja dividida en tres zonas:

- La zona superior el nombre de la clase.
- La zona intermedia es para los atributos de la clase.
- La zona inferior es para las funciones de la clase.



Cada clase tiene un modificador de acceso. Son signos que se establecen delante del nombre de cada clase y nos especifican su nivel de acceso.

- | | |
|---|-----------|
| + | Público |
| - | Privado |
| # | Protegido |

/ Derivado

~ Paquete

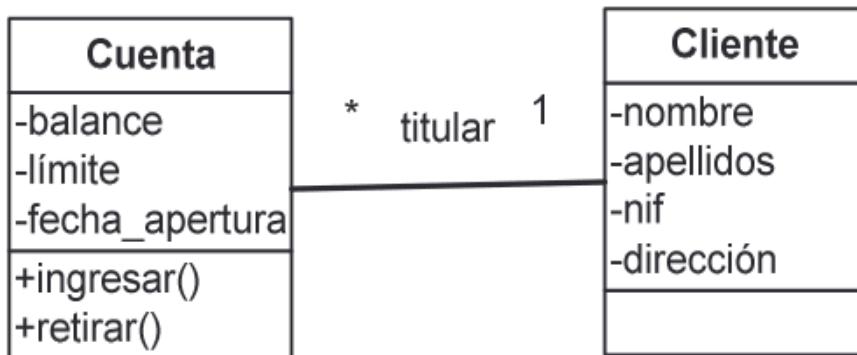
Subrayado Estático

RELACIONES

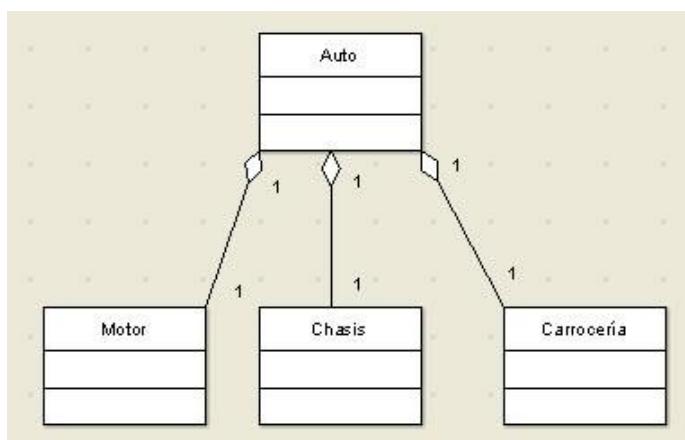
Una relación identifica una dependencia entre dos o más clases, o de una clase hacia sí misma. Las relaciones se representan con una línea que une las clases.

Los tipos de relaciones son:

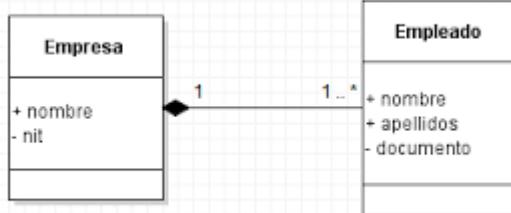
-**Asociación:** Es el más común y sirve para expresar conexión entre clases. Lo más común es que la “navegación” en las relaciones de asociación suela ser bidireccional.



-**Agregación:** Este tipo de relación representa situaciones en las que una clase está compuesta por uno o varios objetos y los pates que los componen. Se representa con un rombo vacío en el extremo de la línea que representa la relación, este rombo esta junto a la clase que representa la agregación de la otra.



-Composición: Es similar al tipo anterior, pero en este caso, un objeto está compuesto por otro u otros objetos. Se representa con un rombo relleno de color negro.



-Dependencia: En este tipo de relaciones hay una clase que requiere de otra para ofrecer sus funcionalidades. Se representa con una flecha con la línea discontinua.



-Herencia: Estas se utilizan para relacionar dos o más clases, una de ellas es la clase “padre” y las demás son las clases “hijas”. En este tipo de relaciones se heredan los atributos y métodos que tenga el padre, y los tienen las clases hijas junto con sus otros atributos y métodos. Se representa mediante una línea continua acabada en una flecha cerrada junto a la clase padre.

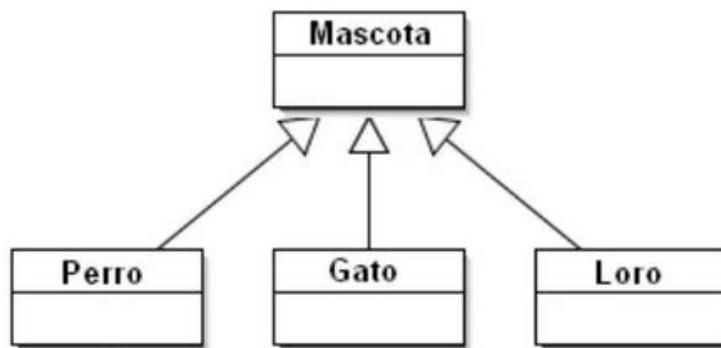
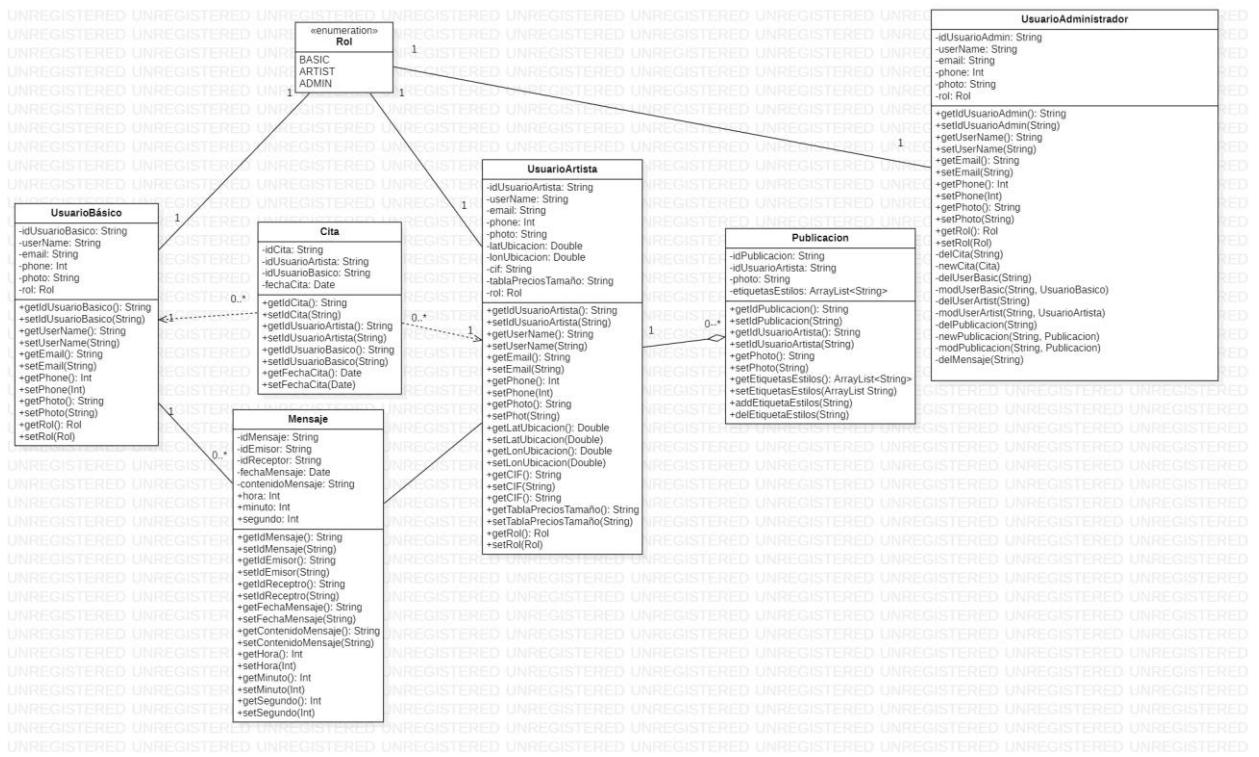


DIAGRAMA DE CLASES

El diagrama de nuestra aplicación es el siguiente



Descripción del diagrama

-**Rol:** Esta clase enumerada nos indica los tres posibles roles que hay en la aplicación.

UsuarioBasico - Rol : La cardinalidad es de 1 -> 1

UsuarioArtista - Rol : La cardinalidad es de 1 -> 1

-**UsuarioBasico:** Esta clase es la encargada de almacenar todos los datos relacionados con los usuarios básicos. Tendrá las siguientes relaciones.

UsuarioBasico - Rol : La cardinalidad es de 1 -> 1

UsuarioBasico- Cita: La cardinalidad es de 1 -> 0..*

UsuarioBasico – Mensaje: La cardinalidad es de 1 -> 0..*

-**UsuarioAdministrador:** Esta clase es la encargada de almacenar todos los datos relacionados con los usuarios básicos. Tendrá las siguientes relaciones.

UsuarioAdministrador - Rol : La cardinalidad es de 1 -> 1

-**UsuarioArtista:** Esta clase es la encargada de almacenar todos los datos relacionados con los usuarios artistas. Tendrá las siguientes relaciones.

UsuarioArtista - Rol : La cardinalidad es de 1 -> 1

UsuarioArtista - Cita: La cardinalidad es de 1 -> 0..*

UsuarioArtista – Mensaje: La cardinalidad es de 1 -> 0..*

UsuarioArtista – Publicacion: La cardinalidad es de 1 -> 0..*

-Cita: Esta clase es la encargada de almacenar todos los datos relacionados con las citas que se concretan entre UsuarioArtista y UsuarioBasico. Tendrá las siguientes relaciones.

UsuarioBasico- Cita: La cardinalidad es de 1 -> 0..*

UsuarioArtista - Cita: La cardinalidad es de 1 -> 0..*

-Mensaje: Esta clase es la encargada de almacenar todos los mensajes de texto que se envíen entre UsuarioArtista y UsuarioBasico cuando abren un chat de conversación. Tendrá las siguientes relaciones.

UsuarioBasico- Mensaje: La cardinalidad es de 1 -> 0..*

UsuarioArtista - Mensaje: La cardinalidad es de 1 -> 0..*

-Publicacion: Esta clase es la encargada de almacenar todos los datos relacionados con cada Publicación que realice un UsuarioArtista en su muro público. Tendrá las siguientes relaciones.

UsuarioArtista - Publicacion: La cardinalidad es de 1 -> 0..*

4. 2. Diagrama de persistencia de la información

La persistencia de datos es la capacidad de almacenar y recuperar datos de los objetos que se usarán en nuestro sistema. Para que este proceso funcione correctamente se suelen utilizar bases de datos o archivos de serialización de datos.

En lo que se refiere a bases de datos, vamos a hacer una pequeña aproximación hacia las diferencias que hay entre las bases de datos SQL y las NoSQL. Y tras esa breve aproximación escogeremos una de las dos opciones para nuestro proyecto.

Diferencia SQL y NoSQL

Existen varias diferencias entre ambos tipos de bases de datos, pero aquí vamos a hacer referencia a las más destacables:

	SQL	NoSQL
RENDIMIENTO	Menor rendimiento	Mayor rendimiento

FIABILIDAD	Mayor fiabilidad	Menor fiabilidad
DISPONIBILIDAD	Igual de validas	Igual de validas
ALMACENAMIENTO	Para manejar menor cantidad de datos	Para mayor cantidad de volumen de datos
ESCALABILIDAD	Costosas económicamente para manejar su escalabilidad. Escalado Vertical (aumenta los recursos del servidor)	Fácilmente de manejar su escalabilidad. Escalado Horizontal (aumenta el número de servidores)

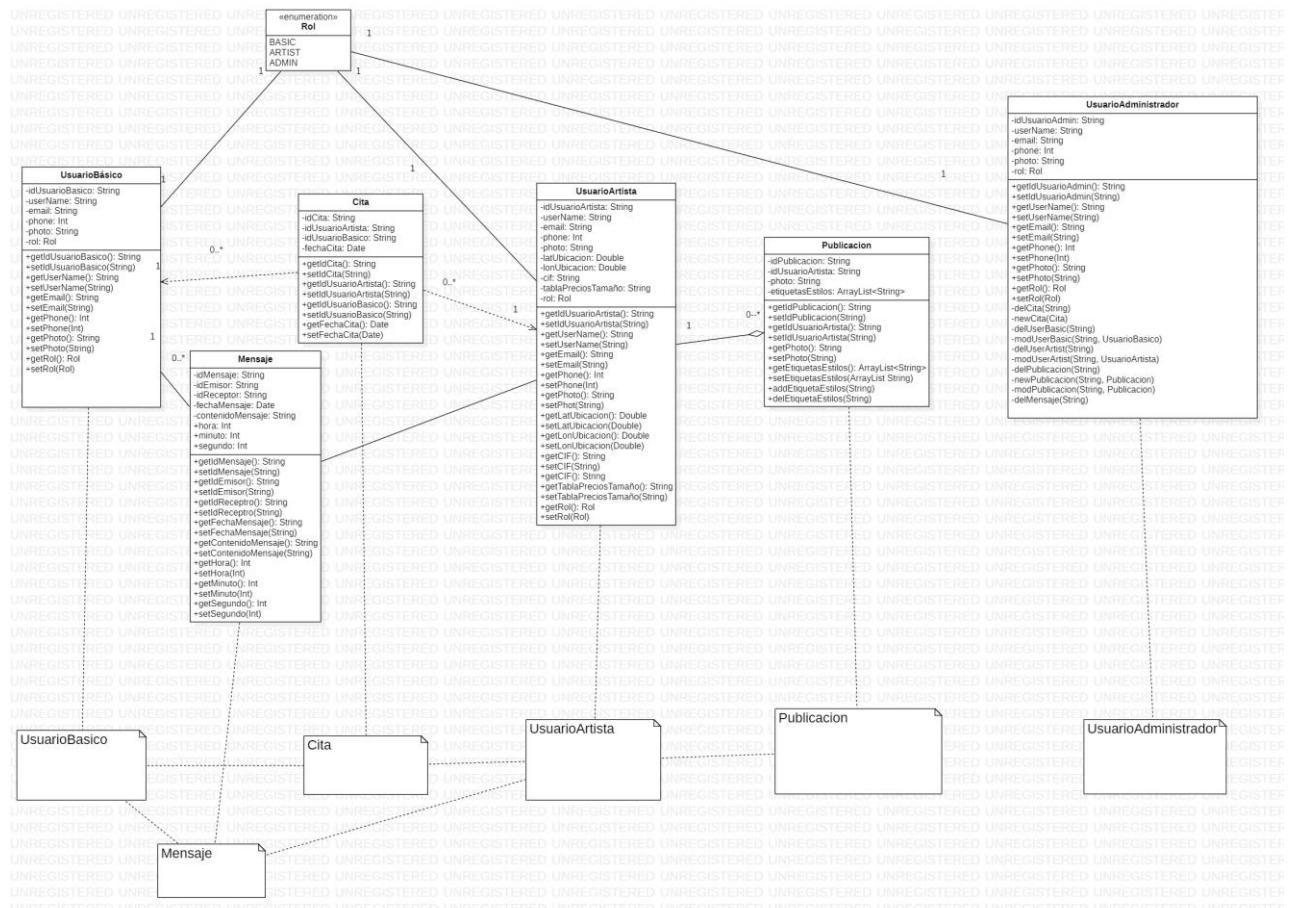
Base de datos NoSQL

En el caso de nuestro proyecto escogemos NoSQL por varios motivos:

- Por haber escogido Firebase (NoSQL) para desplegar este proyecto.
- Nuestro presupuesto es reducido por lo que no se pueden permitir el uso de grandes servidores.
- El crecimiento y la escalabilidad es más fácil de controlar.
- Es ideal para almacenar grandes volúmenes de datos en caso de necesidad del crecimiento.

Diagrama de persistencia

En este diagrama se muestra como está estructurada y organizada la información en la base de datos NoSQL en nuestro proyecto.



4. 4. Arquitectura del sistema

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema. Este tipo de arquitectura indica la estructura, funcionamiento e interacción entre las distintas partes del software.

- Integra el conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código.
- Se diseña en base a objetivos prefijados para el sistema y limitaciones derivadas de las tecnologías disponibles.
- Define los componentes que llevan a cabo las tareas en conjunto y la comunicación entre los mismos.

ARQUITECTURA ESCOGIDA PARA ESTE PROYECTO

Para este proyecto se ha decidido utilizar Firebase, por lo que tendremos un tipo de **Arquitectura en tiempo real**. También se podría definir como **Arquitectura Orientada a Servicios** o **Arquitectura Orientada a Servicios Web**.

Para este tipo de arquitectura en tiempo real los datos han de verse reflejados al instante, sin necesidad de refrescar. Firebase ofrece servicios de consulta de datos en tiempo real. Un ejemplo en nuestro proyecto serán los mensajes de texto que se intercambien en el chat entre el Usuario Básico y el Usuario Artista, estos mensajes son datos que se reflejan en tiempo real.

En lo que se refiere a arquitectura orientada a servicios, Firebase nos ofrece varios servicios que usará esta aplicación. Un usuario consumirá el servicio de autenticación de Firebase cuando quiera entrar a nuestra aplicación, consumirá servicios de base de datos (FireStore Database) y podrá subir imágenes al servicio de almacenamiento (Storage).

La arquitectura orientada a servicios web la podemos ver reflejada en Firebase en un aspecto similar al que se expone en una API Rest, con la diferencia de que Firebase dispone de librerías para diversos lenguajes de programación, con las que se pueden realizar estas peticiones.

En lo que a autenticación en la aplicación se va utilizar el sistema de autenticación de Firebase, pero para las funciones de Login, se utilizará firebase para la comprobación de los usuarios registrados, permitiendo el login a usuarios clientes, usuarios tatuadores y usuarios administradores.

Por lo tanto, podemos ver que tenemos un cliente (aplicación Android) y un Backend (Firebase) con estos servicios:

- Autenticación: Un usuario utiliza el servicio de autenticación de Firebase utilizando su email y contraseña.
- FireStore Database: Se trata del servicio de base de datos de Firebase. Cuando se actualice algún dato por parte de un Usuario Básico o un Usuario Artista, se registrará en base de datos NoSQL.

-Storage: Este servicio lo utilizaremos para almacenar imágenes. Principalmente podrán almacenar imágenes los Usuarios Artistas que suban imágenes a su muro público, pero este servicio también será consumido por los Usuarios Básicos al ver las imágenes en dichos muros.

5. IMPLEMENTACIÓN DE LA SOLUCIÓN

5. 1. Análisis tecnológico

Para entrar en la fase del desarrollo e implementación de la solución debemos plantearnos qué tecnologías vamos a usar.

Originalmente se plantearon muchas posibilidades a la hora de escoger qué tecnología se usaría para desarrollar este proyecto.

El motivo por el que se escogieron las tecnologías que se muestran a continuación es debido al conocimiento y experiencia previa que se tiene sobre ellas. También se muestran otras tecnologías que se han pensado para su implementación a largo plazo debido a la continuidad que se plantea sobre este proyecto.

Análisis tecnológico

Android Studio compilando Kotlin. La principal tecnología que se está usando para desarrollar este proyecto es Android Studio, utilizando el lenguaje de programación Kotlin. Android Studio es un potente Entorno de Desarrollo Integrado, basado en IntelliJ IDEA. Es el IDE oficial para desarrollar apps para Android.

Para la parte relacionada con el cliente también se ha planteado **Kotlin Native**. Esta tecnología permite compilar el código en Kotlin directamente a archivos binarios nativos, por lo que esta tecnología puede ser utilizada sin necesidad de una máquina virtual (como Java). Actualmente ofrece soporte para plataformas como iOS, MacOS, Android, Windows, Linux y WebAssembly.

Android-Kotlin nos permite compilar directamente archivos binarios nativos, por lo que es una tecnología que se puede utilizar sin necesidad de máquina virtual. Se ha escogido utilizar Kotlin debido a su facilidad a la hora de implementar diversas soluciones en el código fuente.

Para la tecnología de la parte servidor o Backend, se ha decidido utilizar **Firebase**. Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Se ha decidido utilizar esta tecnología debido a su gran utilidad y facilidad de incorporación al proyecto. Las principales ventajas que ofrece como tecnología Backend es su facilidad para almacenar y consumir datos desde el cliente Android. Las tecnologías (o servicios) que utilizamos de Firebase son:

-Autenticación: Nos permite de forma fácil, incorporar a nuestra aplicación una forma de registrar usuarios mediante un correo y una contraseña, además de poder comprobar y permitir el acceso a nuestra aplicación a usuarios registrados.

Firestore Database: Este servicio lo usaremos para almacenar todos los datos relevantes de la aplicación. Principalmente serán datos relacionados con los Usuarios Básicos, Usuarios Artistas, con las citas entre ambos, etc.

Storage. Este servicio nos permite almacenar imágenes en Firebase. Principalmente almacenaremos las imágenes que suban los artistas a sus muros públicos.

Firebase ha sido escogida para este proyecto, no solo por su facilidad de incorporación, también por su agilidad, su escalabilidad y su extensa documentación.

-Otras tecnologías complementarias que se usan para el desarrollo de este proyecto son:

Git y Git Kraken : El uso de Git como control de versiones nos facilita el desarrollo de este proyecto, pudiendo llevar un exhaustivo control de versiones y evitando posibles problemas en el desarrollo. Git Kraken es una herramienta para poder gestionar el control de versiones mediante una interfaz gráfica que facilita el uso de Git.

StartUML: Esta tecnología se ha utilizado para realizar los diagramas de casos de uso y diagramas de clase necesarios para los pasos previos al desarrollo de esta aplicación.

Figma: Es una herramienta online con la que se han realizado el diseño de prototipos previos.

Se plantean también estos servicios de Firebase para su posible implementación en el proyecto:

Realtime Database: Consiste en el servicio de base de datos en tiempo real que ofrece Firebase. Estas bases de datos son NoSQL y se alojan en la nube de Firebase. Firebase envía de forma automática eventos a las aplicaciones cuando detecta cambios en los datos.

Cloud Messaging: Este servicio de Firebase consiste en el envío en tiempo real de notificaciones y mensajes a los usuarios.

-Otras tecnologías que se han planteado para su uso en este proyecto son:

Lucene : Es una API de código abierto para recuperación de información. Es considerado un indexador de contenido. Esta tecnología podría resultar muy útil para fortalecer el motor de búsqueda de esta aplicación. Se plantea su incorporación medio- largo plazo en caso de necesidades de crecimiento y escalabilidad de la aplicación.

Migración de la aplicación Android a iOS. Para ello se plantea el uso de compiladores como **LLVM Compiler**. También puede ser viable usar un emulador iOS con coste de 25€. O incluso se puede plantear la opción de una Máquina Virtual incorporada en la aplicación, o dockerizar la aplicación.

En esta fase del proyecto primero se ha de ver el impacto que tiene esta primera versión, por lo que solo se plantean la posible implementación de estas dos tecnologías a corto/medio plazo. Indexador de contenidos para potenciar el motor de búsqueda y migración a iOS para que no se estanque solo en Android.

Este proyecto tiene perspectivas de crecimiento e incorporación de más desarrolladores, ampliación de funcionalidades, uso de nuevas tecnologías y conversión a otras plataformas además de Android, por lo que se valorará más adelante qué tecnologías pueden ser útiles o necesarias para esta futura ampliación

5. 2. Elementos a implementar

En este apartado se van a mostrar las funcionalidades que se han implementado:

- Registro y login (registro por parte de usuarios básicos y usuarios artistas)
- Buscador de artistas con filtros de nombre o distancia máxima
- Listar artistas favoritos
- Muro público de artistas
- Editar, eliminar perfil (usuario básico y usuario artista).
- Subir publicación al muro del artista
- Chat para concertar la cita entre usuario básico y usuario artistas
- Buscador según filtros de precios o estilos
- Resultados del buscador en modo mapa

Procedemos por tanto a detallar los aspectos más relevantes de cada parte.

Aspectos relevantes del login y registro

La aplicación comenzará desde la ventana de **login**. En caso de que el usuario este registrado podrá realizar el login utilizando el servicio de autenticación de Firebase.

La interfaz consta de dos cajas de texto (email y contraseña) un botón para login, otro para registro con Google y un link para navegar a la ventana de registro. En el momento de hacer el login, lo primero a destacar es que primero comprueba la autenticación con Firebase y después comprueba si el usuario es básico o usuario artista

```
private fun checkLogin(){  
    if(checkCamposVacios()){  
        val email = edTxtEmailLogin.text.toString()  
        val pwd = edTxtPwdLogin.text.toString()  
  
        FirebaseAuth.getInstance().signInWithEmailAndPassword(email,pwd).addOnCompleteListener { it: Task<AuthResult!>  
            if (it.isSuccessful){  
                VariablesCompartidas.emailUsuarioActual = (it.result?.user?.email?: "")  
                isBasicUser = false  
                findUserByEmail(email)  
  
                if(!isBasicUser){  
                    findArtistUserByEmail(email)  
                }  
  
                } else {  
                    Toast.makeText( context: this,"Login ERROR" , Toast.LENGTH_SHORT).show()  
                }  
            }  
        }else{  
            Toast.makeText( context: this,"Complete all fields" , Toast.LENGTH_SHORT).show()  
        }  
    }  
}
```

Busca al usuario básico

```

private fun findUserByEmail(email: String){

    db.collection( collectionPath: "${Constantes.collectionUser}")
        .whereEqualTo( field: "email", email)
        .get()
        .addOnSuccessListener { users ->
            //Existe
            for (user in users) {
                var phone = 0;
                if (user.get("phone").toString() != ""){
                    phone = user.get("phone").toString().toInt()
                }
                var us = BasicUser(
                    user.get("userId").toString(),
                    user.get("userName").toString(),
                    user.get("email").toString(),
                    phone,
                    user.get("img").toString(),
                    user.get("rol") as ArrayList<Rol>?,
                    user.get("idFavoritos") as ArrayList<String>?
                )
            }
            VariablesCompartidas.usuarioBasicoActual = us
            VariablesCompartidas.usuarioArtistaActual = null
            isBasicUser = true
            var myIntent = Intent( packageContext: this, TabBasicUserActivity::class.java)
            startActivity(myIntent)
        }
    }
    .addOnFailureListener { exception ->
        //No existe
        isBasicUser = false
        //Log.w(ContentValues.TAG, "Error getting documents: ", exception)
    }
}

```

Busca al usuario artista

```

private fun findArtistUserByEmail(email: String){
    Toast.makeText(context, email, Toast.LENGTH_SHORT).show()
    db.collection(collectionPath: ${Constantes.collectionArtistUser})
        .whereEqualTo(field: "email", email)
        .get()
        .addOnSuccessListener { users ->
            for (user in users) {
                var phone = 0;
                if (user.get("phone").toString() != ""){
                    phone = user.get("phone").toString().toInt()
                }
                var us = ArtistUser(
                    user.get("userId").toString(),
                    user.get("userName").toString(),
                    user.get("email").toString(),
                    phone,
                    user.get("img").toString(),
                    user.get("rol") as ArrayList<Role>?,
                    user.get("idFavoritos") as ArrayList<String>?,
                    user.get("cif").toString(),
                    user.get("latitudUbicacion").toString().toDouble(),
                    user.get("longitudUbicacion").toString().toDouble(),
                )
            }
            VariablesCompartidas.usuarioArtistaActual = us
            VariablesCompartidas.usuarioBasicoActual = null
            isBasicUser = true
            var myIntent = Intent(packageContext, BasicUserNavDrawActivity::class.java)
            startActivity(myIntent)
        }
    }
    .addOnFailureListener { exception ->
        //No existe
        isBasicUser = false
        Log.d(tag: "CHE_TAG", msg: "Error getting documents: ", exception)
    }
}

```

Con este bloque de código se permite el login y registro con Google. Se ha decidido que el usuario que utilice esta forma de registro, se registrará con usuario Básico.

```

private fun check_login_google(){
    val googleConf = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken("694797645424-f142hiec3bdj3ml09nbjq0m7cqbtbtmp.apps.googleusercontent.com") //Este se encuentra en el archivo google-services.json: client->
        .requestEmail()
        .build()

    val googleClient = GoogleSignIn.getClient(this,googleConf) //Este será el cliente de autenticación de Google.
    googleClient.signInIntent //Con esto salimos de la posible cuenta de Google que se encuentre logueada.
    val signInIntent = googleClient.signInIntent
    startActivityForResult(signInIntent, RC_SIGN_IN)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    // Si la respuesta de esta actividad se corresponde con la inicializada es que viene de la autenticación de Google.
    if (requestCode == RC_SIGN_IN) {
        val task = GoogleSignIn.getSignedInAccountFromIntent(data)
        try {
            // Google Sign In was successful, authenticate with Firebase
            val account = task.getResult(ApiException::class.java)!!

            //Ya tenemos la id de la cuenta. Ahora nos autenticamos con Firebase.
            if (account != null) {
                val credential: AuthCredential = GoogleAuthProvider.getCredential(account.idToken, null)
                auth.signInWithCredential(credential).addOnCompleteListener { it: Task<AuthResult> -
                    if (it.isSuccessful) {
                        val user = auth.currentUser!!
                        VariablesCompartidas.emailUsuarioActual = user.email!!
                        regUser(account)
                        findUserByEmail(user.email!!)
                    } else {
                        Utils.showAlert(context)
                    }
                }
            }
            //firebaseAuthWithGoogle(account.idToken!!)
        } catch (e: ApiException) {
            //Handle sign in failed - update UI accordingly
        }
    }
}

```

Una vez se realice el login de forma correcta, se navegará a un NavigationDrawer para el usuario artista y un TabActivity para el usuario básico, ambos se mostrarán más adelante.

Continuemos por detallar el registro. En cuanto se haga click en el Link para registrarse, se pregunta al usuario si se desea registrar como usuario básico o usuario artista. Cada opción navega a un Activity de SignUp diferente en función del tipo de usuario seleccionado.

En caso del **registro de un usuario básico** nos encontramos con una interfaz que contiene una imagen redondeada, cinco cajas de texto (Nombre de usuario, email, contraseña, repetir contraseña y teléfono) y un botón para confirmar.

Al hacer click en el círculo de la imagen, seleccionar una foto para el perfil desde la galería o la cámara.

```
fun cambiarFoto() {
    AlertDialog.Builder(context)
        .setTitle("Choose photo")
        .setMessage("How do you want to add the photo?")
        .setPositiveButton("Camera") { view, _ ->
            hacerFoto()
            view.dismiss()
        }
        .setNegativeButton("Gallery") { view, _ ->
            elegirDeGaleria()
            view.dismiss()
        }
        .setCancelable(true)
        .create()
        .show()
}

private fun elegirDeGaleria() {
    val intent = Intent()
    intent.type = "image/*"
    intent.action = Intent.ACTION_GET_CONTENT
    startActivityForResult(
        Intent.createChooser(intent, title = "Seleccione una imagen"),
        Constantes.CODE_GALLERY
    )
}

private fun hacerFoto() {
    if (ContextCompat.checkSelfPermission(
            context,
            Manifest.permission.CAMERA
        ) == PackageManager.PERMISSION_DENIED
    ) {
        ActivityCompat.requestPermissions(
            this as AppCompatActivity,
            arrayOf(Manifest.permission.CAMERA),
            Constantes.CODE_CAMERA
        )
    }
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    startActivityForResult(intent, Constantes.CODE_CAMERA)
}
```

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    when (requestCode) {
        Constantes.CODE_CAMERA -> {
            if (resultCode == Activity.RESULT_OK) {
                photo = data?.extras?.get("data") as Bitmap
                imgSignUpUser.setImageBitmap(photo)
                photoSt = Utils.ImageToString(photo)
            }
        }
        Constantes.CODE_GALLERY -> {
            if (resultCode === Activity.RESULT_OK) {
                val selectedImage = data?.data
                val selectedPath: String? = selectedImage?.path
                if (selectedPath != null) {
                    var imageStream: InputStream? = null
                    try {
                        imageStream = selectedImage.let { Uri.parse(it) }
                        this.contentResolver.openInputStream(
                            it
                        )
                    } catch (e: FileNotFoundException) {
                        e.printStackTrace()
                    }
                    val bmp = BitmapFactory.decodeStream(imageStream)
                    photo = Bitmap.createScaledBitmap(bmp, dstWidth: 200, dstHeight: 300, filter: true)
                    imgSignUpUser.setImageBitmap(photo)
                    photoSt = Utils.ImageToString(photo)
                }
            }
        }
    }
}
```

Al realizar el registro se comprueba que todos los campos se hayan rellenado, que las contraseñas coincidan, que el formato de teléfono sea correcto y que el formato de email sea correcto.

```

private fun regUser(email: String){
    val id = UUID.randomUUID().toString()
    val rol = Rol( idRol: 1, nombreRol: "${Constantes.rolBasicUser}" )
    var listRoles : ArrayList<Role> = ArrayList()
    var listIdFavoritos : ArrayList<String> = ArrayList()
    listRoles.add(rol)
    var img : String? = null
    if(photo != null){
        img = photoSt
    }

    var userName = edTxtUserNameSignUpUser.text.toString()
    var phone = edTxtPhoneSignUpUser.text.toString().toInt()

    //var user = BasicUser(id,userName,email,phone,img,listRoles,listIdFavoritos)
    var user = hashMapOf(
        "userId" to id,
        "userName" to userName,
        "email" to email,
        "phone" to phone,
        "img" to img,
        "rol" to listRoles,
        "idFavoritos" to listIdFavoritos
    )

    var u = BasicUser(id,userName,email,phone,img,listRoles,listIdFavoritos)
    VariablesCompartidas.usuarioBasicoActual = u
    VariablesCompartidas.usuarioArtistaActual = null

    db.collection( collectionPath: "${Constantes.collectionUser}" )
        .document(id)
        .set(user)
        .addOnSuccessListener { it:Void! ->
            val myIntent = Intent( packageContext: this, TabBasicUserActivity::class.java)
            startActivity(myIntent)

            //Toast.makeText(this,"GO MAIN", Toast.LENGTH_SHORT).show()
        }.addOnFailureListener{ it:Exception ->
            Toast.makeText( context: this, "ERROR" , Toast.LENGTH_SHORT).show()
        }
}

```

Tras haber realizado las comprobaciones se registra al usuario y se navega hasta el TabActivity principal para el usuario básico

En el caso del **registro de un usuario Artista** la ventana cuenta con una interfaz diferente. En este caso también tiene una imagen de perfil redondeada, pero tiene 6 cajas de texto, dos botones y un panel para mostrar un mapa. La caja de texto añadida es el CIF y el nuevo botón es para poder cambiar la ubicación del estudio de tatuajes.

La funcionalidad de cambiar la imagen es idéntica a la mostrada anteriormente.

En esta activity contamos con un Fragment que nos muestra en un mini mapa con el marcador de la ubicación que seleccione el usuario. Al hacer click en el botón de seleccionar la ubicación del estudio, navegamos a una ventana de GoogleMap, donde podemos seleccionar la ubicación.

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    when (requestCode) {
        Constantes.CODE_CAMERA -> {...}
        Constantes.CODE_GALLERY -> {...}
        Constantes.CODE_MAP -> {

            if(VariablesCompartidas.latitudStudioSeleccionado != null && VariablesCompartidas.longitudStudioSeleccionado != null){
                latitudStudio = VariablesCompartidas.latitudStudioSeleccionado.toString().toDouble()
                longitudStudio = VariablesCompartidas.longitudStudioSeleccionado.toString().toDouble()

                ubiActual = LatLng(latitudStudio!!,longitudStudio!!)

                val ubi = LatLng(VariablesCompartidas.latitudStudioSeleccionado.toString().toDouble(), VariablesCompartidas.longitudStudioSeleccionado.toDouble())
                mMap?.clear()
                mMap?.addMarker(MarkerOptions().position(ubi).title(" ${adTxtArtistUserName.text}"))
                mMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(ubi, 200m 15f))
            }
        }
    }
}

//+-----+
//*
Carga un Map en el fragment
*/
private fun cargarMapa() {
    val mapFragment = supportFragmentManager
        .findFragmentById(R.id.frm_MapLocation) as SupportMapFragment
    mapFragment.getMapAsync(callback: this)
}

override fun onMapReady(p0: GoogleMap) {
    mMap = p0!!
    mMap.mapType=GoogleMap.MAP_TYPE_HYBRID
    val Madrid = LatLng(latitude: 40.416, longitude: -3.703)
    mMap?.addMarker(MarkerOptions().position(Madrid).title("SELECT STUDIO LOCATION"))
    mMap?.moveCamera(CameraUpdateFactory.newLatLngZoom(Madrid, zoom: 10f))
}

```

Con las dos funciones de abajo, cargamos el minimapa al abrir la ventana de registro. Con la función de arriba, se actualiza el minimapa con la ubicación seleccionada tras cerrar la ventana a la que hemos navegado previamente

Tras realizar la comprobación de que se han rellenado todos los campos de texto, se ha seleccionado una ubicación, las contraseñas coinciden, el formato de teléfono y email son correcto, se realiza el registro.

```

private fun regArtistUser(email: String){
    val id = UUID.randomUUID().toString()
    val rol = Rol( idRol: 1, nombreRol: "${Constantes.rolArtistUser}" )
    var listRoles : ArrayList<Role> = ArrayList()
    listRoles.add(rol)
    var listIdFavoritos : ArrayList<String> = ArrayList()
    val cif = edTxtArtistCif.text.toString()
    var img : String? = ""
    if(photo != null){
        img = photoSt
    }

    var userName = adTxtArtistUserName.text.toString()
    var phone = adTxtArtistPhone.text.toString().toInt()

    var user = hashMapOf(
        "userId" to id,
        "userName" to userName,
        "email" to email,
        "phone" to phone,
        "img" to img,
        "rol" to listRoles,
        "idFavoritos" to listIdFavoritos,
        "cif" to cif,
        "latitudUbicacion" to latitudStudio,
        "longitudUbicacion" to longitudStudio
    )

    var u = ArtistUser(id,userName,email,phone,img,listRoles,listIdFavoritos,cif,latitudStudio,longitudStudio)
    VariablesCompartidas.usuarioArtistActual = u
    db.collection( collectionPath: "${Constantes.collectionArtistUser}" )
        .document(id)
        .set(user)
        .addOnSuccessListener { it:Void?
            val myIntent = Intent( packageContext: this, BasicUserNavDrawActivity::class.java)
            startActivity(myIntent)

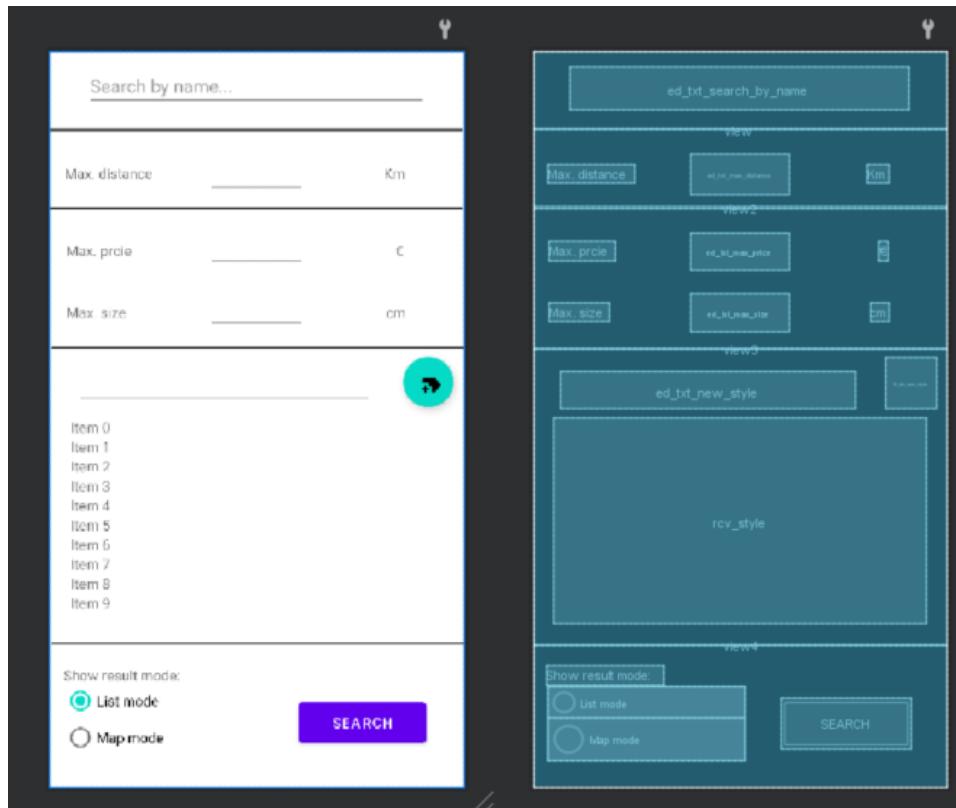
            //Toast.makeText(this,"GO MAIN", Toast.LENGTH_SHORT).show()
        }.addOnFailureListener{ it:Exception?
            Toast.makeText( context: this,"ERROR" , Toast.LENGTH_SHORT).show()
        }
}

```

Tras realizar el registro se navega a al NavigationDrawer para el Artista.

Aspectos relevantes del buscador

Tanto desde el TabActivity del usuario básico como en NavigationDrawer del usuario artista, podemos acceder a la ventana del buscador. La interfaz de esta ventana cuenta con una caja de texto para el nombre, otra para la distancia máxima, dos más para el precio y tamaño máximos, una caja de texto con lista para añadir estilos al os filtros de búsqueda, y por último dos radioButton para poder seleccionar la forma en que se mostrarán los resultados de la búsqueda (modo lista o modo mapa).



Al realizar la búsqueda se obtienen los datos de Firebase

```
private fun busqueda(){
    result.clear()
    auxResult.clear()
    finalResult.clear()
    runBlocking { thisCoroutineScope {
        val job = launch(context = Dispatchers.Default) { thisCoroutineScope {
            var datos : QuerySnapshot = getDataFromFireStore() as QuerySnapshot //Obtenemos la colección
            obtenerDatos(datos as QuerySnapshot?) //Destripamos la colección y la metemos en nuestro ArrayList
        }
        job.join() //Esperamos a que el método acabe: https://dzone.com/articles/waiting-for-coroutines
    }
    aplicarFiltros()
}}
```

```

suspend fun getDataFromFireStore() : QuerySnapshot? {
    return try{
        val data = db.collection( collectionPath: "${Constantes.collectionArtistUser}")
            .get()
            .await()
        data
    }catch (e : Exception){
        null
    }
}

private fun obtenerDatos(datos: QuerySnapshot?) {
    result.clear()
    for(dc: DocumentChange in datos?.documentChanges!!){
        if (dc.type == DocumentChange.Type.ADDED){

            var fav= ArtistUser(
                dc.document.get("userId").toString(),
                dc.document.get("userName").toString(),
                dc.document.get("email").toString(),
                dc.document.get("phone").toString().toInt(),
                dc.document.get("img").toString(),
                dc.document.get("rol") as ArrayList<String>,
                dc.document.get("idFavoritos") as ArrayList<String>?,
                dc.document.get("cir").toString(),
                dc.document.get("latitudUbicacion").toString().toDouble(),
                dc.document.get("longitudUbicacion").toString().toDouble()
            )
            result.add(fav)
        }
    }
}

```

Se aplican los distintos filtros

```

private fun aplicarFiltros() {
    //PRIMER FILTRO EL NOMBRE
    if(ed_txt_search_by_name.text.toString().trim().isNotEmpty()){
        val nombre = ed_txt_search_by_name.text.toString()
        for(artist in result){
            if(artist.userName!! .startsWith(nombre) || artist.userName!! .contains(nombre) || artist.userName.equals(nombre)){
                finalResult.add(artist)
            }
        }
    }
    if(ed_txt_max_distance.text.toString().trim().isNotEmpty()){
        aplicarFiltroDistancia()
    }

    if(checkAllEmpty()){
        val resultIntent = Intent(requireContext(), ListResulActivity::class.java)
        val args = Bundle()
        args.putSerializable("USER_LIST", result)
        resultIntent.putExtra( name:"BUNDLE", args)
        startActivity(resultIntent)
    }else{
        val resultIntent = Intent(requireContext(), ListResulActivity::class.java)
        val args = Bundle()
        args.putSerializable("USER_LIST", finalResult)
        resultIntent.putExtra( name:"BUNDLE", args)
        startActivity(resultIntent)
    }
}

```

Se obtiene la localización actual del usuario y se aplican los filtros de distancia.

```

private fun getMyLocation(): Location? {
    if (ActivityCompat.checkSelfPermission(
            requireContext(),
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
            requireContext(),
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED)
        return null
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location : Location? ->
            userLocation = location
        }
}

private fun calculateByDistance(location: Location?, lat2: Double?, lon2: Double?): Double {
    val Radius = 6371 // radius of earth in Km
    val lat1 = location!!.latitude
    val lon1 = location.longitude
    val dLat = Math.toRadians(lat2!! - lat1)
    val dLon = Math.toRadians(lon2!! - lon1)
    val a = (Math.sin(dLat / 2) * Math.sin(dLat / 2)
        + (Math.cos(Math.toRadians(lat1))
        * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2)
        * Math.sin(dLon / 2)))
    val c = 2 * Math.asin(Math.sqrt(a))
    val valueResult = Radius * c
    val km = valueResult / i
    val newFormat = DecimalFormat(pattern: "####")
    val kmInDec = Integer.valueOf(newFormat.format(km))
    val meter = valueResult % 1000
    val meterInDec = Integer.valueOf(newFormat.format(meter))
    Log.i("Radius Value", msg = "" + valueResult + " KM " + kmInDec
        + " Meter " + meterInDec)
    return Radius * c
}

```

Podemos **listar los resultados de búsqueda** que cumplan con los filtros los mostramos en una lista de un Adaptador de RecyclerView , en el que cada ítem tiene el nombre e imagen de usuario de cada artista.

Al seleccionar uno del ítem de la lista, navegamos hasta el muro público del artista en cuestión. Recordemos que un usuario artista también tiene un muro, es una de las funcionalidades que tiene, por lo que se controla en qué tipo de situación se está viendo el muro. Un usuario básico viendo un muro de un artista, un usuario artista viendo el muro de otro artista o un usuario artista viendo su propio muro.

```

private fun cargarDatosArtist() {
    if(VariablesCompartidas.idUserArtistVisitMode != null){
        userMuro = VariablesCompartidas.usuarioArtistaVisitaMuro
        if(userMuro!!.img != null){
            imgArtist.setImageBitmap(Utils.StringToBitmap(userMuro!!.img.toString()))
        }
        txtUserName.text = (userMuro!!.userName.toString())
        txtEmail.text = (userMuro!!.email.toString())
        btnContactEdit.setText("Contact")
        fltBtnFavCamera.visibility = View.VISIBLE
        checkFav()
        if(VariablesCompartidas.usuarioArtistaActual != null && userMuro!!.userId.equals(VariablesCompartidas.usuarioArtistaActual!!.userId)){
            checkFav()
            btnContactEdit.visibility = View.INVISIBLE
            //btnContactEdit.setText(R.string.edit_muro)
            fltBtnFavCamera.visibility = View.INVISIBLE
            //fltBtnFav.setImageResource(R.drawable.ic_menu_camera)
        }
    }
    if(VariablesCompartidas.idUserArtistVisitMode == null && VariablesCompartidas.usuarioArtistaActual != null){
        userMuro = VariablesCompartidas.usuarioArtistaActual
        txtUserName.setText(userMuro!!.userName.toString())
        btnContactEdit.setText("Edit")
        txtEmail.text = (userMuro!!.email.toString())
        fltBtnFavCamera.setImageBitmap(Utils.StringToBitmap(userMuro!!.img.toString()))
        imgArtist.setImageBitmap(Utils.StringToBitmap(userMuro!!.img.toString()))
        editMode = true
    }
}

```

Acto seguido se comprueba si el usuario que está visitando este muro, lo tiene entre sus favoritos o no.

Desde la ventana de **Favoritos** (también disponible para usuarios básicos y usuarios artista) se pueden **listar también los artistas favoritos** y navegar a su muro público al hacer click en un ítem de la lista.

Otra opción que tienen ambos tipos de usuario son la de gestionar su perfil, pero con las mismas diferencias que se han apreciado en el registro.

En el caso de **editar el perfil del usuario básico** vemos una imagen circular con la imagen de perfil, botón flotante para editar y tres cajas de texto (nombre, email y teléfono). Al hacer click en el botón de editar, se hace visible otro botón, con el que podremos cambiar la contraseña. También se habilitan las cajas de texto para su edición al igual que la imagen se nos permite cambiarla.

```
fun changePwd() {
    val dialog = LayoutInflater.inflate(R.layout.password_changer, root null)
    val pass1 = dialog.findViewById<EditText>(R.id.edPassChanger)
    val pass2 = dialog.findViewById<EditText>(R.id.edPass2Changer)
    AlertDialog.Builder(requireContext())
        .setTitle("Change Password")
        .setView(dialog)
        .setPositiveButton(text "OK") { view, _ ->
            val p1 = pass1.text.toString()
            val p2 = pass2.text.toString()
            if (p1 == p2) {
                currentUser.updatePassword(p1)
                Toast.makeText(
                    context,
                    "SUSCESFULL",
                    Toast.LENGTH_SHORT
                ).show()
            } else Toast.makeText(
                context,
                "ERROR",
                Toast.LENGTH_SHORT
            ).show()
            view.dismiss()
        }
        .setNegativeButton("CANCEL") { view, _ ->
            view.dismiss()
        }
        .setCancelable(false)
        .create()
        .show()
}
```

Tras haber modificado los campos deseados y volver a hacer click en el botón flotante (el cual había cambiado su icono y ahora tiene un icono de “save”) podremos guardar los cambios editados. Actualizando el usuario en Firebase

```

fun editar(){
    if(edTxtBasicUserEmail.text.trim().isNotEmpty() && edTxtBasicUserPhone.text.trim().isNotEmpty() && edTxtBasicUserName.text.trim().isNotEmpty() && Utils.checkMovil(edTxtBasicUserPhone.text))
        var email_mod = edTxtBasicUserEmail.text.toString().trim()
        var userName_mod = edTxtBasicUserName.text.toString().trim()
        var phone_mod = edTxtBasicUserPhone.text.toString().trim().toInt()

        photo = ImagenUsuarioPerfil.drawable
        val imgSt = Utils.ImageToString(photo!!)
        var basicUser = BasicUser(basicUserActual.userId,userName_mod,email_mod,phone_mod,imgSt,basicUserActual.rel,basicUserActual.idFavorites)

        db.collection( collectionName: ${Constantes.collectionUser} )
            .document(VariablesCompartidas.usuarioBasicicoActual!!._userId.toString()) //será la clave del documento.
            .set(basicUser).addOnSuccessListener {
                VariablesCompartidas.usuarioBasicicoActual = basicUserActual
                currentUser?.updateEmail(basicUser.email.toString())
            }
            .addOnFailureListener{ e:Exception ->
                Toast.makeText(requireContext(), "ERROR", Toast.LENGTH_SHORT).show()
            }
        }else{
            Toast.makeText(requireContext(),"Complete all fields", Toast.LENGTH_SHORT).show()
        }
}

```

En el caso de **editar el perfil del usuario artista** contamos con estas funcionalidades y además cuenta con la de un minimapa con la ubicación de su estudio.

Al hacer click en editar, se permite la edición de los campos de texto y se hace visible un botón que nos permite cambiar la ubicación del estudio. Si hacemos click en ese botón se nos abrirá una ventana de GoogleMap donde podremos seleccionar una nueva ubicación para el estudio. En esta ventana, haciendo click en el mapa se crean marcadores, al seleccionar un marcador escogemos la ubicación para el estudio.

```

override fun onMarkerClick(p0: Marker): Boolean {

    AlertDialog.Builder( context: this).setTitle("Select this location?")
        .setPositiveButton("CONFIRM") { view, _ ->
            //guardamos la posí del marcador para usarla en otra activity
            VariablesCompartidas.marcadorActual = p0.position
            VariablesCompartidas.latitudStudioSeleccionado = p0.position.latitude.toString()
            VariablesCompartidas.longitudStudioSeleccionado = p0.position.longitude.toString()
            finish()
            view.dismiss()
        }.setNegativeButton("NO") { view, _ ->
            //elimina marcador
            p0.remove()
            view.dismiss()
        }.create().show()
    return false
}

```

Al cerrar esta ventana de GoogleMaps, volvemos a la ventana de edición de perfil del usuario artista, y se nos ha guardad la ubicación seleccionada, para modificar la anterior.

Tras editar todos los cambios podemos guardar el usuario editado.

```

fun editar(){
    if(edTxtArtistUserEmail.text.trim().isEmpty() && edTxtArtistUserPhone.text.trim().isEmpty() && edTxtArtistUserName.text.trim().isEmpty())
        return

    val email_mod = edTxtArtistUserEmail.text.toString().trim()
    val userName_mod = edTxtArtistUserName.text.toString().trim()
    val phone_mod = edTxtArtistUserPhone.text.toString().trim().toInt()
    val cif = edTxtArtistUserCif.text.toString().trim()
    val lat = VariablesCompartidas.latitudStudioSeleccionado.toString().toDouble()
    val lon = VariablesCompartidas.longitudStudioSeleccionado.toString().toDouble()
    photo = imgUsuarioPerfil.drawableToBitmap()
    val imgSt = Utils.ImageToString(photo!!)
    val artistUser = ArtistUser(artistUserActual.userId,userName_mod,phone_mod,imgSt,artistUserActual.col,artistUserActual.idFavoritos,cif,
        db.collection(collectionPath: "${Constantes.collectionArtistUser}"),
        document(VariablesCompartidas.usuarioArtistaActual!!.userId.toString()) //Send la clave del documento,
        .setOnSuccessListener { it:Void! }

    //val us : User = user as User
    //Log.i("profile", currentUser.email.toString())
    VariablesCompartidas.usuarioArtistaActual = artistUser
    currentUser?.updateEmail(artistUser.email.toString())

    val navigationView: NavigationView =
        (context as AppCompatActivity).findViewById(R.id.nav_view)
    val header: View = navigationView.getHeaderView(index: 0)
    val imgHead: ImageView = header.findViewById(R.id.image_basic_user_header)
    val nameHead: TextView = header.findViewById(R.id.txt_userName_header)
    val emailHead: TextView = header.findViewById(R.id.txt_userEmail_header)

    imgHead.setImageBitmap(photo)
    nameHead.text = artistUser.userName
    emailHead.text = artistUser.email

    Toast.makeText(requireContext(), "SUSCESFULLY", Toast.LENGTH_SHORT).show()

    }.addOnFailureListener{ it:Exception
        Toast.makeText(requireContext(), "ERROR", Toast.LENGTH_SHORT).show()
    }
} else{
    Toast.makeText(requireContext(),"Complete all fields",Toast.LENGTH_SHORT).show()
}
}

```

En el caso de las publicaciones, aquí podemos ver que primero se sube la imagen a Storage y en caso correcto, realiza una inserción en firebase en la tabla post

```

private fun savePost() {
    if (photo != null) {
        val postId = UUID.randomUUID().toString()
        val stId = UUID.randomUUID().toString()
        val imgStId = "$stId.jpg"
        val imageRef = storageRef.child(pathString: "${stId}.jpg")

        //subimos la img a storage
        val uploadTask = imageRef.putBytes(Utils.getBytes(photo!!)!!)
        uploadTask.addOnSuccessListener { it:UploadTask.TaskSnapshot!
            /saveComentarioFirebase( Utils.getBytes(photo!!) )
            val byteArray: ByteArray? = Utils.getBytes(photo!!)

            val post = Post(
                postId,
                VariablesCompartidas.usuarioArtistaActual!!.userId,
                imgStId,
                listaEtiquetas
            )
            //guardamos el post en firebase
            db.collection(collectionPath: "${Constantes.collectionPost}")
                .document(documentPath: "${postId.toString()}") //Será la clave del documento.
                .set(post).addOnSuccessListener { it:Void!
                    listaEtiquetas.clear()
                    finish()
                }.addOnFailureListener{ it:Exception
                    Toast.makeText(context: this, "ERROR", Toast.LENGTH_SHORT).show()
                }
            }.addOnFailureListener{ it:Exception
                Toast.makeText(context: this, "ERROR", Toast.LENGTH_SHORT).show()
            }
    }
}

```

En el caso del chat, primero se comprueba si existe un chat entre el usuarioArtista y el otro usuario (en este caso el usuario registrado), en caso de no existir un chat lo crea realizando una insercción en firebase

```

private fun getChat() {
    var existe = false
    db.collection( collectionPath: "${Constantes.collectionChat}")
        .whereEqualTo( field: "idUserArtist", userMuro!!.userId)
        .get()
        .addOnSuccessListener { chats ->
            //No existen chats
            for (chat in chats) {

                var idChat: String? = null
                if (chat.get("idChat") != null) {
                    idChat = chat.get("idChat").toString()
                }
                var ch = Chat(
                    chat.get("idChat").toString(),
                    chat.get("idUserArtist").toString(),
                    chat.get("userNameArtist").toString(),
                    chat.get("idUserOther").toString(),
                    chat.get("userNameOther").toString(),
                    chat.get("date").toString()
                )
                if (ch.idUserOther!!.toString() == VariablesCompartidas.idUsuarioActual) {
                    existe = true
                    var myIntent = Intent(context, ChatActivity::class.java)
                    myIntent.putExtra( name: "idChat", idChat)
                    myIntent.putExtra( name: "userName", ch.userNameArtist)
                    myIntent.putExtra( name: "date", ch.date)
                    startActivity(myIntent)
                }
            }
            if (!existe) {
                crearChat()
            }
        }
}

```

Para conseguir una funcion de realismo para que el envio de mensajes sea en tiempo real se añade un listener que estara pendiente de si hay nuevos mensajes

```

fun getDataFromFireStore() {
    try{
        db.collection( collectionPath: "${Constantes.collectionComentario}")
            .whereEqualTo( field: "idChat", idChat)
            .addSnapshotListener { snapshots, e ->
                if (e != null) {
                    Toast.makeText( context: this, "ERROR", Toast.LENGTH_SHORT).show()
                    return@addSnapshotListener
                }
                obtenerDatos(snapshots)
            }
    }catch (e : Exception){
        Toast.makeText( context: this, "ERROR", Toast.LENGTH_SHORT).show()
        throw e
    }
}

```

En el caso del buscador, aquí mostraremos de ejemplo la forma para filtrar por distancia

```
private fun calculationByDistance(location: Location?, lat2: Double?, lon2: Double?): Double {
    val Radius = 6371 // radius of earth in Km
    val lat1 = location!!.latitude
    val lon1 = location.longitude
    val dLat = Math.toRadians(lat2!! - lat1)
    val dLon = Math.toRadians(lon2!! - lon1)
    val a = (Math.sin(dLat / 2) * Math.sin(dLat / 2)
        + (Math.cos(Math.toRadians(lat1))
        * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2))
        * Math.sin(dLon / 2)))
    val c = 2 * Math.asin(Math.sqrt(a))
    val valueResult = Radius * c
    val km = valueResult / 1
    val newFormat = DecimalFormat( pattern: "####")
    val kmInDec = Integer.valueOf(newFormat.format(km))
    val meter = valueResult % 1000
    val meterInDec = Integer.valueOf(newFormat.format(meter))
    Log.i(
        tag: "Radius Value", msg: "" + valueResult + " KM " + kmInDec
        + " Meter " + meterInDec
    )
    return Radius * c
}
```

Una vez realizados todos los filtros, la se nos queda un conjunto de resultados que podemos enviar al modo lista o al modo mapa.

```
private fun addMarkers(){
    if(resultados != null){
        for(us in resultados!!){
            val studio = LatLng(us.latitudUbicacion!!, us.longitudUbicacion!!)
            val name = us.userName
            mMap.addMarker(MarkerOptions().position(studio).title( title: "${name.toString()}"))
            resultadosSt.add(name.toString())
            //resultadosIndex.add(resultados!!.indexOf(us))
        }
    }
}
override fun onMarkerClick(p0: Marker): Boolean {

    AlertDialog.Builder( context: this).setTitle( title: "${p0.title.toString()}")
        .setPositiveButton("CONFIRM") { view, _ ->
            goToMuro(p0)
            //finish()
            view.dismiss()
        }.setNegativeButton("NO") { view, _ ->
            //elimina marcador
            //p0.remove()
            view.dismiss()
        }.create().show()
    return false
}

private fun goToMuro(p0: Marker){
    val index : Int = resultadosSt.indexOf(p0.title.toString())
    val artistUser : ArtistUser = resultados!![index]
    VariablesCompartidas.usuarioArtistaVisitaMuro = artistUser
    |
    val intent = Intent( packageContext: this, ArtistMuroConatinerActivity::class.java)
    startActivity(intent)
}
//funcion para limpiar la lista de resultados
```

En el caso del mapa, al navegar a esta ventana, añadiremos un marcador por cada usuario que cumpla los requisitos de búsqueda, y al hacer click en el marcador podremos navegar al perfil público.

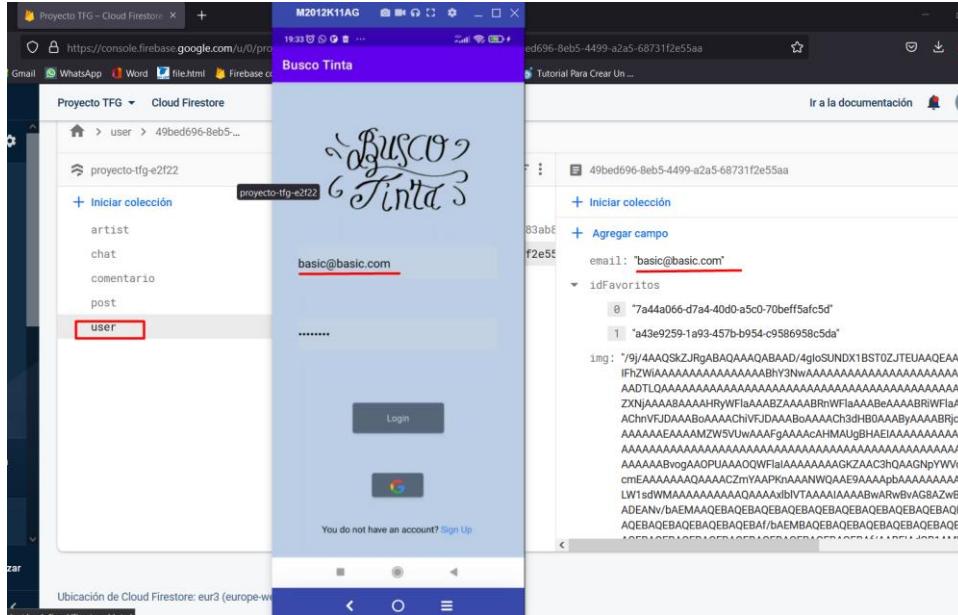
6. PRUEBAS

6. 1. Realización de pruebas

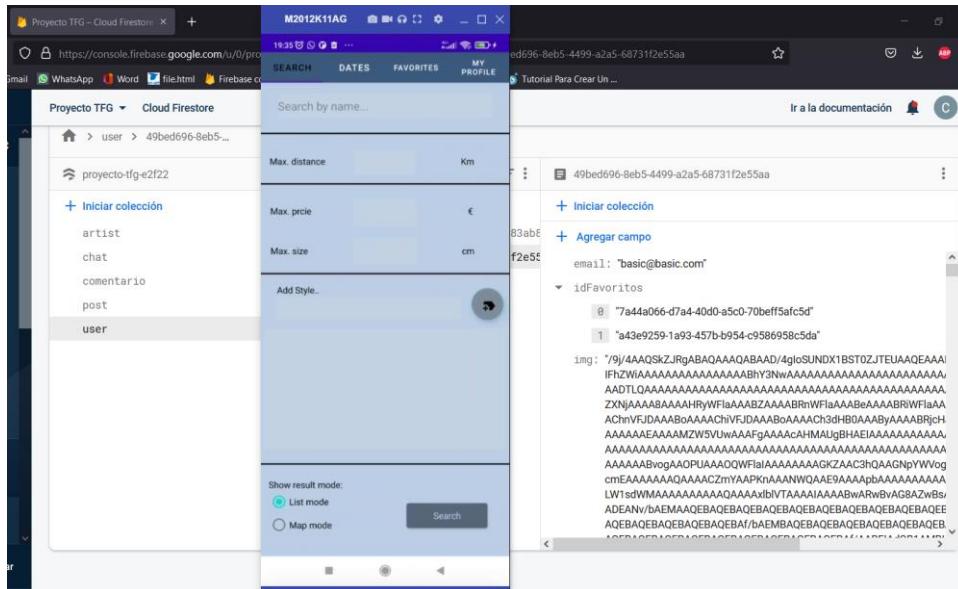
Vamos a realizar una serie de pruebas para comprobar el correcto funcionamiento de la aplicación.

En primer lugar, comenzaremos por el Login.

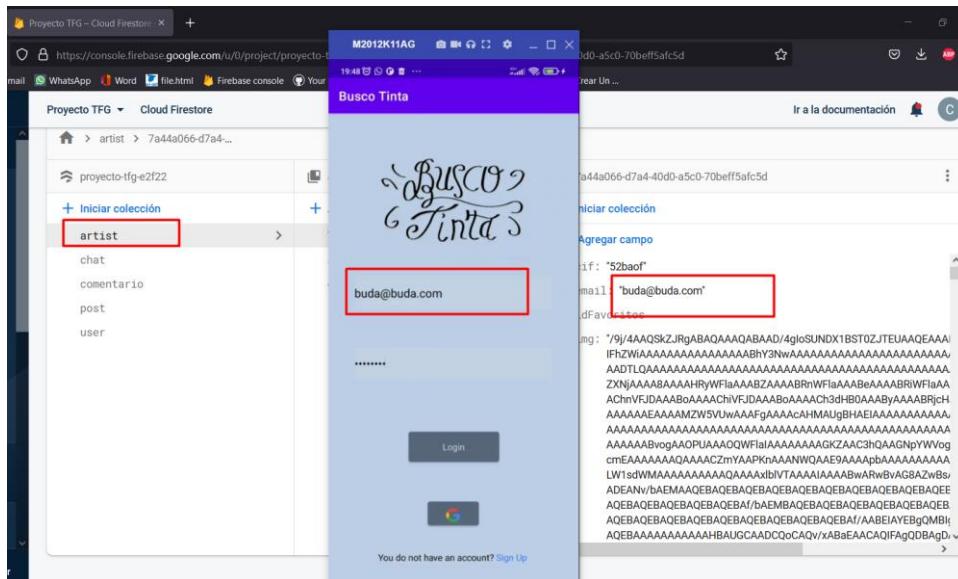
La primera comprobación que haremos será la de entrar con un usuario Básico que exista en la base de datos.



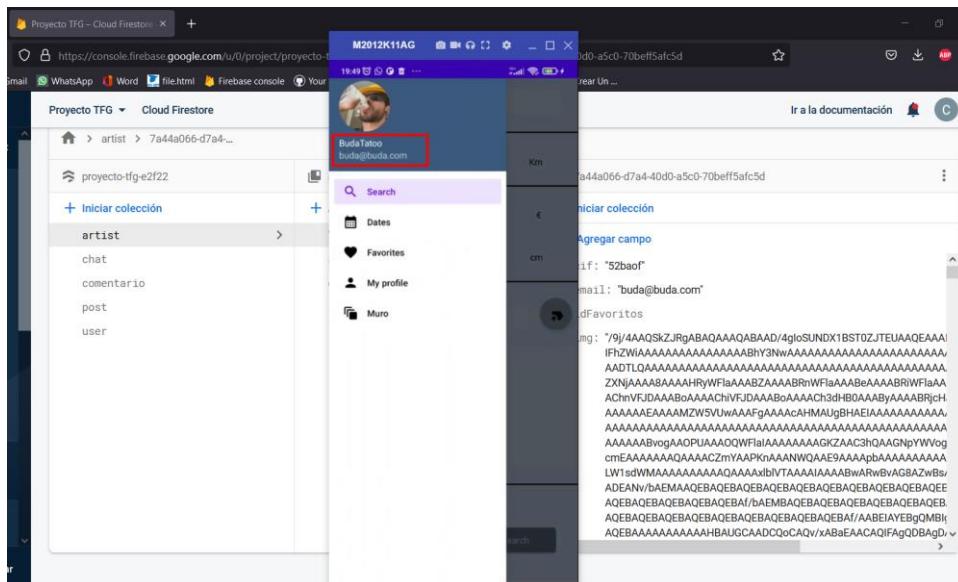
Y podemos observar que accede correctamente a la aplicación. Abriendo un TabActivity con cuatro pestañas para las funciones del usuario básico.



En el caso de un usuario artista que existe en la aplicación



Accede correctamente a la aplicación y se abrirá un NavigationDrawer con las opciones para el usuario artista.



En caso de intentar acceder un usuario que no existe en la base de datos en ninguna de las dos tablas, o que la contraseña sea incorrecta, vemos que la aplicación mostrará un mensaje Toast de "Login Error"



En caso de intentar acceder sin haber escrito nada, mostrará otro Toast pidiendo rellenar los campos.

Para el caso de un Login con rol de administrador, se comprobará en la tabla de users con ese rol

The screenshot illustrates the connection between the app's user interface and the underlying database. The app's login screen on the left shows an empty email field, while the Cloud Firestore database on the right shows a user document with an email value of 'a@a.com'. On the right, a specific field, 'rol', is highlighted with a red box and has a value of 'admin', indicating that the user is an administrator.

Se comprueba que sea rol admin y se accede correctamente al modo Administrador. Que consta de un BottomActivity con las tres listas para las opciones disponibles para el Administrador

Podemos ver también que el recycler view muestra correctamente los usuarios artistas que hay en la base datos.

Tambien podemos ver otros recycler view como los chats que hay entre artistas y usuarios básicos (o entre artistas con otros artistas)

Podemos editar o eliminar los artistas, los post de sus muros, los usuarios básicos, los chats, y los mensajes de los chats.

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar lists collections: 'artist', 'chat', 'comentario', 'post', and 'user'. The 'artist' collection is selected. Inside the 'artist' collection, a document for 'BudaTatoo' is highlighted with a red box. This document's details are shown on the right, including fields like 'attitudUbicacion', 'longitudUbicacion', 'home', 'races', 'rol', 'sizes', 'serId', and 'userName'. A modal dialog box with a red border is overlaid on the document details, containing the text 'Chose option' and two buttons: 'UPDATE' and 'DELETE'.

Y vemos que elimina correctamente a los usuarios

The screenshot shows the 'Users' collection in the Firebase Cloud Firestore interface. The 'BudaTatoo' document from the previous screenshot is no longer present in the list of documents. The remaining documents are 'artista' and 'artis1', both of which have their profile pictures set to a black placeholder icon.

Si escogemos update, podemos modificar los usuarios

Artis1 Document Data:

- userId: d57507b3-cb03-4b10-a5f4-bd9005275558
- userName: artis1
- email: artist1@art.com
- phone: 555444333
- cIv: 57nL
- latitudUbicacion: 39.03094998821529
- longitudUbicacion: -4.180564060807229
- phone: 555444333
- prices
- rol: "artist"
- sizes
- userId: "d57507b3-cb03-4b10-a5f4-bd9005275558"
- userName: "artis1"

Artis2 Document Data:

- userId: d57507b3-cb03-4b10-a5f4-bd9005275558
- userName: artis2
- email: artist2@art.com
- phone: 555444333
- cIv: 57nL
- latitudUbicacion: 39.03094998821529
- longitudUbicacion: -4.180564060807229
- phone: 555444333
- prices
- rol: "artist"
- sizes
- userId: "d57507b3-cb03-4b10-a5f4-bd9005275558"
- userName: "artis2"

BIBLIOGRAFIA

https://cincodias.elpais.com/cincodias/2015/02/01/lifestyle/1422792260_243066.html

<https://www.androidjefe.com/costos-publicar-vender-apps-google-play-store/>

https://play.google.com/intl/ALL_es/about/developer-distribution-agreement.html

<https://firebase.google.com/docs/firestore/billing-example?hl=es>

<https://firebase.google.com/docs/firestore/quotas?hl=es-4199>

https://es.wikipedia.org/wiki/Interfaz_de_usuario

https://es.wikipedia.org/wiki/Arquitectura_de_software

REFERENCIAS ->

Trabajo Fin de Grado INSTADROID

https://aulasciclos2122.castillalamancha.es/pluginfile.php/255143/mod_resource/content/1/proyectoinstadroidcarlosgonzalez.pdf

Trabajo Fin de Grado NaturalezaViva

https://aulasciclos2122.castillalamancha.es/pluginfile.php/255142/mod_resource/content/1/Naturalezaviva.pdf

AGRADECIMIENTOS ->

Noelia Barrionuevo (Diseño gráfico del logo y splash screen)