

# B&A

Desarrollo de aplicaciones multiplataforma

Proyecto Fin de Grado

CIFP Virgen de Gracia

**Beatriz Nielfa Naranjo**

# ÍNDICE

Presentación del proyecto .....	3
1. Explicación resumida .....	3
2. Estudio de merado .....	4
Conocer a la competencia .....	4
Descubrir a nuestro público o target .....	6
Análisis DAFO .....	9
3. Valor del producto.....	10
Planificación de tareas y estimación de costes .....	10
1. Planificación y organización de tareas.....	10
2. Estimación de costes y recursos .....	17
3. Herramientas usadas.....	19
4. Gestión de riesgos .....	20
Riesgos Laborales .....	20
Riesgos específicos de proyectos de software .....	21
Análisis de la solución .....	22
1. Análisis y especificación de requisitos .....	22
Requisitos funcionales.....	22
Requisitos no funcionales.....	23
Requisitos de información.....	23
2. Análisis de escenarios (casos de uso y mapa de interacción). .....	25
Componentes .....	26
Diagrama de casos de uso .....	28
Diseño de la solución .....	29
1. Diseño de la interfaz de usuario y prototipos .....	29
Prototipado de B&A (Usuarios) .....	32
Prototipado de B&A (Administrador) .....	41
2. Diagrama de clases .....	48
Diagrama de clases de B&A .....	51
3. Diseño de la persistencia de la información .....	52
Diagrama de persistencia de datos de B&A.....	54
4. Arquitectura del sistema .....	55
Diagrama de arquitectura de B&A .....	56
Implementación de la solución .....	57
1. Análisis tecnológico .....	57

Tecnología del cliente.....	57
Tecnología del servidor .....	60
2. Elementos a implementar .....	63
3. Aspectos esenciales de la implementación .....	67
Testeo y pruebas de la solución .....	98
1. Plan de pruebas (unitarias, integración, sistema y usuarios).....	98
Pruebas B&A .....	100
2. Solución a problemas encontrados .....	126
Lanzamiento y puesta en marcha.....	127
1. Aspectos relevantes del despliegue y puesta en marcha del sistema .....	127
Despliegue de B&A.....	128
2. Manual de uso .....	132
Valoración y conclusiones .....	133
Bibliografía y recursos utilizados .....	134

# Presentación del proyecto

## 1. Explicación resumida

B&A es una aplicación desarrollada para una tienda de ropa, en nuestra aplicación buscamos sobre todo la simplicidad, que al usuario le resulte lo más sencillo posible toda la gestión que tenga que realizar en nuestra tienda, ya sean compras, devoluciones o aclarar las dudas que les surjan. Esta aplicación será desarrollada para dispositivos Android.



Nuestra aplicación tendrá varias funcionalidades para facilitar al máximo la vida al usuario, para empezar, la primera impresión que se llevará el usuario será un Splash Screen con el logo de nuestra empresa, pasado unos segundos nos llevará a la pantalla de login, donde el usuario tendrá la opción de iniciar sesión si tiene cuenta, o también tendrá la opción de registrarse si es la primera vez que usa la aplicación, también tendrá la opción de iniciar sesión a través de una cuenta de google o twitter.

En caso de tener que registrarse, el usuario tendrá que llenar un formulario de registro, el cual incluirá foto, correo, nombre y contraseña. La foto se podrá tomar en vivo (cámara) o se podrá subir desde la galería de imágenes.

Una vez hayamos iniciado sesión tendremos acceso al menú principal, donde se encontrará el catálogo de ropa, desde el que podremos ver la ropa de la aplicación y mandarla a la cesta, el catálogo que incluirá foto, stock, talla, tipo y precio. En esta pantalla habrá un filtro para ordenar las prendas por precio, también habrá una barra de búsqueda.

Habitualmente se puede dar el caso de que el usuario esté interesado en una prenda, para ello tendrá la opción de darle a me gusta para añadirla a sus prendas favoritas y tenerla a mano para una posible futura compra.

En la cesta tendremos todas las prendas previamente seleccionadas, habrá la opción de eliminar alguna si así lo deseamos y también de comprar la cesta.

Cabe destacar que la aplicación tendrá un historial de compras, donde estarán todas las compras que hayamos realizado anteriormente, no obstante, si el usuario desea realizar una devolución de alguna prenda, en este historial existirá dicha opción, donde podremos exportar la prenda a través de un código QR con el cual podremos devolverla.

Para facilitar la compra al usuario hemos implementado un sistema el cual permite al usuario realizar un pedido a la tienda, pedido que incluiría, el id del comprador, la prenda seleccionada y la dirección a donde hay que mandarla, esta dirección se seleccionará en un mapa.

Con el propósito de resolver cualquier duda que el usuario pueda tener, hemos implementado un chat dentro de la aplicación, que servirá como herramienta para que el usuario se pueda comunicar con un asistente para cualquier duda que pueda tener.

Por último, el usuario tendrá la posibilidad de editar su perfil, modificar la contraseña, foto y nombre.

## 2. Estudio de merado

Un estudio de mercado es un proceso sistemático de recolección y análisis de datos e información acerca de los clientes, los competidores y el mercado.

Vamos a hacer un estudio de mercado para examinar si B&A tendrá posibilidades de éxito. Desde un punto de vista conservador, vamos a tomar decisiones minimizando los riesgos.

Gracias a este estudio de mercado vamos a conocer el perfil y el comportamiento de nuestros clientes, la situación del mercado o la industria a la que nos dedicamos, descubriremos cómo trabaja nuestra competencia y nuestros posibles proveedores.

Para realizar un buen estudio de mercado vamos a llevar a cabo estos conceptos:

- Conocer a la competencia.
- Descubrir a nuestro público o target.
- Análisis DAFO

### Conocer a la competencia

En cuanto a la competencia a nivel nacional, este sector está muy cargado, hay muchas apps de tiendas de ropa a disposición del usuario, tenemos grandes empresas en este sector, como son Inditex, H&M, Shein y Mango, con unas apps muy potentes, muy asentadas en el mercado y muy bien valoradas en la app store de Android.



P&B

PULL&BEAR  
Inditex



ZARA

Zara  
Inditex



Stradivarius - Moda  
Inditex



Bershka

Bershka - Moda y te  
Inditex



MANGO

H&M  
H&M



MANGO - Lo último  
MANGO



SHEIN-Compras de  
ZOETOP BUSINESS CO.



Por otro lado, tenemos empresas que indirectamente nos pueden hacer la competencia, ya que no son exclusivamente apps de ropa, pero su magnitud es tan grande que nos pueden resultar competentes, en este apartado se encuentran Amazon y Aliexpress.



Amazon compras  
Amazon Mobile LLC



AliExpress - Compr  
Alibaba Mobile



A su vez en el sector se encuentran grandes empresas que podrían ser competencia directa, pero no son consideradas como tal, ya que al no ofrecer el mismo tipo de producto (nosotros ofrecemos prendas no deportivas). En este apartado se podrían encontrar Nike y Sprinter.



Sprinter  
Sprinter Sports

★★★☆☆



Nike: calzado y ropa  
Nike, Inc.

★★★★★

## Descubrir a nuestro público o target

El target es sumamente importante porque indicará el tipo de personas al que va dirigido nuestro producto.

Es decir, el target es el público que será el futuro consumidor de nuestra app. Gracias a un exhaustivo estudio vamos a poder conocer cuál va a ser nuestro target y por lo tanto conocer al tipo de personas que queremos “seducir” y evitar así dirigir nuestras acciones a todo el mundo, una misión imposible.

Hay distintos factores que podemos tener en cuenta para determinar nuestro target, pero para nosotros los más importantes y los que vamos a tener en cuenta son la edad y la ubicación.

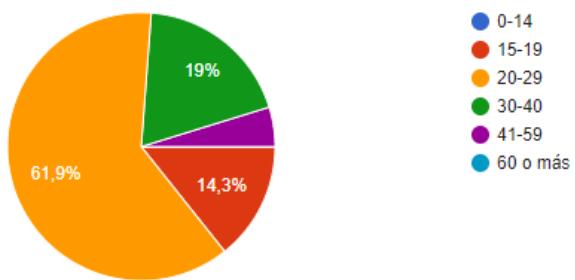
La edad va a ser un factor determinante ya que nuestros productos van a estar destinados a un público joven y adulto, la ubicación también va a ser importante ya que, al estar empezando, ser una empresa pequeña y ser online no vamos a realizar pedidos fuera de España.

Hemos realizado una encuesta para ayudarnos a conocer más a nuestro target, estos han sido los resultados:



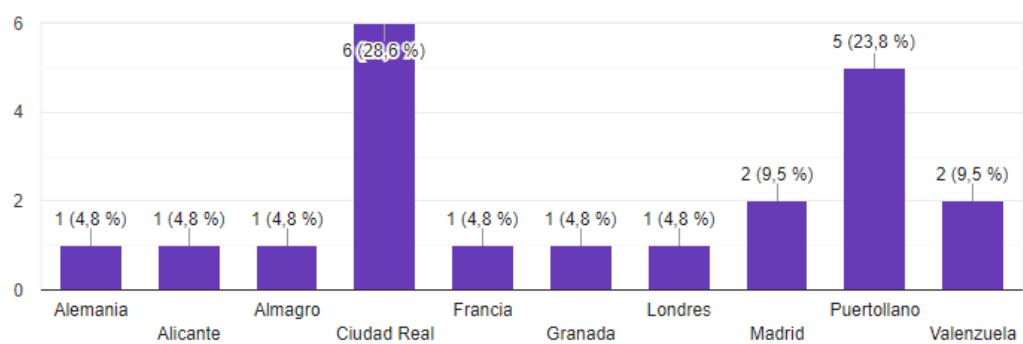
¿Cuántos años tienes?

21 respuestas



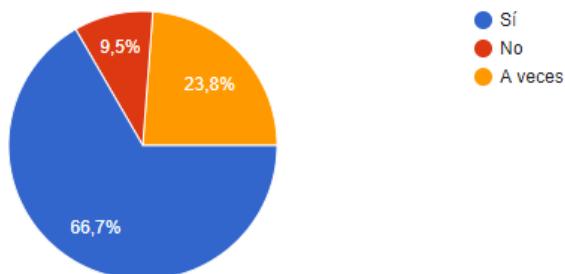
¿Dónde vives?

21 respuestas



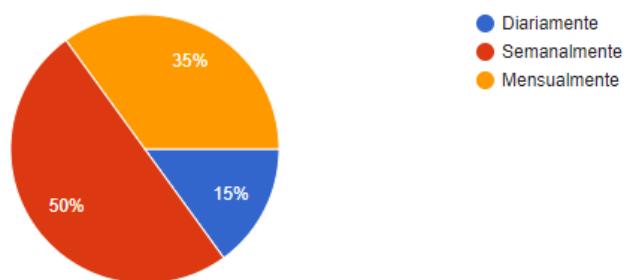
¿Suele hacer compras por internet?

21 respuestas



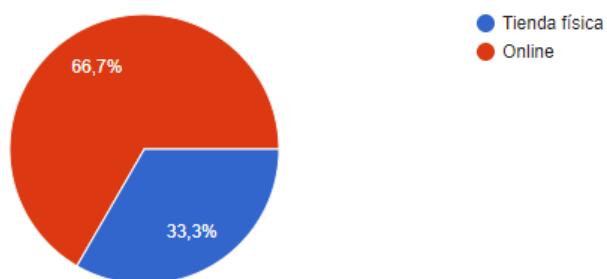
¿Cuán a menudo?

20 respuestas



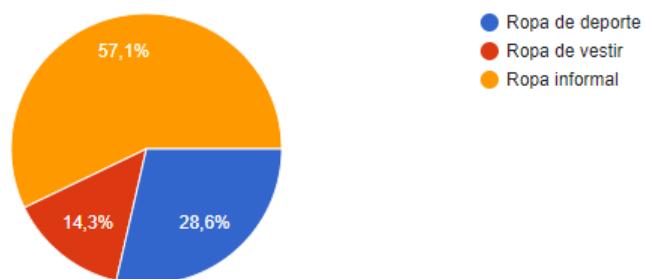
¿Prefiere comprar ropa online o en tienda física?

21 respuestas



¿Qué tipo de ropa suele comprar?

21 respuestas



## Análisis DAFO

El análisis DAFO es una técnica indispensable para analizar la situación actual de la app, y poder tomar las decisiones estratégicas adecuadas.

Este es un análisis del entorno externo y las características internas del negocio, esta herramienta nos permite obtener una representación gráfica de:

- Debilidades: Constituyen los aspectos limitadores de la capacidad de desarrollo de nuestro negocio.
- Amenazas: Son todos aquellos factores externos que pueden llegar a impedir la ejecución de nuestra estrategia empresarial.
- Oportunidades: son cualesquiera factores ajenos a nuestro negocio que favorecen su desarrollo o brindan la posibilidad de implantar mejoras.
- Fortalezas: Reúnen el conjunto de recursos internos, posiciones de poder y cualquier tipo de ventaja competitiva de nuestro negocio.



### 3. Valor del producto

Resumiendo lo planteado anteriormente no cabe duda de que ya hay gigantes asentadas en este sector, pero queremos intentar ser nosotros una app de referencia e innovadora dejando a un lado los diseños típicos y anticuados que predominan en este sector y renovarlos y darles el toque de frescura que nos caracteriza.

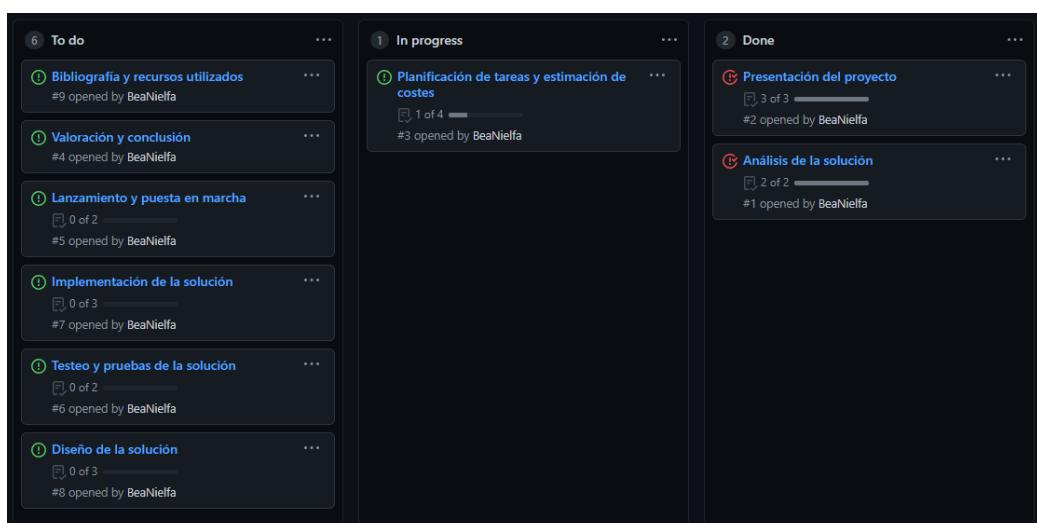
## Planificación de tareas y estimación de costes

### 1. Planificación y organización de tareas

La planificación es un tema importante para nosotros, y por ello nos lo hemos tomado muy en serio, ya que la correcta planificación nos ayuda a establecer la prioridad de cada una de las actividades y a tener un mejor control del tiempo para ejecutar un proyecto con la calidad deseada y con éxito.

Para esta labor hemos utilizado la herramienta de tableros de GitHub donde hemos creado un Kanban básico que hemos ido rellenando con las distintas tareas y subtareas que hemos identificado, incluyendo información resumida de cada punto y la fecha estimada de finalización. Además, también hemos usado la herramienta TimeLines de GitKraken donde hemos creado de manera gráfica una línea temporal donde hemos puesto las tareas a realizar.

Hemos elegido esta herramienta en Kanban porque nos sentimos muy cómodos con esta metodología de trabajo ya que poder visualizar todo el trabajo pendiente, en marcha y finalizado en un golpe de vista nos hace medir mejor el tiempo y ser mucho más óptimos.



Vamos a explicar punto por punto la planificación:

### 1. Presentación del proyecto.

BeaNielfa commented 23 hours ago • edited

Owner

- Explicación resumida
- Estudio de mercado
- Valor del producto

*Fecha estimada para terminar: 12-04-21*  
*Fecha finalizada: 10-04-21*

En esta tarea vamos a realizar una pequeña introducción sobre B&A, el estudio de mercado y el valor del producto.

Teníamos previsto terminar el 12-04-21 y lo hemos hecho el 10-04-21.

### 2. Planificación de tareas y estimación de costes.

BeaNielfa commented 23 hours ago • edited

Owner

- Planificación y organización de tareas
- Estimación de costes y recursos
- Herramientas usadas
- Gestión de riesgos

*Fecha estimada para terminar: 23-04-21*

En esta tarea vamos a realizar la planificación y organización de las tareas, la estimación de los costes y recursos, vamos a exponer las herramientas usadas y vamos a hablar de la gestión de riesgos.

Tenemos previsto terminar el 23-04-21.

### 3. Análisis de la solución.

BeaNielfa commented 23 hours ago • edited

Owner

- Análisis de la solución
- Análisis de escenarios

*Fecha estimada para terminar: 16-04-21*  
*Fecha finalizada: 15-04-21*

En esta tarea vamos a realizar un análisis de la solución y un análisis de escenarios

Teníamos previsto terminar el 16-04-21 y lo hemos hecho el 15-04-21.

#### **4. Diseño de la solución.**

BeaNielfa commented 23 hours ago • edited

Owner

- Diseño de la interfaz de usuario y prototipos
- Diseño de la persistencia de la información
- Diseño de la arquitectura del sistema

*Fecha estimada para terminar: 03-05-21*

En esta tarea vamos a realizar el diseño de la interfaz de usuario y los prototipos, el diseño de la persistencia de la información y el diseño de la arquitectura del sistema.

Tenemos previsto terminar el 03-05-21.

#### **5. Implementación de la solución.**

BeaNielfa commented 23 hours ago • edited

Owner

- Justificación tecnológica
- Aspectos esenciales en la implementación
- Desarrollo de la funcionalidad indicada por el tutor/a

*Fecha estimada para terminar: 07-05-21*

En esta tarea vamos a exponer la justificación tecnológica, los aspectos esenciales en la implementación y el desarrollo de la funcionalidad indicada por el tutor.

Tenemos previsto terminar el 07-05-21.

#### **6. Testeo y pruebas de la solución.**

BeaNielfa commented 23 hours ago • edited

Owner

- Plan de pruebas
- Solución a problemas encontrados

*Fecha estimada para terminar: 03-06-21*

En esta tarea vamos a realizar el plan de pruebas y vamos a exponer todos los problemas encontrados.

Tenemos previsto terminar el 03-06-21.

## 7. Lanzamiento y puesta en marcha.

BeaNielfa commented 23 hours ago • edited

Owner

Aspects relevantes del despliegue y puesta en marcha del sistema  
Manual de uso

Fecha estimada para terminar: 06-06-21

En esta tarea vamos a ver los aspectos más relevantes del despliegue y la puesta en marcha del sistema, también vamos a desarrollar el manual de uso.

Tenemos previsto terminar el 06-06-21.

## 8. Valoración y conclusiones.

BeaNielfa commented 23 hours ago • edited

Owner

Fecha estimada para terminar: 07-06-21

En esta tarea vamos a exponer nuestra valoración y conclusión.

Tenemos previsto terminar el 07-06-21.

## 9. Bibliografía y recursos utilizados.

BeaNielfa commented 1 hour ago • edited

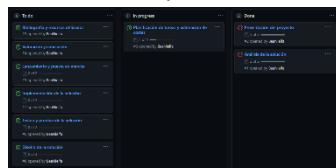
Owner

Fecha estimada para terminar: 10-06-21

En esta tarea vamos a realizar la bibliografía y vamos a mostrar los recursos utilizados.

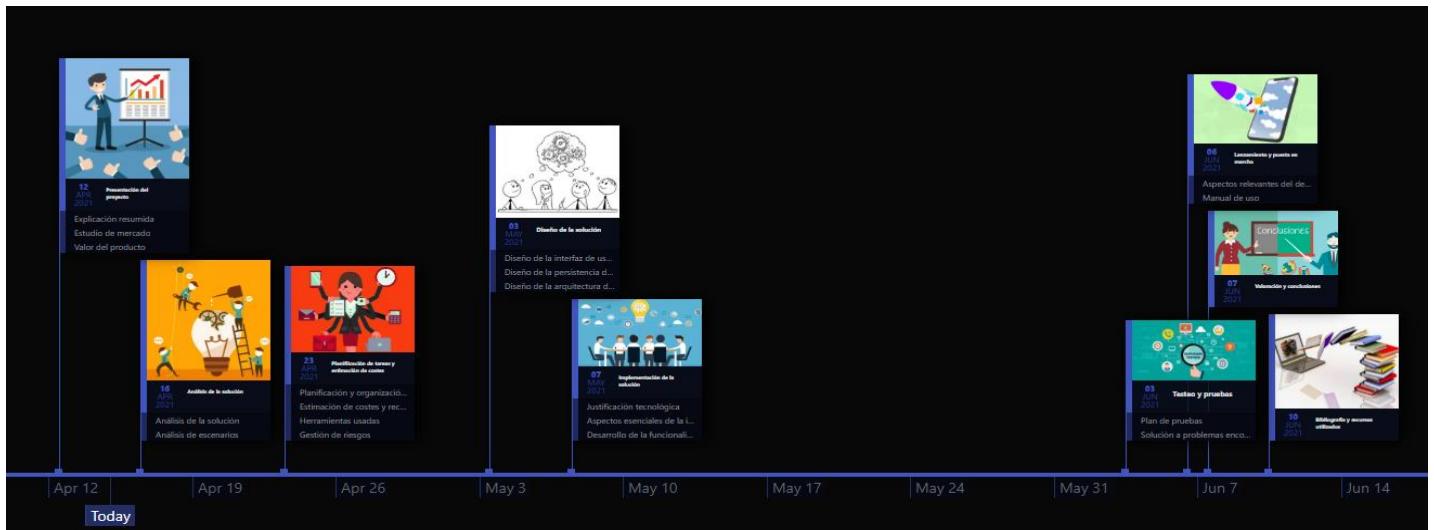
Tenemos previsto terminar el 10-06-21.

### Enlace a la planificación



Línea temporal creada en GitKraken TimeLines:

En esta línea temporal podemos ver más detalladamente las tareas la planificación previamente pensada.



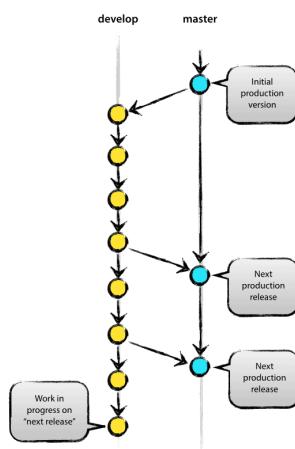
Como flujo de trabajo vamos a usar GitFlow, que es un modelo basado en Git que brinda un mayor control y organización en el proceso de integración continua<sup>1</sup>.

Vamos a tener dos tipos de ramas, las ramas principales y las ramas de apoyo.

- **Las ramas principales:**

Vamos a tener dos ramas principales, la master y la develop.

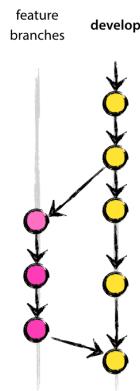
La rama master es la rama que contiene cada una de las versiones estables del proyecto y la rama develop es la rama donde el código va a reflejar un estado con los últimos cambios del desarrollo, cuando el código en esta rama alcanza un punto estable y está listo para ser lanzado, todos los cambios deben fusionarse con la rama master.



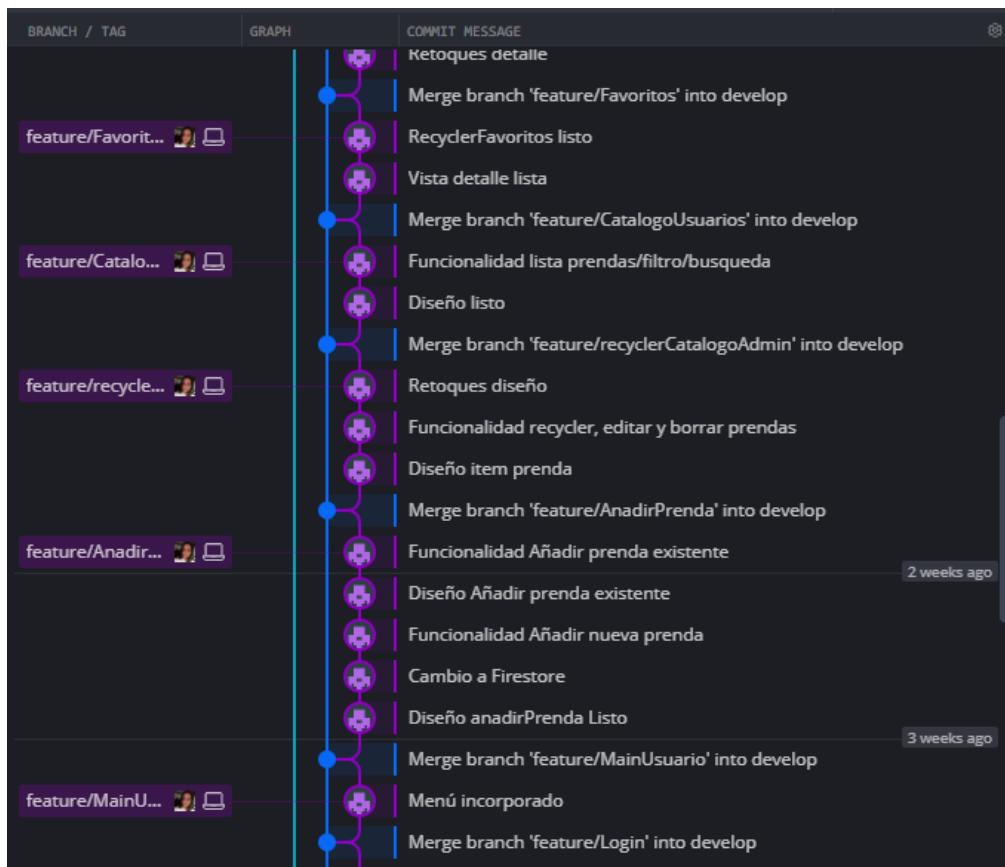
<sup>1</sup> La integración continua es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica.

- **Las ramas de apoyo:**

Vamos a tener ramas auxiliares llamadas features, estas llamas las vamos a usar para desarrollar nuevas funciones para la próxima versión. La esencia de estas ramas es que existen mientras tengamos una característica en desarrollo, pero eventualmente la fusionaremos con la rama develop para agregar definitivamente la característica a la próxima versión, o la descartaremos en caso de que la característica en desarrollo haya sido decepcionante.



Esta es una captura de como he usado GitFlow en el proyecto.



Para concluir la planificación, le hemos querido dar importancia a la documentación, ya que una buena documentación es clave para el éxito de cualquier proyecto. Hacer que la documentación sea accesible permite a las personas aprender sobre un proyecto, hemos elegido dos formas de documentar nuestro proyecto:

- **Mediante archivos README:**

Ya que son una forma rápida y sencilla para que otros usuarios aprendan sobre nuestro proyecto.

Enlace al Readme:

Tabla de contenidos:

- Descripción y contexto
- Guía de usuario
- Wiki del proyecto
- Código fuente
- Herramientas y tecnologías utilizadas
- Autor/es

Descripción y contexto

Tú aplicación de moda favorita

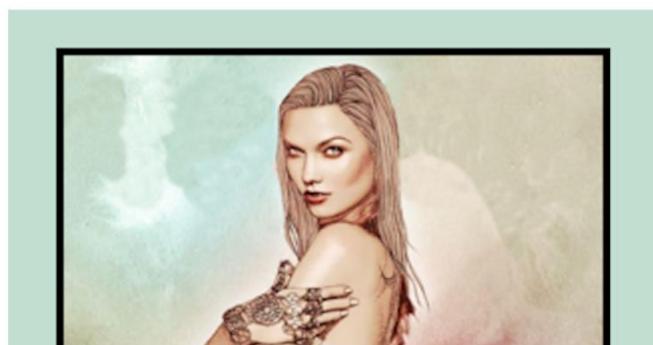
- Compra tu ropa favorita 🛍
- Te la envían directa a casa 🚚
- Devuelve las prendas que no te hayan gustado 💡
- Contacta con el asistente 😊
- Y sobre todo estate a la moda 💃

- **Mediante una Wiki en GitHub:**

Una buena Wiki en GitHub nos va a facilitar el poder presentar información detallada sobre nuestro proyecto de manera útil.

Enlace a la Wiki:

Home  
BeaNieffa edited this page 6 days ago - 5 revisions



## 2. Estimación de costes y recursos

Estimar los costos consiste en desarrollar una estimación aproximada de los recursos monetarios necesarios para completar las actividades del proyecto, suele ser un proceso iterativo, es decir, la exactitud de la estimación del costo aumenta según avanza el proyecto.

Esta tarea es realmente difícil, crear un presupuesto que de verdad funcione para nosotros y para los clientes es prácticamente un arte, nos pueden ser de gran ayuda los [cronogramas del proyecto](#).

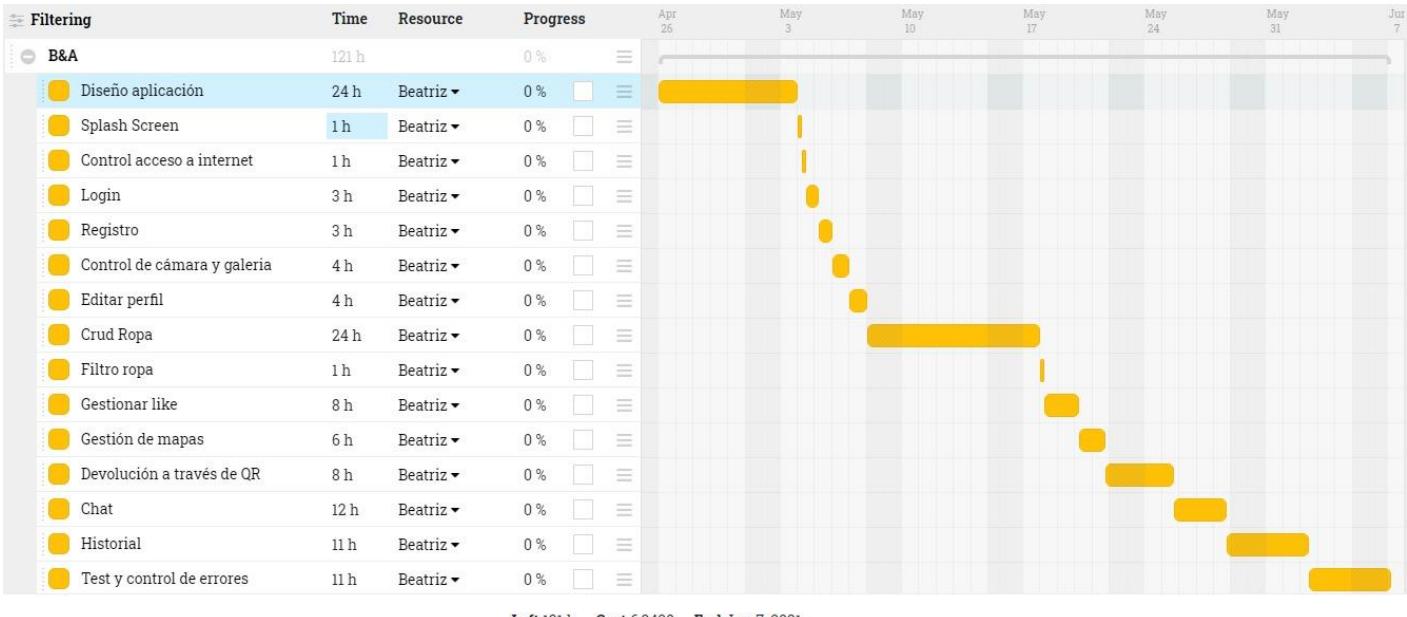


Este proceso puede ser estresante y aterrador, siempre hay conjeturas involucradas (de ahí que sea una estimación de costos y no un pronóstico exacto) y si te equivocas con el presupuesto, como gerente del proyecto puedes verte envuelto en muchos problemas.

Nosotros vamos a realizar la estimación del coste estimando la duración de las tareas a realizar, teniendo en cuenta que vamos a cobrar 20€ por hora trabajada.

La duración de una tarea va a variar en función a los riesgos asociados (retrasos o adelantos que puedan ocurrir), y la variabilidad intrínseca a la tarea (aspectos como la motivación, el cansancio... pueden afectar al tiempo necesario para hacer determinadas tareas)

Nosotros vamos a realizar una estimación por analogía, que consiste en aproximar la duración de una tarea en base a los datos históricos disponibles referentes a la duración real de la misma tarea en proyectos anteriores, personalmente tengo poca experiencia en este tipo de proyectos, pero usaré la que me ha otorgado "Wallapop" (proyecto en Android que realicé hace un año).



Left 121 h Cost € 2420 End Jun 7, 2021

Este es el gráfico que he diseñado, he puesto las distintas tareas y la estimación temporal que he creído necesaria.

Los costes totales de desarrollo de la aplicación los vamos a calcular **estimando los costos de Firebase ya que es una herramienta difícil de cuantificar, te cobran en función de datos almacenados en la nube, autorizaciones<sup>2</sup>, ...** Y teniendo en cuenta que el resto de herramientas que vamos a usar son totalmente gratuitas, el único coste que va a tener la aplicación será el coste de producción.

Se cobrarán unos extras de gastos de mantenimiento que hacen referencia a luz y gastos derivados del trabajo.

Presupuesto			
Tarea	Cantidad	Precio	Importe
Desarrollo de la aplicación	121h	20 €	2420.00 €
Licencia de Play Store	1	25 €	25 €
Gasto de Mantenimiento	121h	2 €	242 €
	-	250 €	250 €
			<b>Subtotal</b>
			2937.00 €
			<b>IVA 21%</b>
			616.77 €
			<b>TOTAL</b>
			<b>3553.77 €</b>

<sup>2</sup> <https://firebase.google.com/pricing?hl=es>

### 3. Herramientas usadas

En este apartado voy a explicar un poco lo que van a ser mis herramientas de trabajo diario. Estas herramientas van a ser muy importantes ya que los programadores pasamos muchas horas con ellas por lo que nos gusta que la herramienta esté a nuestro gusto, que se ajuste a nuestro flujo de trabajo y que no nos obligue a cambiarlo.



Android Studio es el entorno de desarrollo integrado oficial para el desarrollo de apps para Android y está basado en IntelliJ IDEA.



GitKraken es una potente interfaz gráfica multiplataforma para git desarrollada con Electron.



Github es un repositorio online gratuito que permite gestionar proyecto y controlar versiones de código.



Firebase se trata de una plataforma móvil creada por Google, cuya principal función en desarrollar y facilitar la creación de apps de elevada calidad de una forma rápida.

## 4. Gestión de riesgos

Cualquier proyecto, tenga la característica que tenga, presenta problemas. Cuando hablamos de proyectos de implantación de software es casi imposible no pensar en ellos y en cómo ejecutar una correcta gestión de riesgos.

### Riesgos Laborales

- **Fatiga visual o muscular:** Al pasar demasiadas horas mirando una pantalla y sentados en la misma posición nos puede conllevar a posturas inadecuadas y a molestias oculares.

*Solución: Tener una silla adecuada, un escritorio a la altura correcta y un teclado separado del monitor será fundamental para prevenir la fatiga muscular, además es recomendable levantarse y estirarse cada cierto tiempo.*

*Utilizar una colocación ergonómica de la pantalla, es decir, colocarla a un mínimo de 40 centímetros e inclinar ligeramente la parte inferior de modo que el enfoque sea perpendicular a nuestro ángulo de visión nos ayudará a prevenir la fatiga visual, además es recomendable tener una luz adecuada.*

- **Golpes o caídas:** Tener un espacio de trabajo desorganizado y cargado puede dar lugar a una caída desafortunada.

*Solución: Tener un espacio de trabajo limpio y ordenado, es decir, dejándolo libre de obstáculos. El material de trabajo se deberá almacenar en estanterías y armarios.*

- **Contacto eléctrico:** El mal uso de los equipos eléctricos puede causar daños físicos en nosotros.

*Solución: Respetar las normas de seguridad básicas en el uso de los equipos eléctricos y revisar el estado de cada equipo antes de su uso.*

- **Carga mental:** La gran presión ejercida en nosotros por unos plazos de entrega poco generosos puede causarnos estrés o desmotivación, por lo que puede afectarnos a la salud.

*Solución: Es importante realizar tareas variadas, realizar paradas periódicas para prevenir la fatiga, seguir hábitos de vida saludable y realizar ejercicio de forma habitual.*

## Riesgos específicos de proyectos de software

- **Falta de organización:** La falta de experiencia organizando proyectos de envergadura puede dar lugar a una falla en la organización de este.

*Solución: Apoyarnos de todas las herramientas posibles para evitar que esto ocurra.*

- **Pérdida de información:** Puede darse el caso de que se pierda código o algún archivo importante de nuestro proyecto.

*Solución: El uso de GitHub para ir subiendo todo el código poco a poco y el apoyo de algunas plataformas como google drive nos va a facilitar mucho que no ocurran este tipo de fallos.*

- **Falta de conocimiento de la metodología:** Nuevamente la falta de experiencia puede llevarnos a cometer este tipo de errores.

*Solución: Informarnos bien antes del proyecto sobre las herramientas a implementar.*

- **Retraso en la planificación:** El retraso en la planificación es un fallo que puede repercutir en el resto del proyecto, arrastrando las demás entregas y no cumpliendo ninguna con el plazo previsto.

*Solución: Tener muy claro y ser muy consciente de la estimación previamente hecha.*

- **Interfaz mal planteada:** El hecho de que pueden surgir problemas conforme estemos desarrollando nuestro código puede repercutir en que la interfaz que habíamos diseñado se quede obsoleta y tengamos que planificar una completamente nueva.

*Solución: Realizar un prototipado y un diseño antes de implementar código.*

# Análisis de la solución

## 1. Análisis y especificación de requisitos

### Requisitos funcionales

Un requisito funcional define una función del sistema de software o sus componentes. Una función es descrita como un conjunto de entradas, comportamientos y salidas. Tales requisitos pueden ser: detalles técnicos, cálculos, manipulación de datos, ...

En el caso de B&A los requisitos funcionales son:

- RF1.- El sistema debe mostrar una pantalla de bienvenida al usuario, pantalla que mostrará el logo de nuestra aplicación y pasados unos segundos nos llevará a la pantalla de inicio de sesión.
- RF2.- El sistema mostrará una pantalla de inicio de sesión donde podremos entrar a la aplicación.
- RF3.- El sistema contará con una opción de inicio de sesión a través de google y twitter.
- RF4.- El sistema debe controlar cuando se usa la aplicación sin acceso a internet.
- RF5.- El sistema tendrá una pantalla de registro donde podremos introducir nuestros datos para registrarnos en la aplicación.
- RF6.- El usuario podrá usar la cámara de su dispositivo móvil o de la galería de imágenes para subir una foto al registro y al perfil.
- RF7.- El sistema contará con un menú lateral desplegable para organizar las diferentes opciones de la aplicación.
- RF8.- El sistema tendrá un filtro para elegir el tipo de prenda previo a la lista de prendas.
- RF9.- El sistema tendrá una lista con todas las prendas disponibles en el almacenamiento.
- RF10.- El sistema tendrá un detalle de la prenda, visible a través de un clic en la lista.
- RF11.- El sistema contará con un filtro para organizar las prendas por precio.
- RF12.- El usuario podrá darle me gusta a una prenda para añadirla a favoritos.
- RF13.- El sistema dispondrá de una cesta, a la cual se le podrán ir añadiendo prendas para su futura compra.

- FR14.- El usuario podrá eliminar prendas de la cesta.
- FR15.- El sistema tendrá un historial de compras donde se almacenarán las prendas que haya comprado el usuario.
- FR16.- El usuario podrá devolver una prenda de ropa a través de un código QR accesible a través del historial.
- FR17.- El usuario tendrá la opción de realizar un pedido a la tienda a través de la cesta de la aplicación.
- FR18.- El sistema contará con un mapa donde el usuario podrá indicar la dirección de su vivienda.
- FR19.- El sistema tendrá un chat propio por el cual el usuario se podrá comunicar con un asistente (y viceversa).
- FR20.- El usuario podrá modificar los datos de su perfil.
- FR21.- El usuario administrador podrá añadir/editar/borrar prendas del sistema.
- FR22.- El usuario administrador podrá devolver las prendas gracias al escaneo del código QR proporcionado por el usuario.
- FR23.- El usuario administrador podrá gestionar los pedidos de los clientes.

### Requisitos no funcionales

Los requisitos no funcionales son características de funcionamiento, especifican criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos.

En el caso de B&A los requisitos no funcionales son:

- RNF1.- Todos los datos se almacenarán en la base de datos de Firebase.
- RNF2.- La autorización del usuario se realizará mediante Firebase.
- RNF3.- La aplicación se desarrollará en Kotlin y estará disponible para dispositivos Android.

### Requisitos de información

En los requisitos de información vamos a representar entidades e información, los cuales vamos a usar en B&A.

- RI1.- Usuario: Representa a la persona que va a consumir la aplicación, cabe destacar los campos:

- idUsuario

- Correo
- Nombre
- Foto
- Pass
- Tipo
- Chat

RI2.- Prenda: Representa el producto principal de la aplicación, y en lo que el usuario está interesado, cabe destacar los campos:

- idPrenda
- nombre
- Foto
- Stock
- Referencia
- idTipo
- Precio

RI3.- Tipo: Representa el tipo de la prenda, cabe destacar los campos:

- idTipo
- descripcion

RI4.- Favorito: Representa la/s prenda/s preferidas del usuario, cabe destacar los campos:

- idFavorito
- idUsuario
- idPrenda

RI5.- Pedido: Representa la compra que se ha realizado en la aplicación.

- idPedido
- idPrenda
- idUsuario
- FechaCompra
- Latitud
- Longitud
- Estado
- talla

RI6.- Mensaje: Representa la comunicación que tienen los usuarios en sus conversaciones privadas.

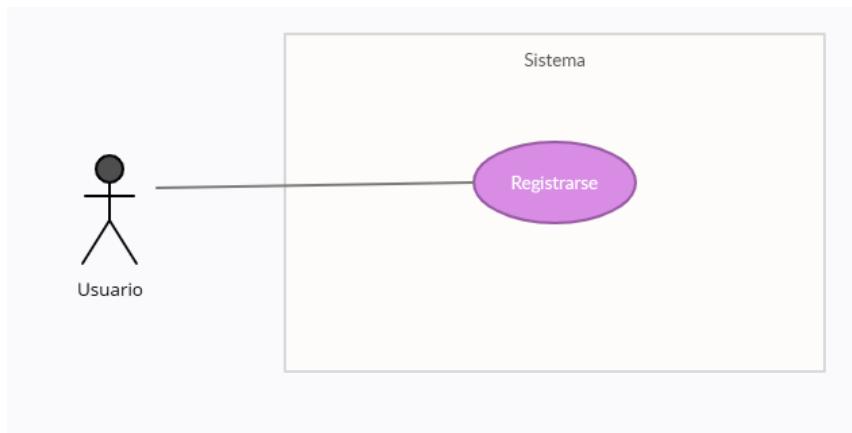
- fecha
- idEmisor
- idReceptor
- Mensaje

RI7.- Cesta: Representa la ropa que el usuario quiere comprar.

- idCesta
- idUsuario
- idPrenda
- talla

## 2. Análisis de escenarios (casos de uso y mapa de interacción).

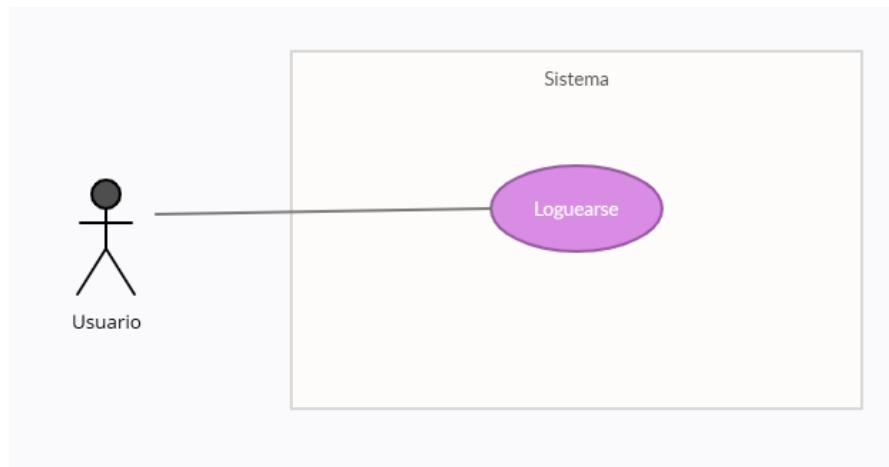
Un caso de uso es una descripción de las actividades que deben realizarse para llevar a cabo un proceso. Representan las funciones que proporciona un sistema que son de valorar para sus usuarios.



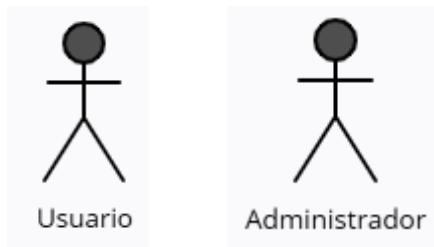
En el ejemplo anterior podemos ver un diagrama de casos de uso con dos componentes, un actor llamado usuario y un caso de uso registrarse. Estos diagramas nos ayudan a modelar los requisitos funcionales de nuestro sistema, de tal forma que veremos las relaciones que existen entre los requisitos y los actores.

## Componentes

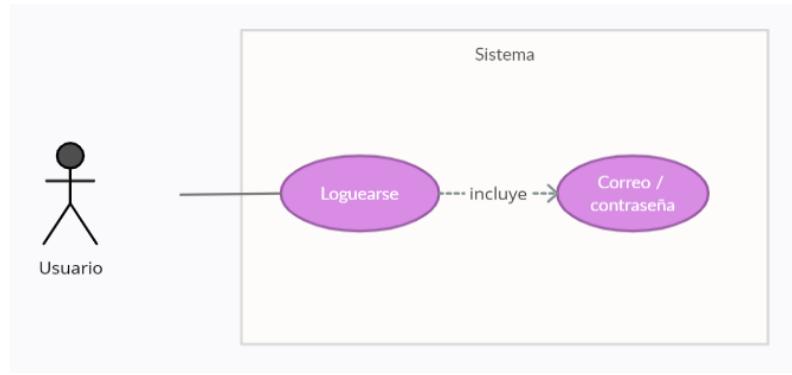
- **Caso de uso:** Son los requisitos funcionales del sistema, como en nuestro caso vamos a realizar un sistema para gestionar una tienda de ropa, tendremos un requisito que será “Loguearse”. La idea es que cuando alguien quiera acceder a la aplicación tendrá que llenar el formulario de login, esto se modelaría así:



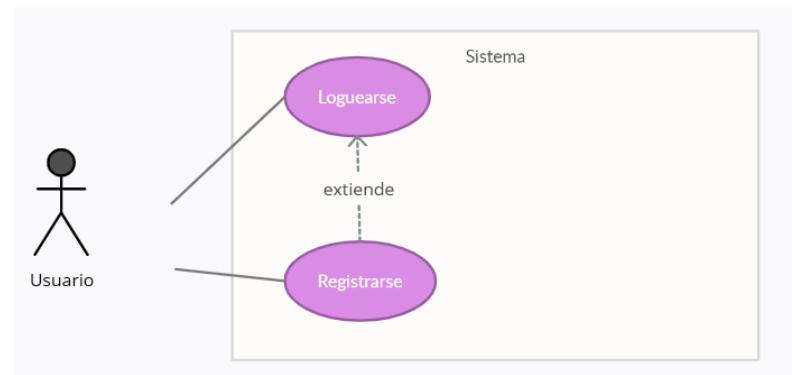
- **Actores:** Los actores son los que interactúan con el sistema. Se representan con una especie de muñeco, en nuestro caso tendríamos dos actores, que serían los administradores y los usuarios.



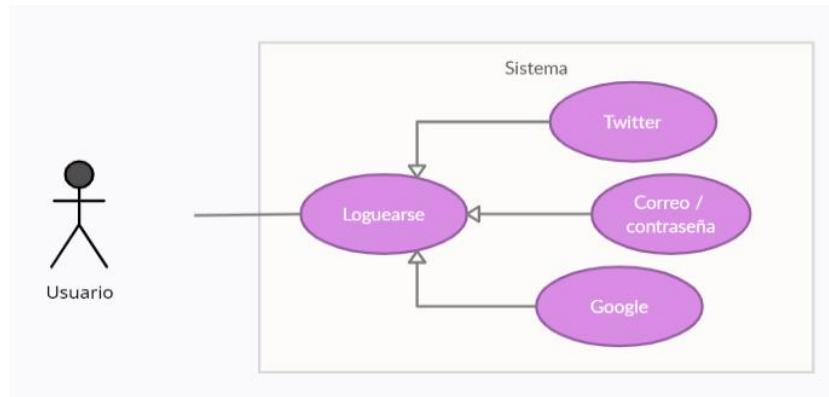
- **Relaciones:** Podemos tener relaciones entre actores y casos de uso, estas relaciones son:
  - **Asociación:** La asociación sólo es entre actores y casos de uso. Denota la participación de este actor en este caso de uso, se representa como en los ejemplos anteriores.
  - **Inclusión:** Esta relación es entre dos casos de uso. Se usa para evitar describir el mismo flujo de eventos repetidas veces. En nuestro caso un ejemplo podría ser cuando el usuario se va a loguear, el sistema tendrá que comprobar que ese usuario está registrado.



- **Extensión:** Esta relación también es entre dos casos de uso. Se utiliza cuando un caso de uso extiende el comportamiento de otro. En nuestro caso un ejemplo podría ser que el registro extiende del login.

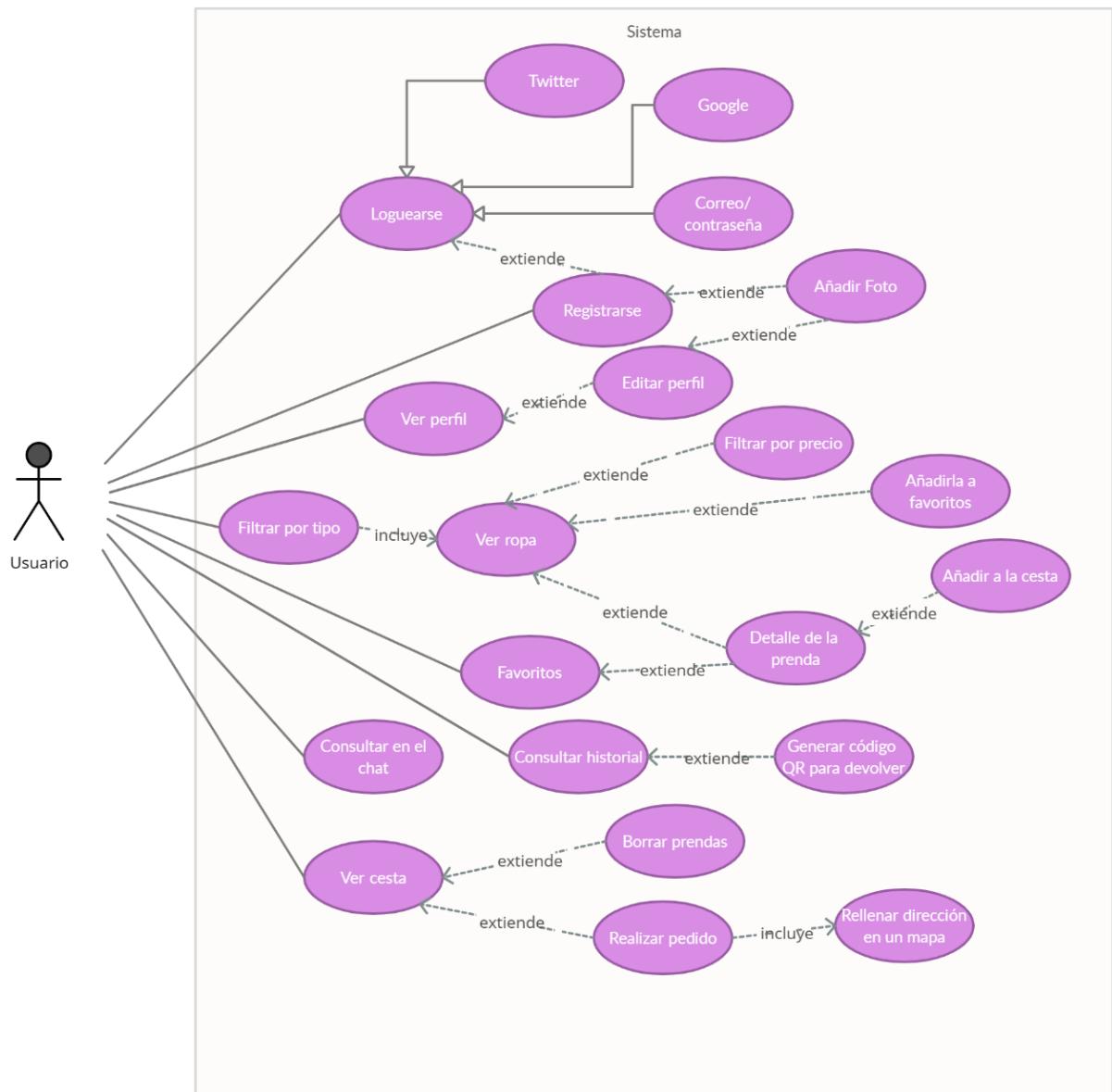


- **Generalización:** La generalización en los casos de uso es igual que la de las clases. Tenemos un caso abstracto cuyo comportamiento lo proporcionarán sus hijos. En nuestro caso un ejemplo podría ser el login ya que podemos loguearnos a través de tres métodos, google, twitter y el método tradicional.



## Diagrama de casos de uso

En el caso de B&A, el diagrama de casos de uso para el usuario sería así:



En el caso de B&A, el diagrama de casos de uso para el administrador sería así:



## Diseño de la solución

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

La interfaz de usuario de tu app es todo aquello que el usuario puede ver y con lo que puede interactuar en ella.

El objetivo de esta es mantener la interacción con los usuarios de forma más atractiva, centrándose en el diseño en ellos. Las herramientas principales que utilizan son recursos como la gráfica, los pictogramas, los estereotipos y la simbología, sin afectar el funcionamiento técnico eficiente.



Los 6 principios para el diseño de la interfaz de usuario son:

- **Familiaridad del usuario:** Utilizar términos y conceptos que se toman de la experiencia de las personas que más utilizan el sistema.
- **Consistencia:** Siempre que sea posible, la interfaz debe ser consistente en el sentido de que las operaciones comparables se activan de la misma forma.
- **Mínima sorpresa:** El comportamiento del sistema no debe provocar sorpresa a los usuarios.
- **Recuperabilidad:** La interfaz debe incluir mecanismos para permitir a los usuarios recuperarse de los errores. Esto puede ser de dos formas: Confirmación de acciones destructivas y características de ayuda sensible al contexto.
- **Guía al usuario:** Cuando los errores ocurren, la interfaz debe proveer retroalimentación significativa y características de ayuda sensible al contexto.
- **Diversidad de usuarios:** La interfaz debe proveer características de interacción apropiada para los diferentes tipos de usuario.

El color elegido es muy importante, ayuda y mejora la presentación de la interfaz, permitiendo al usuario comprender y manejar la complejidad.

Nosotros hemos seguido un estilo de colores Material design, donde nuestro color principal es este.

#F7E281

Y donde nuestro color secundario va a ser este:

#F2C1BC

Por lo tanto, nuestra paleta de colores va a ser la siguiente:

MATERIAL PALETTE      CUSTOM

---



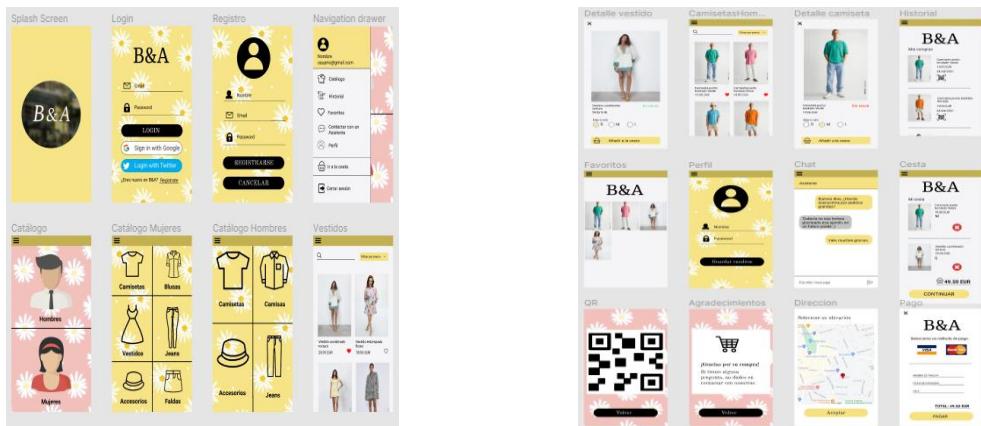
Hex color  
#f2c1bc

---

CURRENT SCHEME      RESET ALL

<p>Primary #f7e281  P</p> <p>P – Light #fffffb2</p>	<p>Secondary #f2c1bc  S RESET</p> <p>S – Light #fff4ee</p>	<p>Text on P #000000 T</p> <p>Text on S #000000 T</p>
<p>P – Dark #c2b052</p>	<p>S – Dark #bf918c</p>	

## Prototipado de B&A (Usuarios)

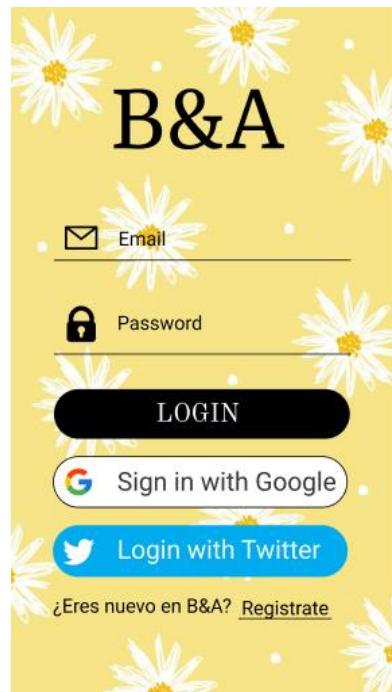


Al iniciar la aplicación lo primero que verá el usuario será una ventana con el logo de la empresa.



Pasados 3 segundos esta pantalla desaparecerá y dejará paso a la pantalla de Login, en esta pantalla tendremos varias formas de iniciar sesión, la primera es la más común en todas las aplicaciones que es mediante un correo y una contraseña (campos que podremos llenar), pero además también tendremos dos alternativas, podremos iniciar sesión con una cuenta de google o con una de twitter.

Si el usuario es nuevo en B&A y no dispone de cuentas en google ni en twitter le vamos a dar la opción de registrarse.



En esta pantalla tendremos varios campos que podremos rellenar (foto, nombre, email y password) que serán los campos del usuario. Al darle al botón de registrarse, se crea un usuario con nuestros datos y nos lleva al catálogo (que será la página principal de la app). Sin embargo, cuando pulsamos en cancelar no nos hace el registro y nos lleva de vuelta al Login.



En esta pantalla vamos a tener el menú lateral de aplicación, menú que siempre va a estar visible y donde tendremos las principales opciones que nos proporciona esta aplicación.

También se nos mostrará nuestra foto, nuestro nombre y nuestro correo en la parte superior de este menú.



Cabe destacar que la opción marcada por defecto en el menú anterior es la opción del catálogo, pantalla en la que tendremos la opción de elegir entre buscar ropa para hombres o para mujeres.



Nos aparecerán distintos tipos de menús según la opción que elijamos, en estos menús podremos seleccionar el tipo de prenda que estemos buscando para que así se nos haga más sencilla su búsqueda.

En el menú de mujeres estarán las opciones de camisetas, blusas, vestidos, jeans, accesorios y faldas.

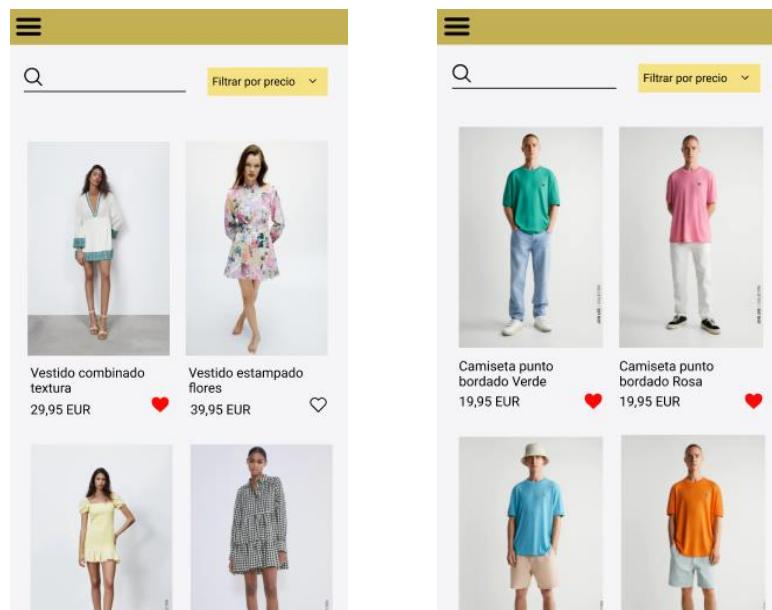
En el menú de hombres estarán las opciones de camisetas, camisas, accesorios y pantalones.



Una vez seleccionemos el tipo de prenda nos aparecerá una pantalla con una lista con ropa de ese tipo, donde al pulsar en ella nos llevará a su detalle para ver la prenda mejor y poder añadirla a la cesta.

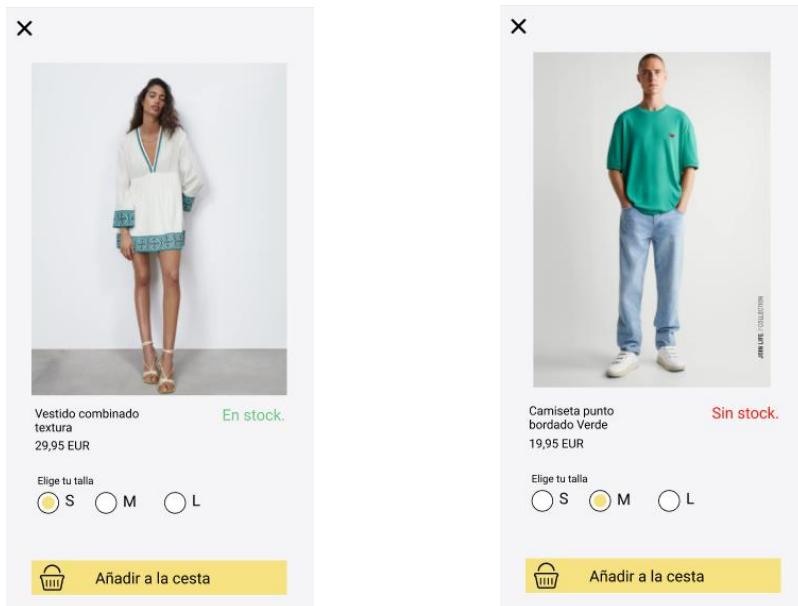
En la parte superior de esta pantalla tendremos una barra de búsqueda y un filtro donde podremos ordenar la ropa por precio.

Además, tendremos opción de añadir a favoritos si pulsamos en el corazón.



Una vez seleccionada la prenda, entraremos en el detalle de esta, donde se puede ver con mayor claridad la foto, además de mostrar más datos sobre ella, como su disponibilidad.

Si nos gusta la prenda y la queremos añadir a la cesta, primero tendremos que seleccionar nuestra talla y después le daremos al botón de añadir a la cesta.



Cuando en el menú desplegable seleccionemos la opción del historial, nos aparecerán todas las compras realizadas con su fecha de compra, su nombre y su precio.

Si queremos devolver la prenda, tendremos la opción de generar un código QR, gracias al cual un administrador nos la podrá escanear para devolverla.

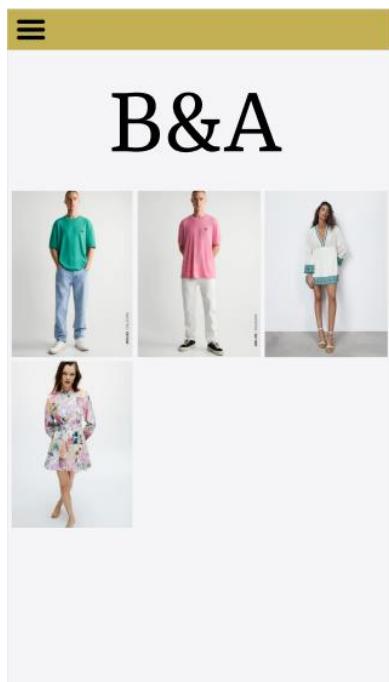


Cuando seleccionemos el ícono del QR en el historial, se nos abrirá una pantalla que nos generará el código.



Cuando en el menú desplegable seleccionemos la opción de favoritos, nos aparecerán todas las prendas que previamente hayamos marcado.

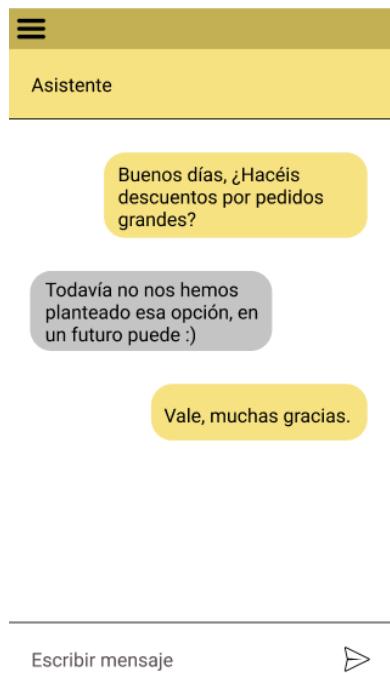
Si hacemos clic en estas prendas nos llevará al detalle de estas.



Cuando en el menú desplegable seleccionemos la opción del perfil, nos aparecerá una pantalla con nuestros datos, donde tendremos la opción de editarlos si así lo vemos conveniente.



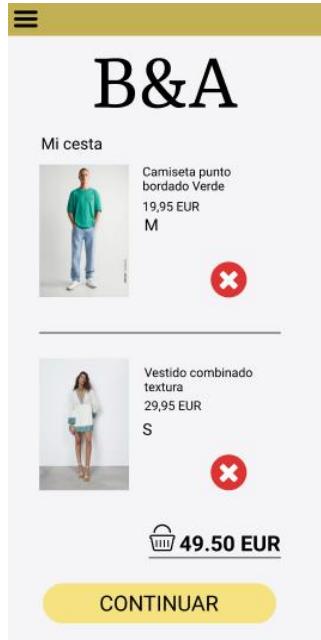
Cuando en el menú desplegable seleccionemos la opción del chat, nos aparecerá una pantalla con un chat para hablarle al administrador.



Cuando en el menú desplegable seleccionemos la opción de ir a la cesta, nos llevará a una ventana donde estarán las prendas que previamente habíamos añadido a la cesta.

Tendremos la opción de eliminar las prendas de la cesta pulsando una cruz situada debajo de la prenda.

También tendremos la opción de continuar con la compra.



Para continuar con la compra nos saldrá una pantalla en la cual tendremos que seleccionar la ubicación donde queremos recibir nuestro pedido.

Por defecto se nos marcará la ubicación donde nos encontramos en ese momento.

Seleccione su ubicación



Al pulsar aceptar, se nos abrirá una ventana donde, para finalizar la compra tendremos seleccionar el método de pago e introducir los datos de la tarjeta.

Si en algún momento nos arrepentimos de la compra, nos podremos volver a la cesta pulsando una cruz situada en la parte superior de la pantalla.

X

# B&A

Selecciona un método de pago.



NÚMERO DE TARJETA \_\_\_\_\_

FECHA DE CADUCIDAD \_\_\_\_\_

CSV2 \_\_\_\_\_

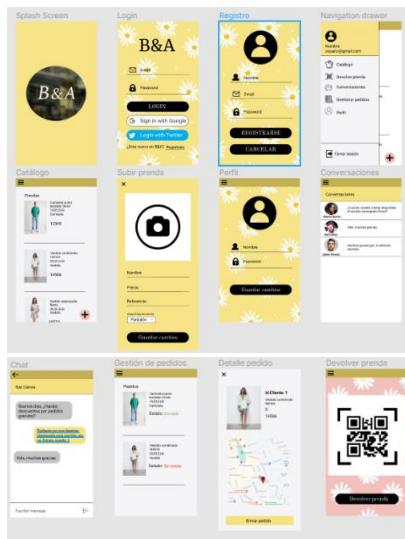
**TOTAL: 49.50 EUR**

PAGAR

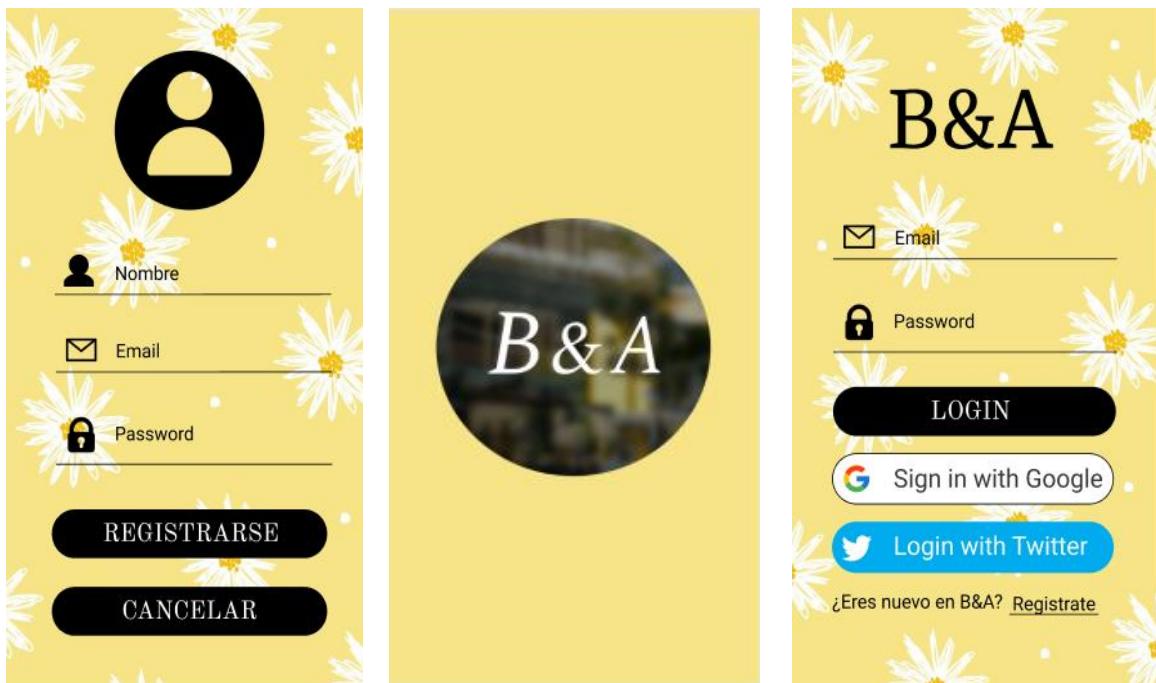
Si queremos realizar la compra, tendremos que pulsar el botón de pagar y a continuación se nos mostrará una pantalla de agradecimiento.



## Prototipado de B&A (Administrador)



Al iniciar la aplicación al igual que cuando un usuario común entra, lo primero que va a aparecer es una pantalla con el logo de la aplicación, después de tres segundos aparecerá el login donde tenemos diversas opciones para iniciar sesión, pero si no estamos registrados, tendremos una opción para hacerlo.



Una vez hayamos accedido a la aplicación como usuario administrador nuestras funciones van a ser muy diferentes comparadas a cuando un usuario común accede a la aplicación, tendremos también un menú lateral deslizable cuyas opciones iremos explicando a continuación.

Al final del menú habrá un botón para cerrar la sesión que nos llevará de vuelta al login.

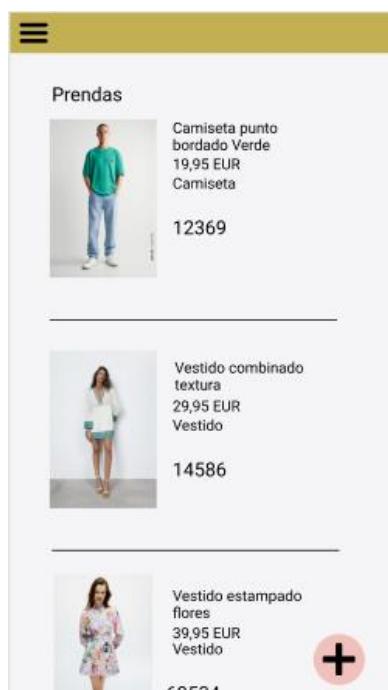
De la misma manera que los usuarios comunes, en la parte superior del menú desplegable el usuario administrador podrá ver su foto de perfil, su nombre de usuario y su correo.



Cuando en este menú seleccionamos la opción de catálogo, se nos abrirá el catálogo de ropa, pero nada tiene que ver a cuando un usuario común accede a su catálogo ya que el administrador no podrá comprar ropa ni darle a favorito, su función será añadirla, editarla o borrarla.

En este catálogo podremos ver el nombre del producto, su foto, su precio, su tipo y su número de referencia.

Podremos añadir una prenda pulsando en un botón flotante situado abajo a la derecha, la forma de editar la prenda será una vez añadida, la desplegamos hacia la izquierda y entraremos dentro de pantalla de editar, la manera que tendrá el administrador de borrar la prenda será deslizando a la derecha de la misma manera.



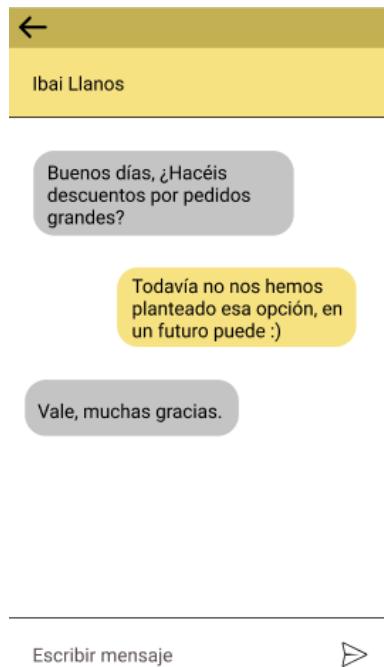
Cuando pulsamos el botón flotante se nos abre la pantalla de añadir una prenda, pantalla donde le tendremos que añadir una foto, un nombre, un precio, un número de referencia y tipo de prenda (elegido desde un menú desplegable). Cuando hacemos clic en el botón de guardar cambios se añade la prenda y volvemos al catálogo.



Cuando en el menú desplegable seleccionamos la opción de conversaciones se nos abrirá una ventana con los usuarios que se hayan puesto en contacto con nosotros para preguntar alguna duda.



Podremos contestar a los usuarios haciendo clic en las conversaciones, momento en el que se nos abrirá un chat.



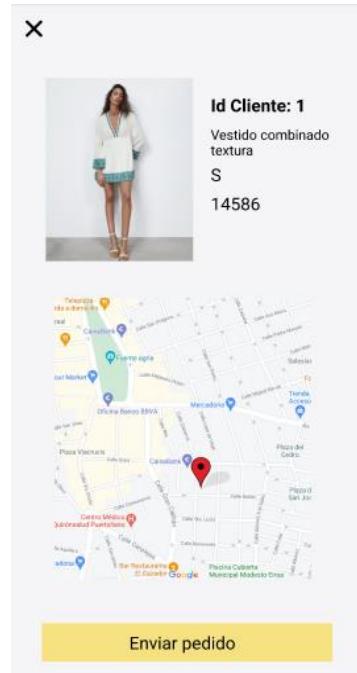
Cuando en el menú desplegable seleccionamos la opción de gestionar pedidos, se nos abrirá una pantalla con una lista con todos los pedidos realizados por los usuarios, estos pedidos tendrán una foto, el nombre del producto, el precio del producto, el tipo y el estado del pedido.



Si queremos entregar el pedido y así cambiar el estado de la entrega, tendremos que pulsar sobre él para que se nos abra el detalle de este.

En este detalle podremos ver más información, como el id del cliente y su ubicación (donde mandaremos el pedido).

Al pulsar el botón enviar pedido cambiaremos su estado.



Al igual que los usuarios comunes, al seleccionar en el menú desplegable la opción del perfil, nos aparecerá una pantalla con nuestros datos, donde tendremos la opción de editarlos si así lo vemos conveniente.



Por último, cuando elegimos la opción de devolver prenda, escaneamos el código QR del cliente que quiera devolver dicha prenda, escaneada esta, la devolvemos.



Al hacer clic en esta imagen podremos ver el video de YouTube en el que se encuentra la ejecución y explicación del Prototipado.

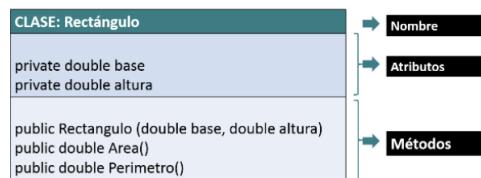
Prototipado B&A

## 2. Diagrama de clases

Un diagrama de clases es una herramienta para comunicar el diseño de un programa orientado a objetos. Nos permite modelar las relaciones entre entidades.

En UML, una clase es representada por un rectángulo de secciones filas:

- **Sección superior:** Contiene el nombre de la clase. Esta sección siempre es necesaria, ya que contiene el nombre del objeto.
- **Sección central:** Contiene los atributos de la clase. Esta sección contiene las cualidades de la clase, cada atributo de la clase está indicado en una línea separada.
- **Sección inferior:** Contiene las operaciones de la clase (métodos). Está organizado en un formato de lista. Cada operación requiere su propia línea. Las operaciones describen cómo una clase puede interactuar con los datos.



Los atributos o características de una clase pueden ser de tres tipos:

- **Público (+):** Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
- **Privado (-):** Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos pueden acceder)
- **Protected (#):** Indica que el atributo no será accesible desde fuera de la clase, pero sí podrá ser accesible por métodos de la clase, además de las subclases que deriven.

Las relaciones existentes entre las distintas clases nos indican cómo se comunican los objetos de esas clases entre sí, existen distintos tipos de relaciones:

- **Asociación:** Es una relación estructural que describe una conexión entre objetos, gráficamente se muestra como una línea continua que une las clases relacionadas entre sí.

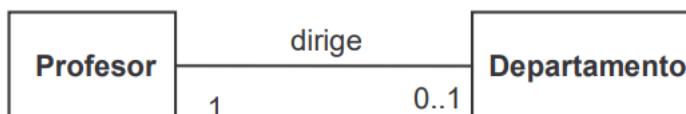


En estas relaciones existe multiplicidad, esta multiplicidad determina cuántos objetos de cada tipo intervienen en la relación. Cada asociación tiene dos multiplicidades (una para cada extremo de la relación), para indicar estas multiplicidades hay que indicar la multiplicidad mínima y la máxima (mínima..máxima).

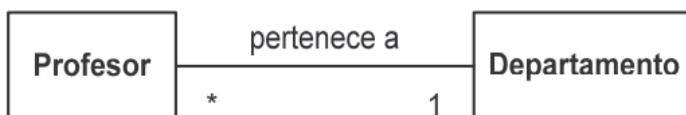
Multiplicidad	Significado
<b>1</b>	Uno y sólo uno
<b>0 .. 1</b>	Cero o uno
<b>N .. M</b>	Desde N hasta M
*	Cero o varios
<b>0 .. *</b>	Cero o varios
<b>1 .. *</b>	Uno o varios (al menos uno)

Cuando la multiplicidad mínima es 0, la relación es opciones, pero una multiplicidad mínima, mayor o igual que 1 establece una relación obligatoria.

Estos son algunos tipos de esta multiplicidad:

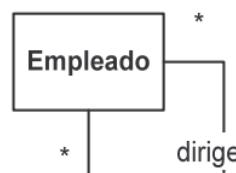


Todo departamento tiene un profesor.  
Un profesor puede dirigir un departamento.



Todo profesor pertenece a un departamento.  
A un departamento pueden pertenecer varios profesores.

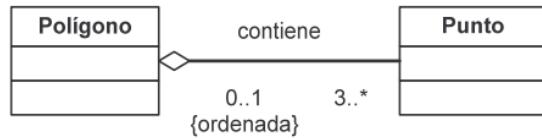
Puede haber relaciones involutivas, que es cuando la misma clase aparece en los dos extremos de la asociación.



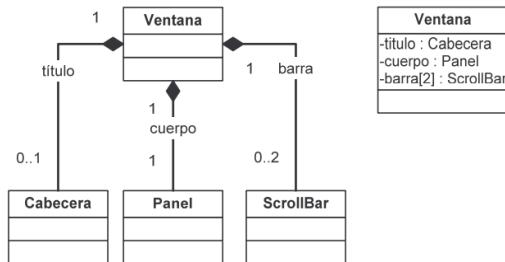
Un empleado puede dirigir a muchos empleados.  
Un empleado puede ser dirigido por muchos empleados.

- **Agregación y composición:** Relación entre un todo y sus partes, gráficamente se muestran como asociaciones con un rombo en uno de los extremos.

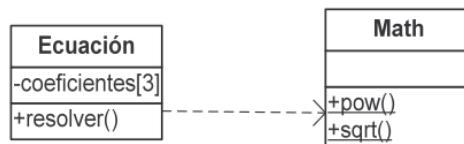
- Agregación: Las partes pueden formar parte de distintos agregados.



- Composición: Las partes sólo existen asociadas al compuesto.



- **Dependencia:** Relación que muestra la relación entre un cliente y el proveedor de un servicio usado por el cliente, gráficamente la dependencia se muestra como una línea discontinua con una punta de flecha que apunta del cliente al proveedor.



El uso de diagrama de clases tiene algunas ventajas y algunos inconvenientes.

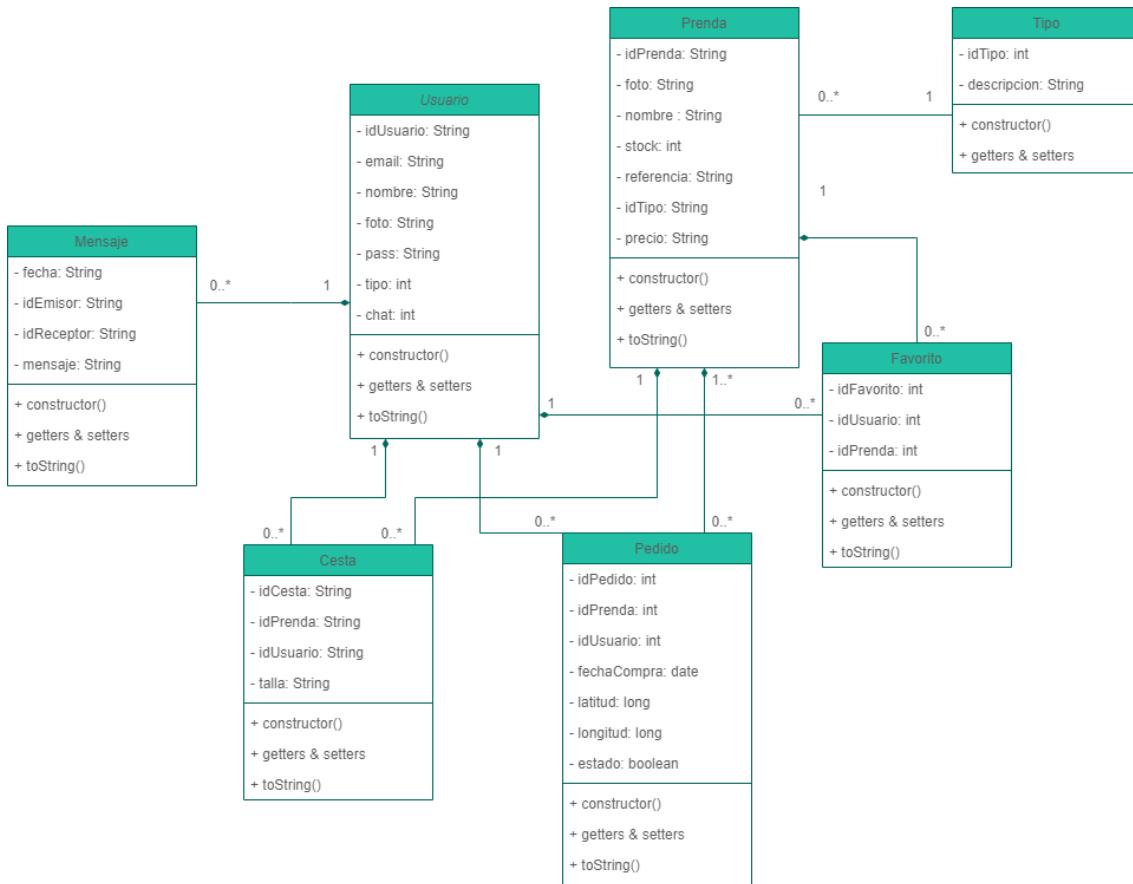
#### Ventajas:

- Propone soluciones a algunos errores.
- Representa las relaciones entre las clases del sistema.
- Se diseña los componentes de los sistemas.
- Se protegen los datos.
- Se posibilita una reducción de acoplamiento.
- Se hace más fácil la comunicación entre los programadores, descubrimiento de fallas del sistema en el diseño.

## Desventajas:

- Los diagramas de clases especifican qué clases hay y cómo están relacionadas, pero no cómo interactúan para alcanzar comportamientos particulares.
- El método tiende a ser muy lento.

## Diagrama de clases de B&A



Este es el diagrama de clases de B&A donde podemos apreciar todas las relaciones entre clases. La mayoría de las relaciones son de composición ya que si borramos los usuarios no tiene sentido que sus pedidos o sus mensajes permanezcan en la bbdd.

### Usuario - Mensaje

Existe una relación de composición entre usuario y mensaje donde un usuario puede tener 0 o muchos mensajes, pero un mensaje sólo puede ser de un usuario.

### Usuario – Pedido

Existe una relación de composición entre usuario y pedido donde un usuario podrá hacer 0 o muchos pedidos, pero un pedido sólo será de un usuario.

### **Usuario – Favorito**

Existe una relación de composición entre usuario y favorito donde un usuario podrá tener 0 o muchos favoritos, pero un favorito sólo será de un usuario.

### **Usuario – Cesta**

Existe una relación de composición entre usuario y cesta donde un usuario podrá tener 0 o muchas cestas, pero una cesta sólo será de un usuario.

### **Prenda – Pedido**

Existe una relación de composición entre prenda y pedido donde una prenda podrá pertenecer a 0 o muchos pedidos y un pedido tendrá 1 o muchas prendas.

### **Prenda – Favorito**

Existe una relación de composición entre prenda y favorito donde una prenda podrá pertenecer a 0 o a muchos favoritos, pero un favorito sólo tendrá una prenda.

### **Prenda - Tipo**

Existe una relación de asociación entre prenda y tipo ya que, aunque se elimine la prenda, el tipo seguirá estando almacenado en la base de datos. Una prenda sólo tendrá un tipo, pero un tipo podrá ser tenido por varias prendas.

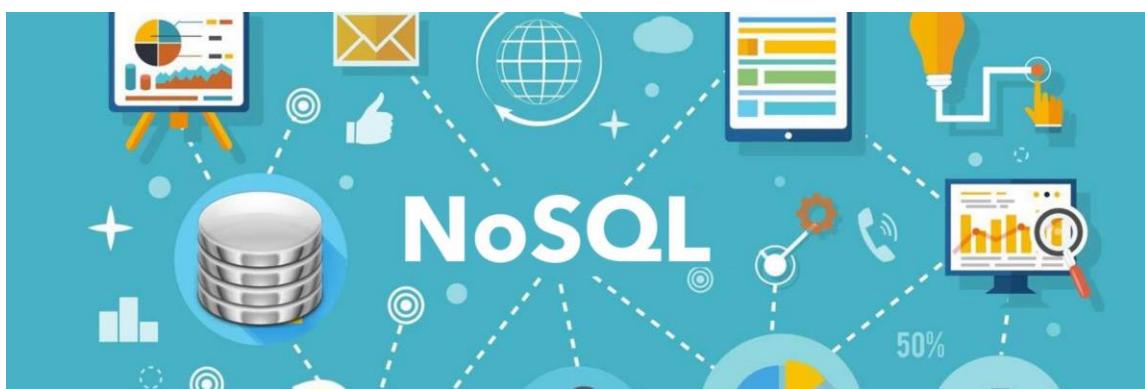
### **Prenda – Cesta**

Existe una relación de composición entre prenda y cesta donde una prenda podrá pertenecer a 0 o a muchas cestas, pero una cesta sólo tendrá una prenda.

## **3. Diseño de la persistencia de la información**

La persistencia es la capacidad de un lenguaje de programación o entorno de desarrollo de programación para, almacenar y recuperar el estado de los objetos de forma que sobrevivan a los procesos que los manipulan.

En este proyecto he decidido utilizar un modelo no relacional (NoSQL), en concreto la base de datos de Google Firebase, la razón por la que he elegido esta opción son las distintas ventajas que nos brinda este modelo que voy a exponer a continuación.

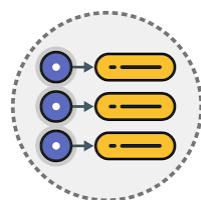


Las bases de datos NoSQL están diseñadas para modelos de datos específicos y tienen esquemas flexibles para crear aplicaciones modernas. Estas bases de datos son ampliamente reconocidas porque son fáciles de desarrollar, gracias a su funcionalidad y el rendimiento a escala.

Estas bases de datos utilizan una variedad de modelos de datos para acceder y administrar datos, están optimizadas específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles.

Hay distintos tipos de bases de datos NoSQL:

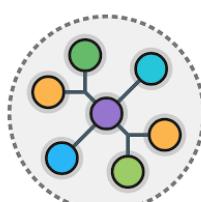
- **Clave-valor:** Las bases de datos clave-valor son altamente divisibles y permiten escalado horizontal a escalas que otros tipos de bases de datos no pueden alcanzar. Los casos de uso como juegos y tecnología publicitaria se prestan particularmente bien con el modelo de datos clave-valor.



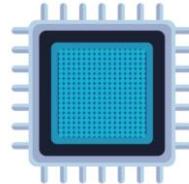
- **Documentos:** En el código de aplicación, los datos se representan a menudo como un objeto o un documento de tipo JSON porque es un modelo de datos eficiente e intuitivo para los desarrolladores. Las bases de datos de documentos facilitan a los desarrolladores el almacenamiento y la consulta de datos en una base de datos mediante el uso del mismo formato de modelo de documento que emplean en el código de aplicación. La naturaleza flexible, semiestructurada y jerárquica de los documentos y las bases de datos de documentos permite que evolucionen según las necesidades de las aplicaciones. Este modelo funciona bien con catálogos, perfiles de usuario y sistemas de administración de contenido en los que cada documento es único y evoluciona con el tiempo.



- **Gráficos:** El propósito de una base de datos de gráficos es facilitar la creación y ejecución de aplicaciones que funcionan con conjuntos de datos altamente conectados. Los casos de uso típicos para una base de datos de gráficos incluyen redes sociales, motores de recomendaciones, detección de fraudes...



- **En memoria:** Las aplicaciones de juegos y tecnología publicitaria tienen casos de uso como tablas de clasificación, tiendas de sesión y análisis en tiempo real que requieren tiempos de respuesta de microsegundos y pueden tener grandes picos de tráfico en cualquier momento.



Las bases de datos NoSQL se adaptan perfectamente a muchas aplicaciones modernas, como dispositivos móviles, web y juegos, que requieren bases de datos con unas grandes cualidades para proporcionar excelentes experiencias de usuario, algunas de estas cualidades son:

- **Flexibilidad:** Las bases de datos NoSQL generalmente ofrecen esquemas flexibles que permiten un desarrollo más rápido y más iterativo. El modelo de datos flexible hace que las bases de datos NoSQL sean ideales para datos semiestructurados y no estructurados.
- **Escalabilidad:** Las bases de datos NoSQL generalmente están diseñadas para escalar usando clústeres distribuidos de hardware en lugar de escalar añadiendo servidores caros y sólidos.
- **Alto rendimiento:** Las bases de datos NoSQL están optimizadas para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que el intento de lograr una funcionalidad similar con bases de datos relacionales.
- **Altamente fucionales:** Las bases de datos NoSQL proporcionan APIs altamente funcionales y tipos de datos que están diseñados específicamente para cada uno de sus respectivos modelos de datos.

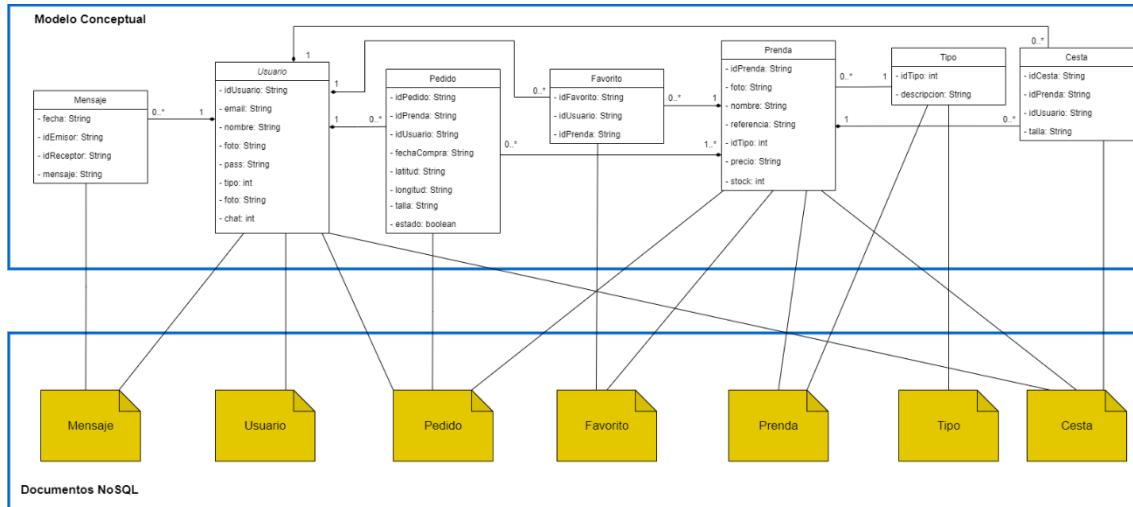
**En definitiva, una vez expuestas y analizadas todas las ventajas de estas bases de datos, y teniendo en cuenta los requisitos que vamos a tener en B&A. Esta es la opción por la que me voy decantar, ya que va a ser una base de datos sujeta a muchos cambios, a un gran crecimiento y queremos que nuestra base de datos cuente con la mayor optimización y rapidez posible.**

#### Diagrama de persistencia de datos de B&A

Ahora vamos a mostrar el diagrama de la base de datos NoSQL de B&A. En la parte superior tendremos las clases y las relaciones entre ellas, y en la parte inferior tendremos los documentos donde se guardan los datos y estarán relacionados con las clases.

Cuando hacemos referencia a más de una clase significará que ese documento tendrá almacenado los datos de una, pero también hará referencia a otra clase.

Por ejemplo, al traernos de la base de datos a un usuario, no quiero traerme también sus pedidos, ya que es innecesario y ralentizaría mucho la base de datos, si quiero acceder a sus pedidos, consultaría el documento pedidos.



- **Mensaje:** Tendrá la información de la clase Mensaje y el idEmisor e idReceptor serán las referencias de la clase Usuario.
- **Usuario:** Tendrá la información de la clase Usuario.
- **Pedido:** Tendrá la información de la clase Pedido, el idUsuario, que hará referencia a la clase Usuario y el idPrenda que hará referencia a la clase Prenda.
- **Favorito:** Tendrá la información de la clase Favorito y el idPrenda que hará referencia a la clase Prenda.
- **Prenda:** Tendrá la información de la clase Prenda y el idTipo que hará referencia a la clase Tipo.
- **Tipo:** Tendrá la información de la clase Tipo.
- **Cesta:** Tendrá la información de Cesta y el idPrenda hará referencia a la clase Prenda y el idUsuario a la clase Usuario.

#### 4. Arquitectura del sistema

Hace tiempo, la programación se consideraba un arte y se desarrollaba como tal debido a la dificultad que conllevaba para la mayoría de las personas. Pero con el tiempo se han ido desarrollando formas y guías generales, a estas se les ha denominado arquitectura de software.

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema.

Consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software.

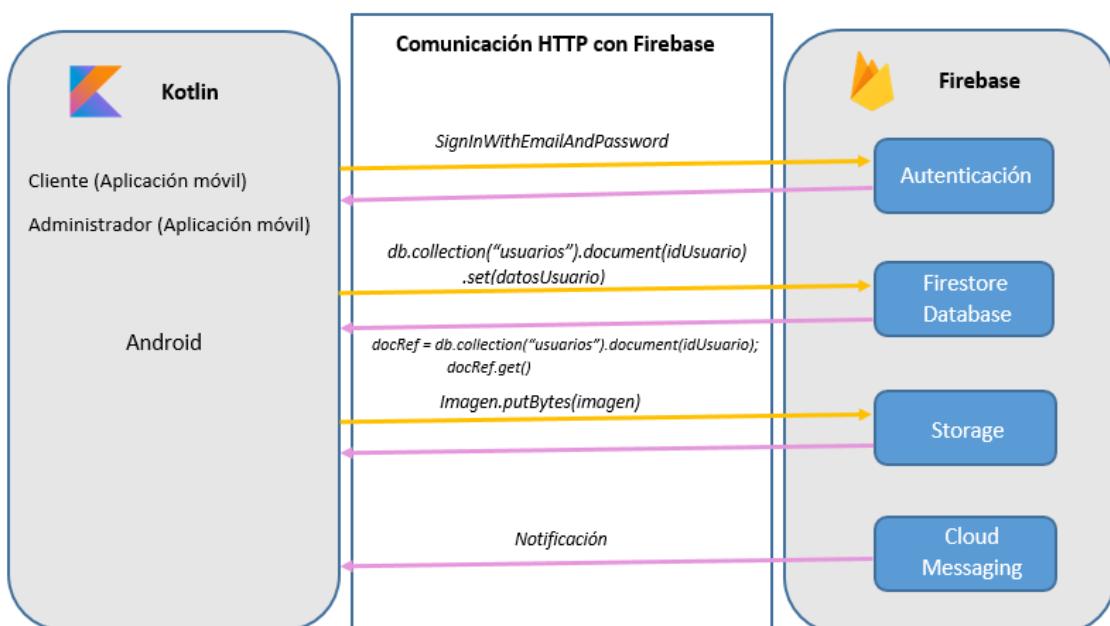
Esta arquitectura se selecciona y diseña con base en objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información y las restricciones son aquellas imitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas.

Por lo que la arquitectura de software define, de manera abstracta los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos. Toda arquitectura debe ser implementable en una arquitectura física.



Diagrama de arquitectura de B&A

A continuación, vamos a mostrar el diagrama de arquitectura de B&A el cual nos va a ayudar a visualizar la estructura general de alto nivel de la aplicación, con el fin de garantizar la satisfacción de las necesidades de los usuarios. En este diagrama podremos distinguir el cliente, el servidor y las comunicaciones que se realizan.



La aplicación de B&A está formada por un cliente en aplicación móvil y un administrador en aplicación móvil ambas para Android utilizando Firebase.

- **Autenticación:** Tanto el cliente como el administrador se autenticarán usando el servicio de Firebase, este servicio ofrecerá autenticación en correo/contraseña, google y twitter.
- **Firestore Database:** Es una base de datos de firebase que usaremos para actualizar la información de los usuarios, favoritos, prendas, pedidos y cesta.
- **Storage:** Esta funcionalidad nos va a permitir almacenar las imágenes.
- **Cloud Messaging:** Nos enviará notificaciones.

## Implementación de la solución

### 1. Análisis tecnológico

A la hora de empezar con un proyecto es muy importante elegir bien las tecnologías implicadas, las tecnologías que se van a usar y esta es una decisión difícil ya que hay tecnologías que se ajustan más a nosotros en lo personal (las tenemos más o menos dominadas y nos gustan más o menos) y tecnologías que se ajustan más o menos a nuestro proyecto, y a veces hay que elegir y dependiendo de muchos factores usaremos unas tecnologías u otras.

En este apartado voy a exponer algunas de las tecnologías que he estado tanteando, las voy a definir, voy a decir sus principales características, y al final voy a decir la conclusión y a la decisión que he llegado.

#### Tecnología del cliente

### Java

Java es un lenguaje de programación orientado a objetos que se incorporó al ámbito de la informática en los años noventa por Sun Microsystems. Esta opción tiene muchas ventajas a la par que inconvenientes.



- **Es un lenguaje simple:** Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos además estoy bastante familiarizada con este lenguaje ya que lo llevo usando algunos años.
- **Es un lenguaje orientado a objetos.**
- **Es distribuido:** Gracias y las librerías y herramientas que este lenguaje posee se puede conseguir que los programas sean distribuidos y que corran en varias máquinas interactuando entre sí.

- **Es robusto:** Realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores lo antes posible.
- **Seguridad:** Las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de cientos de virus.
- **Interpretado:** El intérprete de Java puede ejecutar directamente el código objeto, esto, normalmente consume menos recursos que al compilarlo.
- **Multihilo:** Al ser un lenguaje multihilo permite muchas actividades simultáneas en un programa. Estos hilos son básicamente pequeños procesos o piezas independientes de un gran proceso.

## Kotlin

Kotlin es un lenguaje de programación de tipado estático que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript. Es desarrollado principalmente por JetBrains. A continuación, voy a profundizar diciendo algunas características de este lenguaje para el desarrollo de Android.



- **Importación de diseño estático:** Uno de los códigos que más repetimos en el desarrollo de la aplicación Android con Java es utilizar la función `findViewById()`, función que nos permite obtener las referencias a las vistas ubicadas en las actividades.

Kotlin nos va a permitir importar todas las referencias a las vistas desde el diseño con una sola importación.

- **Escribir clases POJO con Kotlin:** Algo que nos va a ahorrar mucho tiempo en Kotlin es la facilidad y la rapidez con la que se pueden hacer clases POJO.
- **Herencia de clases y constructores:** Las clases en Kotlin tienen un constructor primario y uno o más constructores secundarios.
- **Herencia en Kotlin:** En Kotlin todas las clases extienden de `Any`, que es como `Object` en Java y por defecto todas las clases están cerradas, o son clases finales.
- **Múltiples beneficios:** el soporte de Android Studio es prácticamente perfecto, otra gran ventaja es que es compatible con java pudiendo coexistir en el mismo proyecto.

## React Native

React Native es un framework JavaScript para crear aplicaciones reales nativas para IOS y Android, basado en la librería de JavaScript React para la creación de componentes visuales, cambiando el propósito de los mismos para, en lugar de ser ejecutados en el navegador, correr directamente sobre las plataformas móviles nativas, en nuestro caso Android. A continuación, voy a exponer algunas características de este framework.



- **Compatibilidad Cross-Platform:** Lo cual nos va a ayudar a los desarrolladores a crear aplicaciones que pueden ser ejecutadas tanto en iOS como en Android simultáneamente con el mismo código base.
- **Funcionalidad nativa:** La unión de React Native junto con JavaScript permite la ejecución de aplicaciones más complejas de manera suave, mejorando incluso el rendimiento de las apps nativas y sin el uso de un WebView.
- **Actualizaciones instantáneas:** Con la extensión de JavaScript, los desarrolladores tienen la flexibilidad de subir los cambios contenidos en la actualización directamente al dispositivo del usuario sin tener que pasar por las tiendas de aplicaciones propias de cada sistema.
- **Sencilla curva de aprendizaje:** ReactNative es bastante fácil de leer y sencillo de aprender ya que se basa en los conceptos fundamentales del lenguaje JavaScript.

## Flutter

Flutter es un framework para desarrollar aplicaciones para diferentes plataformas. Elaborado por Google y publicado por primera vez como proyecto de código abierto a finales de 2018. Este kit de desarrollo ofrece un gran número de bibliotecas para elementos estándar de la interfaz de usuario de Android e iOS, pero también sirve para desarrollar aplicaciones web de escritorio.



El uso principal de este framework es el desarrollo de aplicaciones de Android e iOS, sin necesidad de escribir un código base propio para cada uno de estos sistemas completamente diferentes entre sí.

El SDK de Flutter se basa en el lenguaje de programación Dart, también desarrollado por Google, con el fin de convertirse en un sucesor del clásico JavaScript que, igual que este, se ejecuta directamente en el navegador. Algunas de sus principales características son:

- **Desarrollo rápido:** Los widgets son soluciones pre construidas de la interfaz, así que se pueden crear interfaces rápidamente utilizando widgets en lugar de escribirlas desde cero. Además, posee una característica llamada Hot Reload, que permite ver los cambios en caliente sin necesidad de esperar a recomilar.
- **Interfaz flexible y expresiva:** Estos widgets permiten construir interfaces muy rápido, además son altamente personalizables.
- **Rendimiento nativo multiplataforma:** Los widgets añaden personalizaciones para iOS y Android, como: navegación, scrolling, iconos, fuentes, ... Así no tienes que preocuparte de las peculiaridades de cada sistema.

## Xamarin

Xamarin es una plataforma de código abierto, adquirida por Microsoft, para compilar aplicaciones modernas. Esta plataforma nos facilita la tarea de desarrollar cualquier tipo de aplicación/juego de forma nativa para las plataformas más usadas: iOS, Android y Windows. Para ello nos brinda acceso al 100% de las APIs nativas más las APIs comunes de .NET, con un mismo lenguaje de programación C#. Algunas de las principales características de esta plataforma son:



- **Compartir código:** Además de compartir un mismo lenguaje y entorno de desarrollo, podemos utilizar un mismo patrón de desarrollo.
- **Completa cobertura de las APIs de iOS y Android:** Cualquier cosa que se pueda hacer con Objetive-C/Swift o Java, se puede hacer con C# y Xamarin.
- **Aplicaciones nativas:** Las aplicaciones desarrolladas con Xamarin son 100% nativas.
- **Siempre actualizado:** Xamarin suele añadir soporte el mismo día del lanzamiento oficial de una actualización.
- **Open source y gratis:** Tras la compra de Xamarin por parte de Microsoft, pasó a ser Open Source y gratuito.

Para la parte del cliente voy a usar Kotlin, ya que sus características pueden hacer que nuestro proyecto en Android sea más eficiente, Kotlin ofrece la posibilidad de ahorrar mucho tiempo en el momento de escribir nuestra aplicación mediante una sintaxis intuitiva y concisa. Todavía es un lenguaje joven, pero es un lenguaje que se va a usar mucho en un futuro no tan lejano.

Tecnología del servidor

## SpringBoot

SpringBoot es una herramienta que nace con la finalidad de simplificar aún más el desarrollo de aplicaciones basadas en el framework Spring Core. Spring Boot busca que el desarrollador solo se centre en el desarrollo de la solución, olvidándose por completo de la compleja configuración que actualmente tiene Spring Core para poder funcionar.



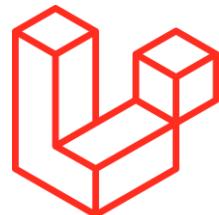
Estas son las principales características que tiene Spring Boot:

- **Configuración:** Cuenta con un complejo módulo que auto configura todos los aspectos de nuestra aplicación para poder simplemente ejecutar la aplicación sin tener que definir absolutamente nada.

- **Resolución de dependencias:** Solo hay que determinar qué tipo de proyecto estamos utilizando y este se encargar de resolver todas las librerías y dependencias para que este funcione.
- **Despliegue:** Puede desplegar las aplicaciones mediante un servidor web integrado, como es el caso de Tomcat.
- **Métricas:** Por defecto, Spring Boot cuenta con servicios que permiten consultar el estado de salud de la aplicación, permitiendo así saber si la aplicación está encendida o apagada y la memoria utilizada y disponible entre otras muchas cosas.
- **Extensible:** Permite la creación de complementos, los cuales ayudan a que la comunidad de Software Libre cree nuevos módulos que faciliten aún más el desarrollo.

## Laravel

Laravel es un framework código abierto para desarrollar aplicaciones y servicios web con PHP 5. Su objetivo es desarrollar aplicaciones con código PHP de forma elegante y simple. Fue creado en 2011 y tiene una gran influencia en otros framework. Es joven y con gran futuro. Cuenta con una comunidad llena de energía y una documentación atractiva de contenido claro y completo. Algunas de las características que posee son:



- **El motor de plantilla:** se llama Blade y da numerosas posibilidades para hacer unas páginas visualmente muy potentes y eficaces.
- **Arquitectura:** tiene una arquitectura MVC (Modelo – Vista – Controlador) que da muchas facilidades para relacionar de manera clara y sencilla todas las partes de una aplicación.
- **Eloquent ORM:** Es muy intuitivo para escribir consultas PHP sobre objetos.
- **Seguridad:** Ofrece un nivel bastante fuerte con mecanismos de hash y salt para encriptar por medio de librerías como BRCrypt.
- **Artisan:** Su sistema de comandos otorga al framework gran poder y a los programadores grandes facilidades y posibilidades, para crear controladores, entidades o actualizar la base de datos.
- **Librerías y modularidad:** Laravel aparte de sus propias librerías cuenta con ayuda de Symfony entre otras muchas.

## Firebase

Firebase es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Está disponible en distintas plataformas (iOS, Android y web), por lo que es más rápido trabajar en el desarrollo.



Fue creada en 2011 pero pasó a ser parte de Google en 2014 comenzando como una base de datos en tiempo real. Sin embargo, se añadieron más funciones y características:

- **Realtime database:** Es una de las herramientas más destacadas y esenciales de Firebase, son bases de datos en tiempo real. Estas se alojan en la nube, son NoSQL y almacenan los datos como JSON. Permiten alojar y disponer de los datos e información de la aplicación en tiempo real.
- **Autenticación de usuarios:** Ofrece un sistema de autenticación de usuarios que permite tanto el registro propiamente dicho (mediante email y contraseña) como el acceso utilizando perfiles de otras plataformas externas (Google, Facebook...).
- **Almacenamiento en la nube:** Cuenta con un sistema de almacenamiento donde los desarrolladores puedan guardar los ficheros de sus aplicaciones. Este almacenamiento es de gran ayuda para tratar los archivos de los usuarios.
- **CrashReporting:** Esta funcionalidad detecta y ayuda a solucionar los problemas de la app, consiguiendo un informe de errores muy detallado y organizado.
- **TestLab:** Con esta opción podremos testear la app en dispositivos Android virtuales basados en los parámetros que configuremos.
- **Remote Config:** La configuración remota nos va a servir para modificar ciertas funciones, aspectos o incluso la apariencia de la aplicación sin que sea necesario publicar una actualización.
- **Cloud Messaging:** Esta funcionalidad nos permitirá el envío de notificaciones y mensajes a diversos usuarios en tiempo real y a través de varias plataformas.

La función principal de Firebase es hacer más sencilla la creación de tanto aplicaciones webs como móviles y su desarrollo, procurando que el trabajo sea más rápido, pero sin renunciar a la calidad requerida.

## AWS

Amazon Web Services (AWS) es una plataforma de servicios de nube que proporciona una variedad de servicios de infraestructura tales como almacenamiento, redes, bases de datos, servicios de aplicaciones, potencia de cómputo, entre otros, los cuales permiten el crecimiento de las empresas.



- **Funcionalidad:** Cuenta con una cantidad de servicios y características incluidas.
- **Gran comunidad:** Tiene una de las comunidades más grandes y dinámicas del mercado, con millones de clientes activos y decenas de miles de socios en todo el mundo.
- **Seguro:** Está diseñado para ser uno de los entornos de informática de nube más flexible y seguro.

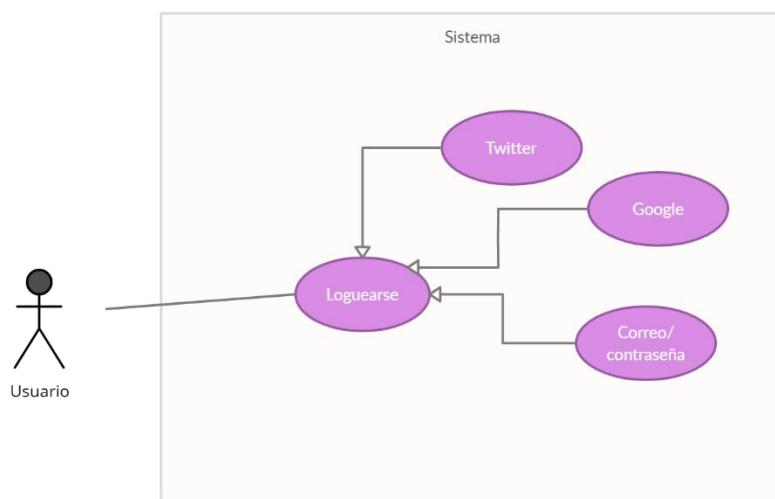
- **Innovación:** Utiliza las últimas tecnologías para experimentar e innovar de forma más rápida.

Para la parte del backend voy a usar Firebase ya que es una plataforma muy completa y con grandes y numerosos beneficios que terminan por dejar en segundo plano las posibles desventajas que presenta. Una de estas desventajas podría ser la necesidad de pago, pero para nuestro proyecto que se encuentra en las primeras etapas la versión gratuita de Firebase es más que suficiente.

## 2. Elementos a implementar

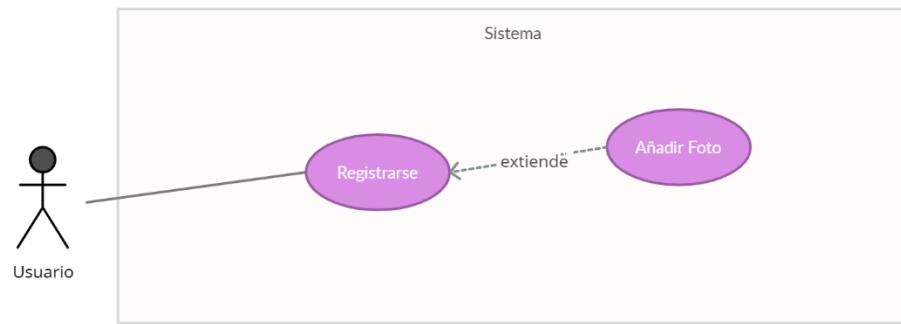
Vamos a exponer los diferentes elementos que vamos a implementar, lo vamos a hacer siguiendo los diferentes casos de uso que previamente hemos expuesto en los diagramas de casos de uso ([punto 3.2.2](#)).

### - Login



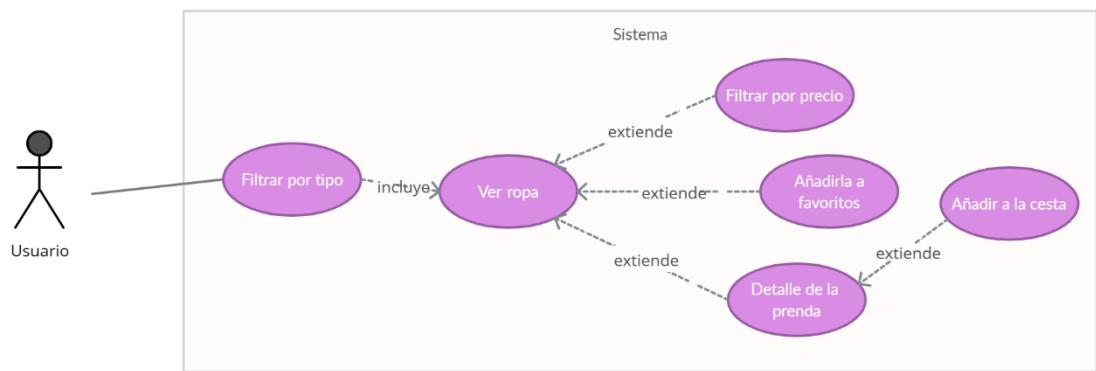
En el Login, utilizando la tecnología de autenticación y de base de datos de Firebase, los usuarios normales podrán, no solo iniciar sesión a través de correo y contraseña, sino que también lo podrán hacer a través de sus cuentas de Google y Twitter. Sin embargo, los usuarios administradores solo podrán acceder a través de correo y contraseña.

- **Registro**



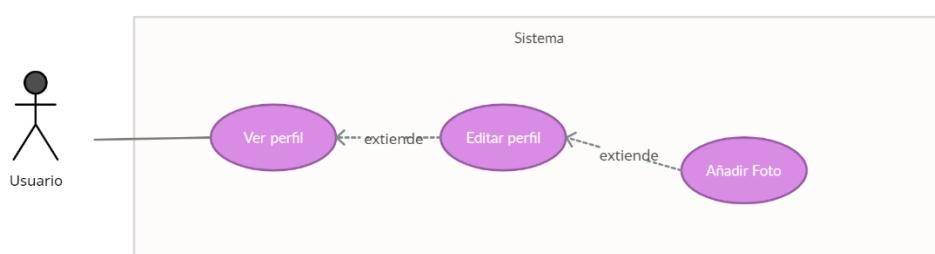
En el registro, utilizando la tecnología de autenticación, base de datos y almacenamiento de Firebase, los usuarios normales podrán crearse una cuenta en B&A.

- **Catálogo**



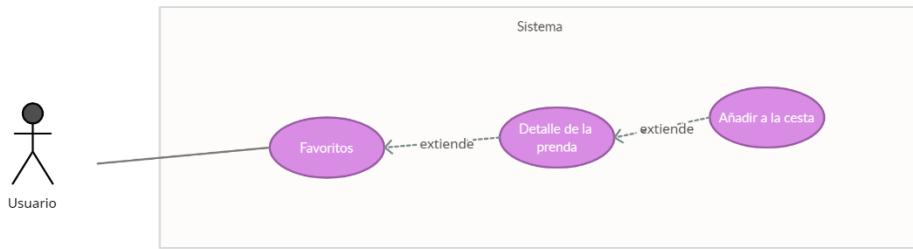
Cuando el usuario normal quiera ver las prendas, tendrá que decidir primero el tipo de ropa que quiere ver, esta selección se hará gracias a un filtro. Posteriormente, ya en la lista de ropa, el usuario tendrá varias opciones, podrá reordenar la lista por precio, añadir a favoritos una prenda o acceder al detalle de la prenda para después añadirla a la cesta.

- **Perfil**



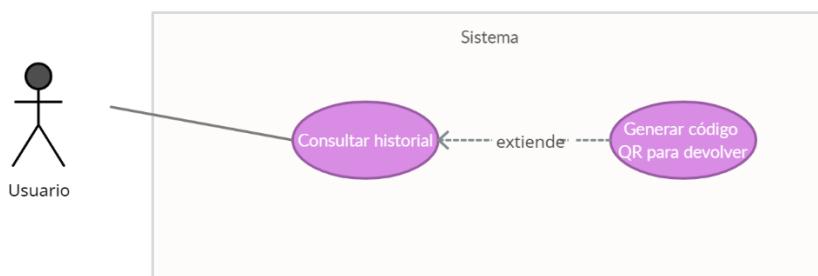
Cuando tanto los usuarios normales como los administradores quieran ver los datos de su perfil, podrán hacerlo, teniendo la posibilidad de editar estos datos, incluida la foto.

### - **Favoritos**



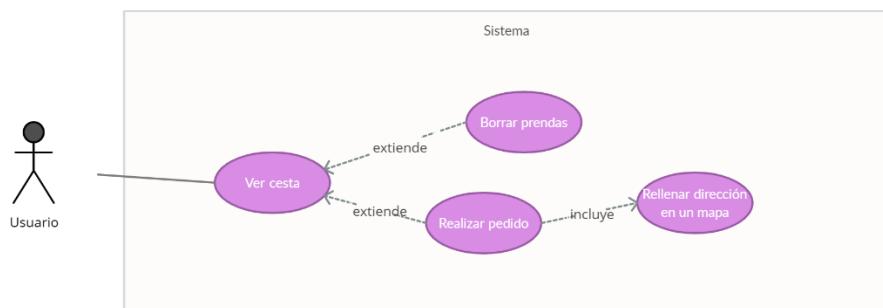
El usuario normal tendrá la opción de ver en una lista, las prendas a las que le ha dado favoritos, si hacemos clic en esta prenda, nos llevará al detalle de esta, teniendo la posibilidad de añadirla a la cesta.

### - **Historial**



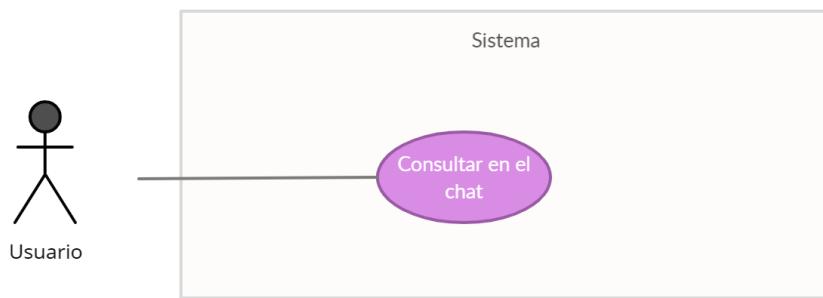
Si el usuario normal lo desea, tendrá la posibilidad de ver su historial de compras, pudiendo generar un código QR con los datos de la compra para una posible devolución.

### - **Cesta**



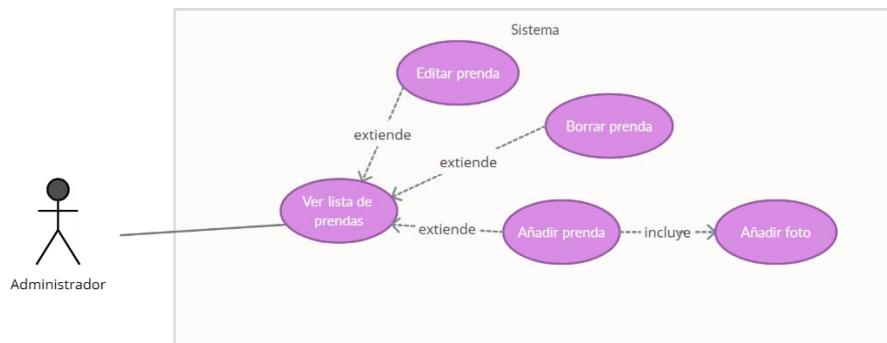
El usuario normal tendrá a su disposición una opción para ver la cesta, aquí estarán las prendas que previamente el usuario ha seleccionado. En la cesta podrá tanto borrar las prendas que finalmente no quiera comprar, como realizar el pedido, teniendo que introducir la dirección del envío en un mapa.

- **Contactar con un asistente**



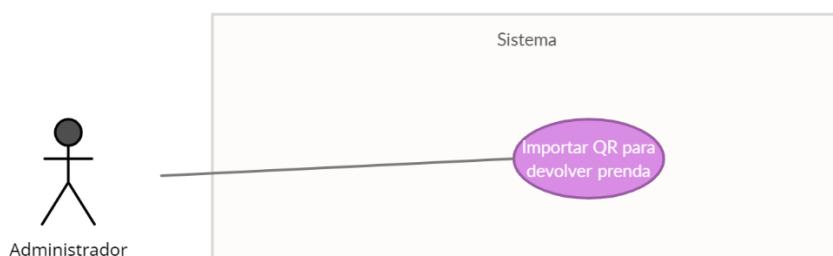
El usuario normal tendrá la posibilidad de contactar con el asistente a través de un chat, para posibles dudas que le surjan.

- **Gestión de prendas**



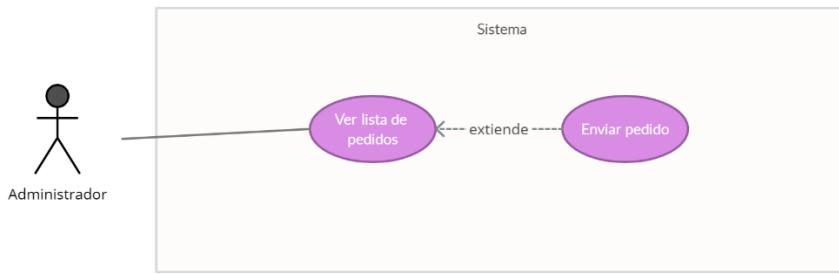
El usuario administrador tendrá la opción de ver una lista con las prendas que él ha subido (solo habrá un administrador). Una vez en la lista, tendrá la posibilidad de editar o borrar la prenda que desee, también podrá añadir prendas nuevas, donde la foto será obligatoria.

- **Devolver prenda**



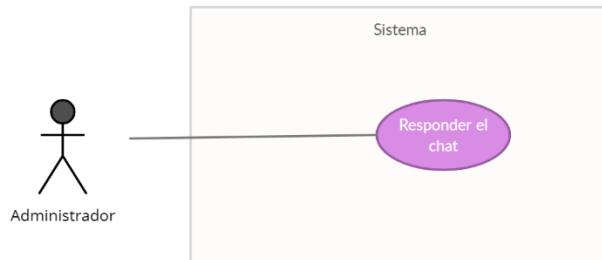
El usuario administrador tendrá la opción de devolver una prenda escaneando el código QR con los datos de la compra, proporcionado por el usuario.

## - Pedidos



El usuario administrador podrá ver la lista de pedidos realizados por los usuarios y enviarlos.

## - Conversaciones



El usuario administrador podrá responder a través de un chat las consultas que tengan los clientes.

### 3. Aspectos esenciales de la implementación

Una vez expuestos los elementos a implementar, voy a explicar detalladamente su desarrollo uno por uno.

El primer aspecto a desarrollar es el **login**, habrá varias formas de iniciar sesión, podremos hacerlo a través de correo/contraseña y a través de Google.

```
//COMPROBAMOS SI ESTA AUTENTICADO EN BYA
Auth.signInWithEmailAndPassword(email, pass).addOnCompleteListener { it: Task<AuthResult>?

    if (it.isSuccessful) {//SI LO ESTA
        idUsuario = Auth.currentUser.uid
        val pref = getSharedPreferences("Preferencias", Context.MODE_PRIVATE).edit()
        pref.putString("idUsuario", idUsuario)
        pref.apply()

        entrarMain()//ENTRAMOS EN LA APP
    } else {
        Snackbar.make(context: this, login, text: "El email o la contraseña son incorrectos", Snackbar.LENGTH_LONG).
    }

}
```

Aquí vemos como una vez rescatados y comprobados que no son nulos los campos de correo y contraseña que el usuario ha introducido, comprobamos que el usuario existe en la autenticación, si existe, guardamos el id de ese usuario en nuestras SharedPreferences, para posteriores gestiones y entramos en la aplicación.

Como hemos dicho anteriormente, también existe la posibilidad de acceder a la aplicación a través de **Google**.

```
if(account != null){//Si la cuenta existe, intentamos loguearnos con Google
    val credential = GoogleAuthProvider.getCredential(account.idToken, accessToken: null)
    FirebaseAuth.getInstance().signInWithCredential(credential).addOnCompleteListener{ it: Task<AuthResult> }

    if(it.isSuccessful){//Si ha salido bien

        idUsuario = Auth.currentUser.uid//Recogemos su id de la autenticación y lo guardamos en la sharedPreferences
        val pref = getSharedPreferences(name: "Preferencias", Context.MODE_PRIVATE).edit()
        pref.putString("idUsuario", idUsuario)
        pref.apply()

        var existe = false
        //Comprobamos si el usuario esta en la bbdd de firestore
        db.collection(collectionPath: "usuarios")
            .get()
            .addOnSuccessListener { result ->
                for(usuario in result){

                    if(usuario.get("idUsuario").toString().equals(idUsuario)) {
                        existe = true//El usuario existe
                    }
                }
            }

        if(!existe){//Si no existe
            //Añadimos el usuario a la bbdd de B&A
            val u = Usuario(idUsuario, account.displayName.toString(), account.email.toString(), pass: "", account)
            db.collection(collectionPath: "usuarios").document(idUsuario).set(u)
        }
    }

    google = true//PARA QUE NO PUEDA EDITAR SU PERFIL
    entrarMain()//Entramos al main
}
}

else{
```

Comprobamos que la cuenta no sea nula, una vez comprobado esto nos logueamos con Google y, si todo ha ido bien, guardamos su id de usuario en nuestras SharedPreferences.

Si es la primera vez que el usuario inicia sesión, lo insertamos en nuestra base de datos Firestore, para futuras gestiones, una vez hecho esto, entraríamos al main indicando que el usuario viene desde Google, gracias a esto los usuarios que vengan de Google, no tendrán la opción de editar su perfil.

Cabe destacar, que existen dos menús, el **menú del administrador** y el **menú de usuarios normales**.

```

    /**
     * Método que nos lleva a la actividad Main
     */
    private fun entrarMain(){
        var tipo = ""
        //Consultamos el tipo de usuario (admin, normal)
        val pillarTipo = db.collection(collectionPath: "usuarios").document(idUsuario)

        pillarTipo.get().addOnSuccessListener { it: DocumentSnapshot!
            tipo = it.get("tipo").toString()//Recogemos el tipo
            main(tipo)//Entramos al main
        }
    }

    /**
     * Método que nos lleva al main
     */
    private fun main(tipo : String){

        if(tipo.equals("1")){//Si el tipo es 1, nos lleva al main de los usuario normales
            val main = Intent(packageContext: this, MainActivity::class.java)
            main.putExtra(name: "Google", google)//Le indicamos si viene de google o no
            startActivity(main)
        }else{//Si no, nos lleva al main del administrador
            val main = Intent(packageContext: this, AdministradorActivity::class.java)
            startActivity(main)
        }
    }
}

```

Dependiendo del tipo de usuario que esté intentando acceder a la aplicación, tendrá acceso al menú de administrador o al menú de los usuarios normales.

El siguiente aspecto a desarrollar es el **registro**, a esta funcionalidad tendremos acceso a través de login, si un usuario no dispone de cuenta en nuestra aplicación, podrá crearse una a través del registro.

Una vez comprobado que el usuario no introduce campos vacíos o no válidos, procedemos a registrar el usuario, primero el usuario se registrará en la autenticación, aquí solo lo registraremos con email y contraseña, si ha salido bien, guardaremos el id del usuario en las SharedPreferences, crearemos un id para la foto que haya elegido (en el caso de que haya elegido, si no, se creará un id para la foto por defecto) y la insertaremos en el storage, una vez haya salido todo bien, insertaremos el usuario en la tabla usuarios de firestore, con los campos id, nombre, email, pass y foto.

Finalmente entraremos a la aplicación.

```

//PRIMERO NOS AUTENTICAMOS EN FIREBASE
FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, pass).
addOnCompleteListener{ it: Task<AuthResult>
    if(it.isSuccessful){//SI HA IDO BIEN
        id = Auth.currentUser.uid//recogemos el id del usuario
        //Lo metemos en SharedPreferences
        val pref = getSharedPreferences( name: "Preferencias", Context.MODE_PRIVATE).edit()
        pref.putString("idUsuario", id)
        pref.apply()

        val img = UUID.randomUUID().toString()//DAMOS UN NOMBRE A LA IMAGEN
        val ref = Storage.getReference( location: "/fotosUsuarios/$img")//DAMOS UNA URL A LA IMAGEN
        if(fotoUri == null){//cuando no hay foto, ponemos una por defecto
            //Cogemos una imagen por defecto
            var fotoDefecto = Uri.parse( uriString: "android.resource://com.example.bya/"+R.drawable.profile_user)
            ref.putFile(fotoDefecto).addOnSuccessListener { //Subimos la foto
                ref.downloadUrl.addOnSuccessListener { //descargamos su url
                    foto = it.toString() //lo asignamos a la variable
                }
            }
        }
        else{//cuando la hay la subimos

            ref.putFile(fotoUri!!).addOnSuccessListener { //Subimos la foto
                ref.downloadUrl.addOnSuccessListener { //descargamos su url
                    foto = it.toString() //lo asignamos a la variable
                }
            }
        }
        //UNA VEZ REGISTRADOS NOS LLEVA A LA ACTIVIDAD MAIN
        entrarMain()
    }
}

```

Como hemos mencionado anteriormente, el usuario puede elegir su foto de perfil, dependiendo de la opción que haya seleccionado (cámara o galería), se abrirá una u otra.

```

/**
 * En función de la constante recibida se abre la galería o la cámara
 * Los datos obtenidos los asignamos a la variable fotoUri
 * y gracias a Picasso lo metemos en la imagen
 */
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    Log.d( tag: "FOTO", msg: "Opción:---$requestCode")
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode == Activity.RESULT_CANCELED) {
        return
    }
    if (requestCode == GALERIA) {
        Log.d( tag: "FOTO", msg: "Entramos en Galería")
        if (data != null) {
            // Obtenemos su URI con su dirección temporal
            fotoUri = data.data
            try {
                Picasso.get().load(fotoUri).transform(CirculoTransformacion()).into(imgRegistro)
            } catch (e: IOException) {
                e.printStackTrace()
                Toast.makeText( context: this, text: "¡Fallo Galería!", Toast.LENGTH_SHORT).show()
            }
        }
    } else if (requestCode == CAMARA) {

        Picasso.get().
        load(fotoUri),
        transform(CirculoTransformacion()),
        into(imgRegistro)

    }
}

```

El siguiente aspecto a desarrollar será el **catálogo de los usuarios**, en esta funcionalidad, tendremos diferentes layouts y diferentes opciones para elegir el tipo de prenda que estamos buscando. Finalmente, cuando ya hemos decidido qué tipo de prenda buscamos, entraremos a un listado donde se mostrarán.

```
private fun llenarArrayPrendas(){

    //Limpiamos las listas
    listaPrendas.clear()
    listaPrendas2.clear()

    db.collection( collectionPath: "prendas")//Consultamos todas las prendas de un determinado tipo
        .whereEqualTo( field: "idTipo", tipo)
        // .orderBy("precio")
        .get()
        .addOnSuccessListener { result ->
            for (prenda in result) {
                //Recogemos los datos de la prenda
                val idPrenda = prenda.get("idPrenda").toString()
                val nombre = prenda.get("nombre").toString()
                val precio = prenda.get("precio").toString()
                val idTipo = prenda.get("idTipo").toString()
                val referencia = prenda.get("referencia").toString()
                val stock = prenda.get("stock").toString()
                val foto = prenda.get("foto").toString()

                //Añadimos la prenda a las listas
                val p = Prenda(idPrenda, idTipo, nombre, precio, foto, referencia, stock.toInt())
                listaPrendas.add(p)
                listaPrendas2.add(p)
            }

            listaPrendas.sortByDescending{ it.precio.toFloat() }
            listaPrendas2.sortByDescending{ it.precio.toFloat() }

            //Le indicamos el adaptador
            recy.adapter = prendasAdapter
        }
    }
}
```

Esta es la consulta de donde obtendremos la lista de prendas a mostrar, en ella, primero filtramos por tipo (solo se mostrará un tipo de prenda, ej. Si el usuario busca pantalones de hombre, solo se mostrarán pantalones de hombre) y después, ordenaremos por precio.

Una vez se nos muestre la lista con las prendas, tendremos la opción de ordenarlas por precio y filtrarlas por nombre.

```

/**
 * Cuando escribimos en la barra de búsqueda
 */
searchField.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
    override fun onQueryTextSubmit(query: String?): Boolean {
        return false
    }

    //Cuando cambia el texto crea una nueva lista con las prendas que coincidan con ese filtro
    override fun onQueryTextChange(newText: String?): Boolean {
        newText?.let { it: String ->
            if(newText.equals("")){
                prendasAdapter.filterByName(listaPrendas2)
            } else {
                val filteredList = listaPrendas.filter { it: Prenda
                    it.nombre.toLowerCase(Locale.getDefault()).contains(newText)
                }
                prendasAdapter.filterByName(filteredList)
            }
        }
        return false
    }
})

```

Este es el método que nos ayudará a filtrar por nombre, tenemos en el layout un searchField en el que podremos escribir el nombre de la prenda que queremos buscar, entonces se llamará a este método que según cambie el texto se filtrarán las prendas.

```

/**
 * Cuando seleccionamos una opción del spinner del filtro, ordena la lista de prendas por precio mayor o menor
 */
spiFiltro.setOnItemSelectedListener(object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(parentView: AdapterView<*>?, selectedItemView: View, position: Int, id: Long) {
        var item = spiFiltro.selectedItem.toString()

        if(item.equals("Precio: Mayor")){
            prendasAdapter.orderByPrecioAsc()
        }
        if (item.equals("Precio: Menor")) {
            prendasAdapter.orderByPrecioDes()
        }
    }

    override fun onNothingSelected(parentView: AdapterView<*>?) {
        // your code here
    }
})

```

Este es el método que nos ayudará a ordenar por precio, en el layout, tenemos un spinner donde podremos seleccionar el orden, pudiendo cambiar este a mayor o menor precio, cuando el spinner se modifique se llamará a este método que, según la opción seleccionada, así se ordenará.

Cuando pulsamos en un ítem de esta lista, nos llevará al detalle de la prenda, a este detalle, se le pasa el objeto prenda, y se mostrarán todos sus datos. Tendremos varios radio button donde podremos seleccionar la talla, y una vez seleccionada, podremos **añadir esta prenda a la cesta**.

```
/**  
 * Método que rellena los campos de una prenda  
 */  
private fun llenarCampos() {  
  
    // Consultamos los favoritos del usuario activo, para marcar los corazones  
    db.collection( collectionPath: "favoritos")  
        .whereEqualTo( field: "idUsuario", idUsuario)  
        .whereEqualTo( field: "idPrenda", p.idPrenda )  
        .get()  
        .addOnSuccessListener { result ->  
            for (fav in result) {  
                imgFav.setImageResource(R.drawable.twitter_like_rojo)  
                imgFav.setTag(R.drawable.twitter_like_rojo)  
            }  
        }  
  
    // Rellenamos los campos de la prenda  
    Picasso.get().load(Uri.parse(p.foto)).into(imgCamiseta)  
    tvNombre.setText(p.nombre)  
    tvPrecio.setText(p.precio + " EUR")  
  
    // Comprobamos si hay stock para indicarlo en el detalle  
    if(p.stock > 0){  
        tvStock.setTextColor(resources.getColor(R.color.verde))  
        tvStock.setText("(" + p.stock + ") en stock.")  
    } else {  
        tvStock.setTextColor(resources.getColor(R.color.rojo))  
        tvStock.setText("Sin stock.")  
        btnCesta.isEnabled = false  
        btnCesta.setBackgroundColor(resources.getColor(R.color.browser_actions_bg_grey))  
        Toast.makeText(requireContext(), text: "No hay stock", Toast.LENGTH_SHORT).show()  
    }  
}
```

Aquí podemos observar como gracias a el objeto prenda que le llega al detalle y a una consulta para saber si esa prenda está en sus favoritos o no, obtenemos toda la información de la prenda y la mostramos en los componentes del layout. La consulta de favoritos, filtramos por el usuario actual y la prenda del detalle para saber si ese usuario tiene en favoritos esa prenda, si esto es así el corazón se mostrará en rojo.

Al pulsar en el dibujo del corazón del layout, si está en rojo, quitaremos la prenda de favoritos, y si, por el contrario, está en blanco, la añadiremos.

```


    /**
     * Al pulsar en el corazon de favoritos
     */
    imgFav.setOnClickListener { it: View! }

    //Recogemos el tag de la imagen (corazon) de favoritos
    var idFoto = imgFav.tag
    var idFavorito = idUsuario + p.idPrenda//Creamos un id al favorito

    if(idFoto == R.drawable.twitter_like_rojo){//si idFoto esta en rojo
        imgFav.setImageResource(R.drawable.twitter_like)//cambiamos el icono
        imgFav.setTag(R.drawable.twitter_like)//cambiamos el tag
        db.collection( collectionPath: "favoritos").document(idFavorito).delete()//Eliminamos de la bbdd el favorito

    } else {
        imgFav.setAnimation(R.raw.black_joy)//Asignamos la animacion
        imgFav.playAnimation()//La reproducimos
        imgFav.setTag(R.drawable.twitter_like_rojo)//Cambiemos el tag

        //Añadimos el favorito a la bbdd
        val f = Favorito (idFavorito,idUsuario,p.idPrenda)
        db.collection( collectionPath: "favoritos").document(idFavorito).set(f)

    }
}


```

Al pulsar en la foto del corazón, recogemos el tag de la imagen y el id del favorito, que siempre va a ser el usuario + prenda, si el tag de la imagen, corresponde al tag de la imagen en rojo, cambiamos la imagen a blanca, le damos su nuevo tag, y borramos la foto de favoritos, si, por el contrario, el tag corresponde a la imagen en blanco, lanzamos una animación, le damos su nuevo tag y la añadimos a favoritos.

El siguiente aspecto a desarrollar es la lista de **favoritos**, en esta lista tendremos todas las prendas que el usuario tiene guardadas como favoritas, esta lista se mostrará sobre un recycler view que contará con un adaptador que enseñará la foto de la prenda en cada ítem.

```


private fun llenarArrayFavoritos() {

    listaFavoritos.clear()//Limpiamos la lista

    //Consultamos los favoritos del usuario activo
    db.collection( collectionPath: "favoritos")
        .whereEqualTo( field: "idUsuario", idUsuario)
        .addSnapshotListener{ snapshot, e->
            listaFavoritos.clear()//limpiamos la lista
            for (fav in snapshot!!) {

                val idPrenda = fav.get("idPrenda").toString()//Recogemos el id de la prenda

                //Consultamos los datos de la prenda
                db.collection( collectionPath: "prendas")
                    .whereEqualTo( field: "idPrenda", idPrenda)
                    .addSnapshotListener { snap, a>
                        for (prenda in snap!!){

                            //Recogemos los datos
                            val foto = prenda.get("foto").toString()
                            val idTipo = prenda.get("idTipo").toString()
                            val nombre = prenda.get("nombre").toString()
                            val precio = prenda.get("precio").toString()
                            val referencia = prenda.get("referencia").toString()
                            val stock = prenda.get("stock").toString().toInt()

                            //Añadimos la prendaa a la lista
                            val p = Prenda(idPrenda, idTipo, nombre, precio, foto, referencia, stock)
                            listaFavoritos.add(p)
                        }
                    }
                    //Lo asignamos al adapter
                    recy.adapter = favoritosAdapter
            }
        }
}


```

Esta será la consulta que usaremos para llenar la lista con todas las prendas favoritas del usuario, lista que mostrará posteriormente el recycler. En esta consulta, filtramos por el usuario que actualmente está usando la aplicación.

Cabe destacar que, si pulsamos en un ítem de la lista, al igual que si lo hacemos en el catálogo, nos llevará al detalle de esta prenda.

```
/**  
 * Se llama cuando hacemos clic en una prenda, para abrir su detalle  
 */  
private fun eventoClicFila(prenda: Prenda) {  
    abrirPrenda(prenda)  
}  
  
/**  
 * Método que nos lleva al fragment del detalle de la prenda  
 */  
private fun abrirPrenda(prenda: Prenda) {  
  
    val transaction = requireActivity().supportFragmentManager.beginTransaction()  
    transaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)  
    transaction.add(R.id.fragmentFavorites, CatalogoUsuarioDetalleFragment(prenda, tipo: 1))  
    transaction.addToBackStack(name: "favoritos")  
    transaction.commit()  
}
```

El siguiente aspecto a desarrollar será la **cesta**, en esta funcionalidad tendremos una lista que un recycler mostará, con las prendas previamente añadidas. La función principal de la cesta es que el usuario pueda comprar las prendas que el desee.

```
/**  
 * Método que rellena la lista de la cesta  
 */  
private fun llenarArrayCesta() {  
  
    //Consultamos la cesta del usuario activo  
    db.collection(collectionPath: "cesta")  
        .whereEqualTo(field: "idUsuario", idUsuario)  
        .addSnapshotListener{ snapshot, e->  
  
            listaCesta.clear()//Limpiamos la cesta  
            for (cesta in snapshot!!) {  
  
                //Recogemos los valores de la cesta  
                val idCesta = cesta.get("idCesta").toString()  
                val idPrenda = cesta.get("idPrenda").toString()  
                val talla = cesta.get("talla").toString()  
  
                //Añadimos la cesta a la lista  
                val c = Cesta(idCesta, idUsuario, idPrenda, talla)  
                listaCesta.add(c)  
            }  
  
            //Le indicamos el adaptador  
            recy.adapter = cestaAdapter  
        }  
}
```

Esta es la consulta que hay en la cesta para que se muestren todas las prendas de la cesta del usuario que está activo en ese momento.

En este layout tenemos un text view con el precio total de cesta que se quiere comprar.

```
/**  
 * Método que rellena el precio en función de los precios de las prendas que tengamos en la cesta  
 */  
private fun llenarPrecio(){  
  
    //Consultamos las prendas  
    db.collection( collectionPath: "prendas")  
        .addSnapshotListener{ snapshot, e->  
            precio = 0f  
            for (prenda in snapshot!!) {  
                //AQUÍ TENEMOS TODAS LAS PRENDAS DE LA BASE DE DATOS  
                for (p in 0..listaCesta.size - 1){  
  
                    //si la prenda esta en nuestra cesta  
                    if (listaCesta[p].idPrenda.equals(prenda.get("idPrenda"))){  
                        //Sumamos su precio al que ya teníamos  
                        precio = prenda.get("precio").toString().toFloat() + precio  
                    }  
  
                }  
                //PASAMOS EL NÚMERO A DOS DECIMALES  
                decimal()  
                tvPrecio.text = pre + " EUR"//LO ASIGNAMOS  
                pre = pre.replace( oldChar: ',', newChar: '.');//REEMPLAZAMOS LA COMA POR EL PUNTO  
                precioMostrar = pre.toDouble()  
            }  
        }  
    }  
}
```

Este text view lo rellenamos haciendo una consulta a la base de datos, en esta consulta filtramos por el id de las prendas que hay en la cesta y miramos su precio, se hace un sumatorio con todos los precios y esto es lo que muestra el text view.

En la lista, tenemos un botón para eliminar las prendas directamente en la cesta, cuando pulsamos en este botón ejecutamos un delete y borramos la prenda de la cesta.

```
/**  
 * Se llama cuando hacemos clic en el botón X  
 */  
private fun eventoClicFila(cesta: Cesta) {  
    borrarPrenda(cesta)  
}  
  
/**  
 * Método que borra una prenda de la cesta de la bbdd  
 */  
private fun borrarPrenda(cesta: Cesta) {  
  
    db.collection( collectionPath: "cesta").document(cesta.idCesta).delete()  
    llenarPrecio()//Volvemos a llenar el precio  
}
```

Cuando continuamos con la cesta, la siguiente pantalla será la de **seleccionar la ubicación**.

```
/**  
 * Obtiene la posición actual para pasarsela al mapa y que se cargue en nuestra posición  
 */  
private fun mirarPosi() {  
  
    val task = fusedLocationProviderClient.lastLocation  
  
    if (ActivityCompat.checkSelfPermission(requireContext(), android.Manifest.permission.ACCESS_FINE_LOCATION) !=  
        PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(requireContext(),  
            android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED){  
  
        ActivityCompat.requestPermissions(requireActivity(), arrayOf(android.Manifest.permission.ACCESS_FINE_LOCATION),  
            requestCodes)  
        return  
    }  
  
    task.addOnSuccessListener { it: Location! ->  
        if(it != null){  
            posicion = LatLng(  
                it.latitude,  
                it.longitude  
            )  
            mMap.moveCamera(CameraUpdateFactory.newLatLng(posicion));  
        }else{  
            Toast.makeText(requireContext(), text: "Error al obtener su ubicación, Refresce", Toast.LENGTH_LONG).show()  
        }  
    }  
}
```

En esta pantalla, por defecto se obtiene la ubicación actual del usuario, por si no selecciona ninguna. Puede pasar que a veces no encuentra la ubicación, para ello, hemos puesto un mensaje de advertencia en un toast. Si obtiene correctamente la ubicación, se guarda en la variable posición la latitud y la longitud obtenidas y mueve el mapa a esa ubicación.

```
/**  
 * Cuando pulsemos en mapa se nos crea un marcador  
 */  
private fun activarEventosMarcadores() {  
    mMap.setOnMapClickListener { point -> // Creamos el marcador  
        // Borramos el marcador Touch si está puesto  
        marcadorTouch?.remove()  
        marcadorTouch = mMap.addMarker(  
            MarkerOptions() // Posición  
                .position(point) // Titulo  
                .title( title: "Posición Actual") // título  
                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW))  
        )  
        mMap.moveCamera(CameraUpdateFactory.newLatLng(point))  
        posicion = point  
    }  
}
```

Si el usuario lo desea puede seleccionar otra ubicación, esto lo puede hacer pulsando en el mapa, donde se cargará automáticamente un marcador amarillo, con el título de posición actual, si el usuario ha utilizado esta opción, esta será la ubicación que se guardará en la variable posición.

Cuando le damos a siguiente, continuamos con el pago de la cesta, en esta pantalla, si hemos introducido los datos correctos de la tarjeta, podremos finalizar el pago de nuestras prendas.

```

//Si hemos rellenado toda la informacion correctamente
if(pedido){
    //Quitamos los errores
    etTarjeta.setError(null)
    etCsv.setError(null)

    //Recogemos la fecha actual y le damos un formato
    val fechaCompra = LocalDate.now()
    var fechaBDD = DateTimeFormatter.ofPattern(pattern: "dd/MM/yyyy").format(fechaCompra)

    //Recorremos la cesta y por cada prenda de la cesta, hacemos un pedido
    for (i in 0..listaCesta.size - 1){
        val idPedido = UUID.randomUUID().toString()//Le damos un id al pedido

        //Insertamos el pedido en la bbdd
        val p = Pedido(idPedido, listaCesta[i].idPrenda, idUsuario, fechaBDD.toString(), latitud.toString(),
            longitud.toString(), listaCesta[i].talla, estado: 0)
        db.collection(collectionPath: "pedidos").document(idPedido).set(p)
    }

    Toast.makeText(requireActivity(), text: "¡Compra realizada con éxito!", Toast.LENGTH_SHORT).show()

    borrarCesta()//Borramos las prendas de la cesta
    agradecimientos()//Nos vamos al fragment de agradecimientos

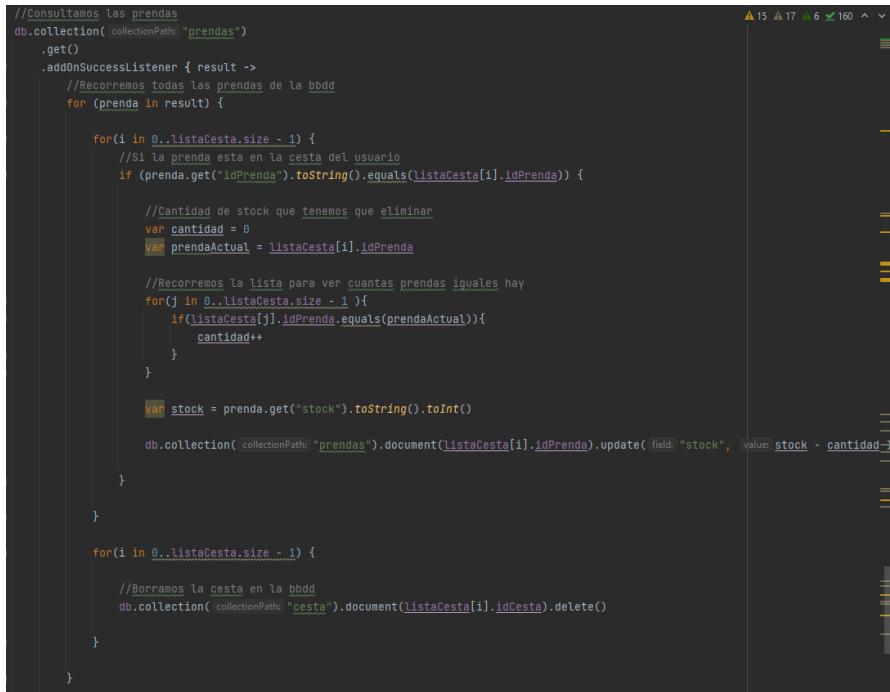
} else{
    Toast.makeText(requireActivity(), text: "¡Datos Incorrectos!", Toast.LENGTH_SHORT).show()
}

```

Aquí recogemos la fecha actual, y por cada ítem de la cesta, hacemos un insert de un pedido, introduciéndole los parámetros idPrenda, idUsuario, fecha, latitud, longitud, talla y estado, siendo estado 0. Estado puede tener 3 valores, 0 que significa que todavía no está enviado, 1 que significa que ya está enviado y 2 que significa que el pedido está ya devuelto.

Si todo ha salido bien, nos saldrá un mensaje avisándonos y ya habremos comprado las prendas.

Una vez hecho esto modificaremos el stock de las prendas compradas y borraremos la cesta.



```

//Consultamos las prendas
db.collection(collectionPath: "prendas")
    .get()
    .addOnSuccessListener { result ->
        //Recorremos todas las prendas de la bbdd
        for (prenda in result) {
            for(i in 0..listaCesta.size - 1) {
                //Si la prenda esta en la cesta del usuario
                if (prenda.get("idPrenda").toString().equals(listaCesta[i].idPrenda)) {

                    //Cantidad de stock que tenemos que eliminar
                    var cantidad = 0
                    var prendaActual = listaCesta[i].idPrenda

                    //Recorremos la lista para ver cuantas prendas iguales hay
                    for(j in 0..listaCesta.size - 1) {
                        if(listaCesta[j].idPrenda.equals(prendaActual)){
                            cantidad++
                        }
                    }

                    var stock = prenda.get("stock").toString().toInt()

                    db.collection(collectionPath: "prendas").document(listaCesta[i].idPrenda).update(field: "stock", value: stock - cantidad)
                }
            }
        }
    }

    for(i in 0..listaCesta.size - 1) {
        //Borramos la cesta en la bbdd
        db.collection(collectionPath: "cesta").document(listaCesta[i].idCesta).delete()
    }
}

```

Aquí tenemos la consulta donde recorremos la lista de las prendas que hemos comprado, y se la restaremos al stock actual de cada prenda. Una vez hemos modificado todo el stock, también hacemos otra consulta para borrar la cesta.

Una vez hecho esto ya hemos terminado de comprar la prenda.

El siguiente aspecto a desarrollar será el **historial**, aquí tendremos una lista que mostrará todos los pedidos que el usuario que está usando la aplicación ha realizado.

```
/**  
 * Método que rellena la lista del historial  
 */  
private fun llenarArrayHistorial() {  
  
    //Consultamos el historial del usuario activo  
    db.collection( collectionPath: "pedidos")  
        .whereEqualTo( field: "idUsuario", idUsuario)  
        .addSnapshotListener{ snapshot, e->  
            listaHistorial.clear()//limpiamos la lista  
            for (historial in snapshot!!) {  
  
                //Recogemos los datos de la lista  
                val idPedido = historial.get("idPedido").toString()  
                val idPrenda = historial.get("idPrenda").toString()  
                val idUsuario = historial.get("idUsuario").toString()  
                val fechaCompra = historial.get("fechaCompra").toString()  
                val latitud = historial.get("latitud").toString()  
                val longitud = historial.get("longitud").toString()  
                val talla = historial.get("talla").toString()  
                val estado = historial.get("estado").toString().toInt()  
  
                //Añadimos el pedido a la lista del historial  
                val p = Pedido(idPedido, idPrenda, idUsuario, fechaCompra, latitud, longitud, talla, estado)  
                listaHistorial.add(p)  
            }  
  
            //Lo asignamos al adaptador  
            recy.adapter = historialAdapter  
        }  
}
```

Esta es la consulta que tenemos, filtramos por el usuario que está usando la aplicación, una vez hecho esto recogemos todas las prendas que este usuario ha pedido, rescatamos todos sus datos, creamos el objeto, lo guardamos en la lista y llamamos al adaptador del recycler para que muestre la lista.

En esta lista, tendremos una funcionalidad extra, podremos devolver la prenda generando un código QR. Para esto, tenemos que pulsar en el ícono del QR de la lista. Esto hará que llamemos a un método que, pasándole un texto, en este caso le pasaremos el id del pedido, nos generará una imagen en bitmap que insertaremos en una dialogo.

```

private fun eventoClicFila(pedido: Pedido) {
    dialog.setContentView(R.layout.codigo_qr_layout)
    dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

    //Enlazamos con el diseño
    val imgQr: ImageView = dialog.findViewById(R.id.imgCodigoQRImagen)
    //Recogemos el idPedido
    var texto = pedido.idPedido
    //Generamos el código QR con el idPedido
    val bitmap = generateQRCode(texto)
    imgQr.setImageBitmap(bitmap)//La asignamos a la imagen

    dialog.show()// mostramos el dialog
}

/**
 * Método que genera una imagen con el código qr
 */
private fun generateQRCode(text: String): Bitmap {
    val width = 500
    val height = 500
    val bitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888)
    val codeWriter = MultiFormatWriter()
    try {
        val bitMatrix = codeWriter.encode(text, BarcodeFormat.QR_CODE, width, height)
        for (x in 0 until width) {
            for (y in 0 until height) {
                bitmap.setPixel(x, y, if (bitMatrix[x, y]) Color.BLACK else Color.WHITE)
            }
        }
    } catch (e: WriterException) {
        Log.d("QR", msg: "generateQRCode: ${e.message}")
    }
    return bitmap
}

```

Aquí tenemos los dos métodos de los que hemos hablado anteriormente.

El siguiente aspecto a desarrollar será el **contactar con un asistente**, al acceder a esta funcionalidad, nos aparecerá un chat para preguntarle cualquier duda que nos surja al asistente.

```

//Consultamos el usuario administrador
db.collection( collectionPath: "usuarios")
    .whereEqualTo( field: "tipo", value: 0)
    .get()
    .addOnSuccessListener { result ->
        for (usu in result) {

            //Rescatamos la información del usuario
            val idUsuario = usu.get("idUsuario").toString()
            val nombre = usu.get("nombre").toString()
            val foto = usu.get("foto").toString()

            //Recogemos el idUsuario del usuario activo con las SharedPreferences
            val pref = activity?.getSharedPreferences( name: "Preferencias", Context.MODE_PRIVATE)
            idUsuarioEmisor = pref?.getString( key: "idUsuario", defaultValue: "null").toString()

            //El usuario receptor, va a ser el usuario administrador
            idUsuarioReceptor = idUsuario

            //Asignamos la información en el layout
            Picasso.get().load(Uri.parse(foto)).transform(CirculoTransformacion()).into(imgChatFoto)
            tvChatNombre.text = nombre

            //Cargamos los mensajes
            leerMensaje(idUsuarioEmisor, idUsuarioReceptor)
        }
    }

```

En esta consulta, rescataremos el id, nombre y foto del usuario administrador, el nombre y la foto la pondremos arriba para mostrar que estamos teniendo una conversación con él.

Para que funcione el chat, necesitamos un emisor y un receptor, como emisor, cogeremos al usuario está activo, y como receptor, cogeremos el id del usuario administrador que hemos rescatado de la consulta. Rellenamos los componentes con los datos del administrador y llamamos al método de leer mensaje.

```
//Consultamos el chat y ordenamos por fecha
db.collection( collectionPath: "chat")
    .orderBy( field: "fecha")
    .addSnapshotListener{ snapshot, e->
        chatList.clear()//Limpiamos la lista

        for (chat in snapshot!!) {

            //Rescatamos los mensajes si hemos mantenido una conversación
            if(chat.get("idEmisor").toString().equals(idEmisor) && chat.get("idReceptor").toString().equals(
                idReceptor
            ) ||

                chat.get("idReceptor").toString().equals(idReceptor) && chat.get("idEmisor").toString().equals(
                    idEmisor
                )){

                //Recogemos los datos del chat
                val idEmisor = chat.get("idEmisor").toString()
                val idReceptor = chat.get("idReceptor").toString()
                val mensaje = chat.get("mensaje").toString()
                val fecha = chat.get("fecha").toString()

                val c = Chat(idEmisor, idReceptor, mensaje, fecha)
                //Añadimos ese objeto a la lista de chat
                chatlist.add(c)
            }
        }

        val chatAdapter = ChatListAdapter(requireContext(), chatList)
        //Le asignamos el adaptador
        recyclerView.adapter = chatAdapter
    }
}
```

En este método, recibimos el emisor y el receptor, consultamos todos los mensajes que hayan intercambiado entre ellos, los añadimos a una lista, lo ordenamos por fecha y los mostramos en el recycler, usando un adaptador especial, que explicaremos a continuación.

```
/*
 * Método que recibe, el idEmisor, idReceptor y el mensaje para poder guardarlo en la bbdd
 */
@RequiresApi(Build.VERSION_CODES.O)
private fun enviarMensaje(idEmisor: String, idReceptor: String, mensaje: String){
    //Actualizamos el valor de chat, para indicar que hemos hablado con el administrador
    db.collection( collectionPath: "usuarios").document(idEmisor).update( field: "chat", value: 1)

    //Creamos un hashMap con los valores del mensaje
    var hashMap : HashMap<String, String> = HashMap()
    hashMap.put("idEmisor", idEmisor)
    hashMap.put("idReceptor", idReceptor)
    hashMap.put("mensaje", mensaje)
    hashMap.put("fecha", LocalDateTime.now().toString())

    //Añadimos el chat(mensaje,idEmisor,idReceptor,fecha) a la bbdd
    db.collection( collectionPath: "chat").add(hashMap)
}
```

Cuando rellenamos el edit text de enviar mensaje, se llama a este método, donde creamos un hashMap con los campos y los valores que vamos a introducir en la base de datos, se introduce el idEmisor, el idReceptor, el mensaje y la fecha, y se inserta el mensaje en la base de datos.

Cabe destacar que cuando se envía un mensaje, la variable chat del usuario se actualiza a 1 para que este usuario aparezca en la lista de personas que le han hablado al administrador.

```
/*
 * Recogemos el tipo de vista
 */
override fun getItemViewType(position: Int): Int {

    firebaseUser = FirebaseAuth.getInstance().currentUser//Recogemos el usuario autenticado activo

    if(listaChat[position].idEmisor == firebaseUser!!.uid){//Si el usuario actual es el emisor
        return MENSAJE_DERECHA//Los mensajes van a la derecha
    } else {
        return MENSAJE_IZQUIERDA//Si no, van a la izquierda
    }

}
```

Este código es lo que tiene de especial el adaptador, ya que dependiendo de quien manda el mensaje, este tendrá un tipo de vista, si es el emisor el que ha mandado el mensaje, este llamará a un tipo de layout donde pondrá el mensaje a la derecha, siendo al revés cuando es el receptor el que manda el mensaje.

El siguiente aspecto a desarrollar será el **perfil**, en esta funcionalidad el usuario, tanto el administrador, como el usuario normal, podrán editar su perfil de usuario (menos el email o si eres un usuario que viene de Google).

Cuando entramos en esta pantalla, se hace una consulta donde se cargan los datos en los edit text.

```
/*
 * Metodo que carga los datos del usuario activo
 */
private fun cargarDatos() {

    //Consultamos los datos del usuario
    db.collection( collectionPath: "usuarios").document(idUsuario).get().addOnSuccessListener {
        //Rescatamos los datos del usuario
        name = it.get("nombre").toString()
        email = it.get("email").toString()
        photoUrl = it.get("foto").toString()
        pass = it.get("pass").toString()

        //Asignamos la información en el layout
        etNombre.setText(name)
        etEmail.setText(email)
        etPass.setText(pass)

        Picasso.get()
            .load(Uri.parse(photoUrl))
            .transform(CirculoTransformacion())
            .resize( targetWidth: 178, targetHeight: 178)
            .into(imgPerfil)

        //Deshabilitamos el email, ya que no se va a poder cambiar
        etEmail.isEnabled = false
        etEmail.setBackgroundColor(resources.getColor(R.color.dark))

    }
}
```

En esta consulta, rescatamos el usuario que está usando la aplicación y obtenemos sus datos para llenar los edit text con ellos, también deshabilitamos el edit text del email y le ponemos un tono de color un poco más oscuro para que el usuario no lo pueda cambiar.

```
//ACTUALIZAMOS LA CONTRASEÑA EN LA TABLA DE AUTENTICACION (SI LA CAMBIA)
user!!.updatePassword(pass)

if (fotoUri == null) {//Si el usuario no ha elegido foto

    //actualizamos solo los campos nombre y pass
    db.collection( collectionPath: "usuarios").document(idUsuario).update( field: "nombre", nombre)
    db.collection( collectionPath: "usuarios").document(idUsuario).update( field: "pass", pass)

    Toast.makeText(requireContext(), text: "Usuario actualizado",Toast.LENGTH_SHORT).show()

} else {//Si ha elegido foto
    val filename = UUID.randomUUID().toString()//Le creamos un id
    val ref = firebaseStorage.getInstance().getReference( location: "/fotosUsuarios/$filename")//Creamos una url
    ref.putFile(fotoUri!!).addOnSuccessListener { //Metemos la imagen en el storage
        ref.downloadUrl.addOnSuccessListener { //Descargamos la url de la imagen

            photoUrl = it.toString()//Rescatamos la url de la imagen

            //Actualizamos todos los campos, incluida la foto
            db.collection( collectionPath: "usuarios").document(idUsuario).update( field: "nombre", nombre)
            db.collection( collectionPath: "usuarios").document(idUsuario).update( field: "pass", pass)
            db.collection( collectionPath: "usuarios").document(idUsuario).update( field: "foto", photoUrl)

        }
    }

    Toast.makeText(requireContext(), text: "Usuario actualizado",Toast.LENGTH_SHORT).show()
}
```

Cuando le damos a guardar cambios, primero revisamos que los campos que haya introducido sean válidos, después actualizamos la contraseña en la autenticación, y hacemos un update en firestore, actualizando los campos nombre y pass si no hemos cambiado la foto y nombre, pass y foto si hemos cambiado la foto.

El siguiente aspecto a desarrollar será el **catálogo del administrador**, en esta funcionalidad, el usuario administrador tendrá acceso a una lista de todas las prendas, donde podrá añadir/editar/borrar estas prendas.

```
private fun llenarArrayPrendas(){

    listaPrendas.clear()//Limpiamos la lista

    db.collection( collectionPath: "prendas")//Consultamos todas las prendas
        .get()
        .addOnSuccessListener { result ->
            for (prenda in result) {
                //Recogemos los datos de la prenda
                val idPrenda = prenda.get("idPrenda").toString()
                val nombre = prenda.get("nombre").toString()
                val precio = prenda.get("precio").toString()
                val idTipo = prenda.get("idTipo").toString()
                val referencia = prenda.get("referencia").toString()
                val stock = prenda.get("stock").toString()
                val foto = prenda.get("foto").toString()

                //Añadimos la prenda a la lista
                val p = Prenda(idPrenda, idTipo, nombre, precio, foto, referencia, stock.toInt())
                listaPrendas.add(p)
            }
        }

        //Le damos valor a prendasAdapter
        prendasAdapter = CatalogoListAdapter(listaPrendas)

        //Le indicamos el adaptador
        recyclerView.adapter = prendasAdapter
    }
}
```

Esta será la consulta con la cual obtendremos una lista de todas las prendas que hay en la aplicación, esta será la lista que se pasará al recycler para que se muestre.

Si en esta lista deslizamos el ítem a la derecha accederemos a la pantalla de editar la prenda, sin embargo, si deslizamos a la izquierda nos saldrá un diálogo preguntándonos si queremos borrar la prenda.

```
private fun iniciarSwipeHorizontal() {
    val simpleItemTouchCallback: ItemTouchHelper.SimpleCallback = object : ItemTouchHelper.SimpleCallback(
        dragDirs: 0, swipeDirs: ItemTouchHelper.LEFT or
        ItemTouchHelper.RIGHT
    ) {

        override fun onMove(
            recyclerView: RecyclerView,
            viewHolder: RecyclerView.ViewHolder,
            target: RecyclerView.ViewHolder
        ): Boolean {
            return false
        }

        //Según donde deslizemos
        override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
            val position = viewHolder.adapterPosition

            //izquierda -> borramos
            //derecha -> editamos
            when (direction) {
                ItemTouchHelper.LEFT -> {
                    borrarPrenda(position)
                }
                else -> {
                    editarPrenda(position)
                }
            }
        }
    }
}
```

Este es el método que inicia el deslizamiento horizontal, donde le decimos que si hemos deslizado el ítem a la derecha nos lleve al método de editar la prenda y si lo hemos hecho a la izquierda que nos lleve al método de borrarla.

```
/*
 * Se crea el dibujo cuando deslizamos
 */
override fun onChildDraw(
    canvas: Canvas,
    recyclerView: RecyclerView,
    viewHolder: RecyclerView.ViewHolder,
    dX: Float,
    dY: Float,
    actionState: Int,
    isActive: Boolean
) {
    if (actionState == ItemTouchHelper.ACTION_STATE_SWIPE) {
        val itemView = viewHolder.itemView
        val height = itemView.bottom.toFloat() - itemView.top.toFloat()
        val width = height / 3

        if (dX > 0) { //Creamos el botón de borrar
            botonIzquierdo(canvas, dX, itemView, width)
        } else { //Creamos el botón de editar
            botonDerecho(canvas, dX, itemView, width)
        }
    }
    super.onChildDraw(canvas, recyclerView, viewHolder, dX, dY, actionState, isActive)
}
```

Tenemos también un método donde si deslizamos a la derecha, se nos dibuja un botón que tendrá una imagen de un lápiz que hace referencia a editar, lo mismo para la izquierda donde nos saldrá una cruz.

```
/**  
 * Método que elimina una prenda del recycler y de la bbdd  
 */  
private fun borrarPrenda(position: Int) {  
    //Cuando hemos deslizado, quitamos el elemento del swipe y lo ponemos  
    //instantáneamente para que desaparezca el color del fondo  
    val deleteModel: Prenda = listaPrendas[position]  
    prendasAdapter.removeItem(position)  
    prendasAdapter.restoreItem(deleteModel, position)  
  
    //Alert dialog para confirmar si desea eliminar la prenda deslizada  
    Log.i("tag: Eliminar", "msg: Eliminando...")  
    AlertDialog.Builder(requireContext())  
        .setIcon(R.mipmap.ic_launcher_bya_round)  
        .setTitle("Eliminar prenda")  
        .setMessage("¿Desea eliminar la prenda existente?")  
        .setPositiveButton(text: "Sí"){ dialog, which -> eliminarPrendaConfirmada(position)}  
        .setNegativeButton(text: "No", listener: null)  
        .show()  
}  
  
/**  
 * Si en el alert dialog hemos confirmado que sí queremos eliminar la prenda  
 * la borramos de la base de datos  
 */  
private fun eliminarPrendaConfirmada(position: Int) {  
  
    borrarFavoritosPrendaCesta(listaPrendas[position])  
    val snackbar = Snackbar.make(requireView(), text: "Prenda eliminada con éxito", Snackbar.LENGTH_LONG)  
    prendasAdapter.removeItem(position)  
    snackbar.show()  
}  
  
/**  
 * Método que borra un favorito de la bbdd  
 */  
private fun borrarFavoritosPrendaCesta(p: Prenda){  
  
    //Hacemos una consulta para borrar todos los favoritos que tengan ese idPrenda  
    db.collection( collectionPath: "favoritos")  
        .whereEqualTo( field: "idPrenda", p.idPrenda)  
        .get()  
        .addOnSuccessListener { result ->  
            for (fav in result) {  
  
                val idFavorito = fav.get("idFavorito").toString()  
  
                db.collection( collectionPath: "favoritos").document(idFavorito).delete()  
            }  
        }  
  
    //Hacemos una consulta para borrar todos las prendas de la cesta que tengan ese idPrenda  
    db.collection( collectionPath: "cesta")  
        .whereEqualTo( field: "idPrenda", p.idPrenda)  
        .get()  
        .addOnSuccessListener { result ->  
            for (cesta in result) {  
  
                val idCesta = cesta.get("idCesta").toString()  
  
                db.collection( collectionPath: "cesta").document(idCesta).delete()  
            }  
        }  
  
    //Borramos la prenda  
    db.collection( collectionPath: "prendas").document(p.idPrenda).delete()  
}
```

Cuando se llama al método de borrar, lo primero que hace es saltarnos un alert dialog donde nos pide confirmación para borrar la prenda, si pulsamos en “no”, no pasa nada, se cierra el alert dialog y se vuelve a la lista de prenda. Si pulsamos en “sí” abriremos otro método donde eliminaremos el ítem de la lista y llamaremos a otro método al que le pasaremos la prenda que queremos eliminar. En este último método haremos tres consultas.

En la primera consulta, eliminaremos la prenda de todos los favoritos de todos los usuarios, ya que no existe la prenda nadie la puede tener en favoritos.

En la siguiente consulta, eliminaremos la prenda de todas las cestas de todos los usuarios.

En la última consulta, eliminaremos la prenda.

```
/**  
 * Método para editar una prenda  
 */  
private fun editarPrenda(position: Int) {  
  
    //Ocultamos el floating button  
    anadirPrenda.hide()  
  
    //Cuando hemos deslizado, quitamos el elemento del swipe y lo ponemos  
    //instantáneamente para que desaparezca el color del fondo  
    val editedModel: Prenda = listaPrendas[position]  
    prendasAdapter.removeItem(position)  
    prendasAdapter.restoreItem(editedModel, position)  
  
    //llamamos al fragment editar  
    editar(position)  
  
}
```

Cuando seleccionamos la opción de editar prenda ocultamos el botón flotante y llamamos al fragment de editar.

```
/**  
 * Método que carga la información de la prenda  
 */  
private fun cargarDatos() {  
  
    //Consultamos el tipo que tiene la prenda  
    db.collection(collectionPath: "tipo").document(p.idTipo).get().addOnSuccessListener { it: DocumentSnapshot! ->  
  
        etNombre.setText(p.nombre)  
        etPrecio.setText(p.precio)  
        etReferencia.setText(p.referencia)  
        etTipo.setText(it.get("descripcion").toString())  
  
        Picasso.get().load(Uri.parse(p.foto)).into(imgPrenda)  
  
        //La referencia no se puede editar  
        etReferencia.isEnabled = false  
        etReferencia.setBackgroundColor(resources.getColor(R.color.dark))  
  
        //El tipo no se puede editar  
        etTipo.isEnabled = false  
        etTipo.setBackgroundColor(resources.getColor(R.color.dark))  
  
    }  
  
}
```

En este fragment podremos editar todos los campos de la prenda (menos el tipo y la referencia), nada más entrar al fragment cargamos los campos, aprovechando que le pasamos la prenda que queremos editar, rescatamos sus valores y los metemos en sus componentes.

También deshabilitamos los edit text de referencia y tipo para que no se puedan tocar.

```
if (fotoUri == null) {//SI EL USUARIO NO HA ELEGIDO FOTO DE LA PRENDAS

    //Se actualizan todos los campos menos la imagen
    db.collection( collectionPath: "prendas").document(p.idPrenda).update( field: "nombre", nombre)
    db.collection( collectionPath: "prendas").document(p.idPrenda).update( field: "precio", precio)

    Toast.makeText(requireContext(), text: "Prenda actualizada", Toast.LENGTH_SHORT).show()

} else {//SI HA CAMBIADO LA FOTO DE LA PRENDAS

    val filename = UUID.randomUUID().toString()//Le damos un id a la imagen
    val ref = FirebaseStorage.getInstance().getReference( location: "/fotosPrendas/$filename")//Le damos una ruta
    ref.putFile(fotoUri!!).addOnSuccessListener {//Metemos la imagen en el storage
        ref.downloadUrl.addOnSuccessListener {//Descargamos la imagen

            photoUrl = it.toString()//Recogemos la url, para introducirla en el campo foto

            //Actualizamos todos los campos de la prenda, incluida la foto
            db.collection( collectionPath: "prendas").document(p.idPrenda).update( field: "nombre", nombre)
            db.collection( collectionPath: "prendas").document(p.idPrenda).update( field: "precio", precio)
            db.collection( collectionPath: "prendas").document(p.idPrenda).update( field: "foto", photoUrl)

            Toast.makeText(requireContext(), text: "Prenda actualizada", Toast.LENGTH_SHORT).show()

        }
    }
}
```

Cuando pulsamos en editar la prenda, primero, comprobamos que todos los datos introducidos no son nulos y son válidos, y después, si todo está correcto hacemos un insert en la base de datos para actualizar a los nuevos valores.

En la lista de prendas abajo a la derecha tendremos un botón flotante donde podremos añadir prendas a esa lista. Al pulsar en el botón nos aparecerá un diálogo con dos opciones, nos preguntará si la prenda que vamos a insertar existe o no.

```
var dialog = Dialog(requireActivity())//Instanciamos un dialogo

//Al pulsar en añadir prenda, se abrirá un dialogo
anadirPrenda.setOnClickListener { it: View!

    //Abrimos un dialog con las 2 opciones (prenda existente, prenda nueva)
    dialog.setContentView(R.layout.anadir_prenda_existente_layout)
    dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

    //Se rescatan los datos del layout
    var imgExistente: ImageView = dialog.findViewById(R.id.imgPrendaExistente)
    var imgNueva: ImageView = dialog.findViewById(R.id.imgPrendaNueva)
    var tvExistente: TextView = dialog.findViewById(R.id.tvPrendaExistente)
    var tvNueva: TextView = dialog.findViewById(R.id.tvPrendaNueva)

    /**
     * botón para añadir una prenda ya existente en la bbdd
     */
    imgExistente.setOnClickListener(){ it: View!
        entrarAnadirPrendaExistente()
        dialog.dismiss()
    }

    /**
     * botón para añadir una nueva prenda en la bbdd
     */
    imgNueva.setOnClickListener(){ it: View!
        entrarAnadirPrenda()
        dialog.dismiss()
    }

}
```

Cuando pulsamos en la opción de añadir prenda nueva, nos llevará al fragmente de añadir prenda.

```
//DAMOS UN ID A LA PRENDA
_idPrenda = UUID.randomUUID().toString()

//En función de la prenda seleccionada, le damos el idTipo de la bbdd
when (spiprencia.selectedItem.toString()) {

    "Camiseta M" -> idTipo = "0"
    "Blusa M" -> idTipo = "1"
    "Vestido M" -> idTipo = "2"
    "Jeans M" -> idTipo = "3"
    "Accesorio M" -> idTipo = "4"
    "Falda M" -> idTipo = "5"
    "Camiseta H" -> idTipo = "6"
    "Camisa H" -> idTipo = "7"
    "Accesorio H" -> idTipo = "8"
    "Jeans H" -> idTipo = "9"

    else -> idTipo = "No encontrado"
}

var existe: Boolean = false

//recogemos el valor introducido en el layout
nombre = etNombre.text.toString()
precio = etPrecio.text.toString()
referencia = etReferencia.text.toString()
```

En este fragment tendremos varios edit text donde podremos rellenar con los datos de la nueva prenda, también tendremos un spinner que se llenará con los tipos de prenda existentes.

Al recoger los datos introducidos, cuando recogemos el tipo, dependiendo de que opción del spinner haya elegido, cogeremos un tipo numérico de prenda u otro.

También crearemos un id a la prenda y recogeremos el nombre, precio, referencia y foto introducidos.

```
//Hacemos una consulta para comprobar que no se introduzca una referencia existente
db.collection( collectionPath: "prendas")
    .get()
    .addOnSuccessListener { result ->
        for (prenda in result) {

            if (prenda.get("referencia").toString() == referencia) {

                tilAnadirPrendaReferencia.setError("La referencia ya existe")

                existe = true //Indicamos que la referencia existe
            }
        }

        if (!existe) {//Si la referencia no existe, insertamos la prenda

            tilAnadirPrendaReferencia.setError(null)
            val img = UUID.randomUUID().toString() //DAMOS UN NOMBRE A LA IMAGEN
            val ref = Storage.getReference( location: "/fotosPrendas/$img") //Le damos una ruta a la imagen

            ref.putFile(fotoUri!!).addOnSuccessListener { //Subimos la foto
                ref.downloadUrl.addOnSuccessListener { //descargamos su url
                    foto = it.toString() //lo asignamos a la variable
                }

                //Finalmente subimos la prenda
                val p = Prenda(idPrenda, idTipo, nombre, precio, foto, referencia, stock: 1)
                db.collection( collectionPath: "prendas").document(idPrenda).set(p)
            }
        }
    }

    Toast.makeText(requireContext(), text: "Prenda insertada correctamente", Toast.LENGTH_SHORT).show()
```

Una vez recogidos los campos, comprobamos que estos no son nulos y son válidos, comprobando también con una consulta que la referencia introducida no es repetida.

Si todo ha ido bien, le damos un id a la foto, la insertamos en storage y hacemos un insert en la base de datos con los campos de idPrenda, idTipo, nombre, precio, foto, referencia y stock que inicialmente será 1.

```
Toast.makeText(getApplicationContext(), "Prenda insertada correctamente", Toast.LENGTH_SHORT).show()

//Se borran los datos, para poder introducir una nueva prenda
etNombre.setText("")
etPrecio.setText("")
etReferencia.setText("")
fotoUri = null

//Ponemos la imagen por defecto de la camara
imgPrenda.setImageResource(R.drawable.ic_menu_camera)

}

}
```

Una vez hemos terminado de insertar se vacían todos los campos y se pone la imagen por defecto.

Cuando pulsamos en la opción de añadir prenda existente, nos llevará al fragmento de añadir prenda existente.

```
/**
 * Método que rellena el spinner con todas las referencias de las prendas
 */
private fun llenarSpinnerReferencia() {

    //Consultamos todas las referencias de las prendas
    db.collection("prendas")
        .get()
        .addOnSuccessListener { result ->
            for (prenda in result) {

                var ref = prenda.get("referencia").toString().toInt()
                //Las añadimos a la lista
                listaReferencias.add(ref)
            }

            //Las ordenamos, para que sea más fácil su elección
            listaReferencias.sort()
            //Las añadimos en el spinner
            spiReferencia.setAdapter(
                ArrayAdapter<Int>(
                    requireActivity(),
                    android.R.layout.simple_spinner_dropdown_item,
                    listaReferencias
                )
            )
        }
}
```

Cuando entramos en esta clase, lo primero que hace es llenar el spinner de referencias con todas las referencias existentes, esto lo hacemos para que no se pueda introducir una referencia nula.

A través de una consulta a la tabla prendas rellenamos una lista con todas las referencias existentes y las ordenamos de menor a mayor, una vez hecho esto, llamamos al adaptador del spinner.

```
//Al cambiar en el spinner la referencia se llama a este metodo
spiReferencia.setOnItemSelectedListener(object : OnItemSelectedListener {
    override fun onItemSelected(parentView: AdapterView<*>?, selectedItemView: View, position: Int, id: Long) {
        var item = spiReferencia.selectedItem.toString()//recogemos la referencia seleccionada

        //Hacemos una consulta de esa prenda para rescatar sus datos
        db.collection( collectionPath: "prendas")
            .get()
            .addOnSuccessListener { result ->
                for (prenda in result) {
                    if(prenda.get("referencia").toString().equals(item)){
                        var foto = prenda.get("foto").toString()
                        var nombre = prenda.get("nombre").toString()
                        var precio = prenda.get("precio").toString()

                        //Y los asignamos al layout
                        Picasso.get().load(Uri.parse(foto)).into(imaFoto)
                        etNombre.setText(nombre)
                        etPrecio.setText(precio + " EUR")
                    }
                }
            }
    }

    override fun onNothingSelected(parentView: AdapterView<*>?) {
        // your code here
    }
})
```

Cuando este spinner cambia de valor, automáticamente se hace una consulta a la tabla prendas, en esta consulta obtenemos el nombre, precio y foto de esa prenda con la referencia indicada por el usuario en el spinner y cambiamos los valores de los edit text y la image view.

```
//Al pulsar el botón guardar
btnGuardar.setOnClickListener { v: View! ->
    var item = spiReferencia.selectedItem.toString()//recogemos la referencia seleccionada

    //hacemos una consulta de la prenda con esa referencia
    db.collection( collectionPath: "prendas")
        .get()
        .addOnSuccessListener { result ->
            for (prenda in result) {
                if(prenda.get("referencia").toString().equals(item)){

                    //recogemos el id de la prenda y el stock de la misma
                    var idPrendaActual = prenda.get("idPrenda").toString()
                    var stock = prenda.get("stock").toString().toInt()

                    //Actualizamos el stock ++ de esa misma prenda
                    db.collection( collectionPath: "prendas").document(idPrendaActual).update( fields: "stock", value: stock + 1)
                }
            }
        }

        Toast.makeText(requireContext(), text: "Prenda insertada correctamente", Toast.LENGTH_SHORT).show()
    }
}
```

Una vez tenemos la prenda que queremos añadir, pulsamos el botón de añadir prenda y automáticamente, hacemos una consulta a la tabla prendas y filtramos por la referencia seleccionada, una vez tengamos acceso a esa prenda aumentamos su stock en 1.

El siguiente aspecto a desarrollar será el **gestionar pedidos**, en esta clase habrá una lista con todos los pedidos que han realizado todos los usuarios, pulsando en un ítem de la lista, podremos acceder al detalle para enviar el pedido.

```
/**  
 * Método que rellena la lista de pedidos  
 */  
private fun llenarArrayPedidos() {  
  
    //Consultamos todos los pedidos  
    db.collection( collectionPath: "pedidos")  
        .addSnapshotListener{ snapshot, e->  
            listaPedidos.clear()//limpiamos la lista  
  
            for (historial in snapshot!!) {  
  
                //Recogemos la información del pedido  
                val idPedido = historial.get("idPedido").toString()  
                val idPrenda = historial.get("idPrenda").toString()  
                val idUsuario = historial.get("idUsuario").toString()  
                val fechaCompra = historial.get("fechaCompra").toString()  
                val latitud = historial.get("latitud").toString()  
                val longitud = historial.get("longitud").toString()  
                val talla = historial.get("talla").toString()  
                val estado = historial.get("estado").toString().toInt()  
  
                //Añadimos el pedido a la lista  
                val p = Pedido(idPedido, idPrenda, idUsuario, fechaCompra, latitud, longitud, talla, estado)  
                listaPedidos.add(p)  
            }  
  
            recy.adapter = pedidosAdapter  
        }  
}
```

Esta será la consulta con la que rellenaremos la lista que cargará el recycler, en ella cogeremos todos los pedidos de todos los usuarios, obtendremos sus datos, crearemos un objeto pedido y los añadiremos a una lista.

Pulsando en un ítem de la lista, se nos abrirá el detalle del pedido, en este layout tendremos diferente información sobre el pedido y el usuario que lo ha realizado, además tendremos un mapa donde gracias a la latitud y longitud guardadas en el pedido de la base de datos, rescataremos la ubicación exacta y la mostraremos con una chincheta en el mapa.

Cabe destacar que dependiendo de la disponibilidad del usuario el botón estará habilitado o deshabilitado ya que no se puede enviar el pedido varias veces.

```

    /**
     * Obtiene la posición del pedido
     */
    private fun obtenerPosicion() {

        posicion = LatLng(
            p.latitud.toDouble(),
            p.longitud.toDouble()
        )
        mMap.moveCamera(CameraUpdateFactory.newLatLng(posicion));

        mMap.addMarker(
            MarkerOptions() // Posición
                .position(posicion) // Titulo
                .title( title: "Posición Pedido") // título
                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_YELLOW))
        )
    }
}

```

Este será el método que pintará la chincheta en la latitud y longitud rescatadas de la base de datos.

```

    /**
     * Cargamos los campos del pedido
     */
    private fun cargarCampos() {

        //Si el estado del pedido es 1 o 2
        if (p.estado == 1 || p.estado == 2){
            //Deshabilitamos el botón de enviar
            btnEnviar.isEnabled = false
            btnEnviar.setBackgroundColor(resources.getColor(R.color.browser_actions_bg_grey))
        }

        //Consultamos la prenda del pedido
        db.collection( collectionPath: "prendas")
            .whereEqualTo( field: "idPrenda", p.idPrenda)
            .get()
            .addOnSuccessListener { result ->
                for (prenda in result) {

                    //Recogemos sus datos
                    val nombre = prenda.get("nombre").toString()
                    val referencia = prenda.get("referencia").toString()
                    val foto = prenda.get("foto").toString()

                    //Los mostramos en el layout
                    tvIdCliente.text = p.idUsuario
                    tvReferencia.text = referencia
                    tvNombre.text = nombre
                    tvTalla.text = p.talla
                    Picasso.get().load(Uri.parse(foto)).into(imgPedido)

                }
            }
    }
}

```

Dependiendo del estado de pedido, deshabilitaremos el botón de envío, además, haremos una consulta a la tabla prendas y rescataremos los datos de esta y los mostraremos en los componentes.

```

/**
 * Al pulsar en el botón enviar, cambiamos el estado del pedido
 */
btnEnviar.setOnClickListener { it: View? ->
    //Actualizamos el estado del pedido a 1 (enviado)
    db.collection( collectionPath: "pedidos" ).document(p.idPedido).update( field: "estado", value: 1 )
    btnEnviar.isEnabled = false//Deshabilitamos el botón
    btnEnviar.setBackgroundColor(resources.getColor(R.color.browser_actions_bg_grey))
    Toast.makeText(requireContext(), text: "Pedido enviado", Toast.LENGTH_SHORT).show()
}

```

Al pulsar el botón enviar, actualizamos el estado del pedido a 1, deshabilitamos el botón para que no se le pueda dar dos veces y mostramos un mensaje de texto informativo.

El siguiente aspecto a desarrollar será el **devolver pedidos**, en esta funcionalidad tendremos una imagen de un QR, que, pulsando en ella, nos llevará a la cámara donde podremos escanear un código QR de un pedido, y dependiendo del estado del pedido escaneado, este se podrá devolver o no se será posible su devolución.

```

if (result != null) {//Si es null, ha habido un error al escanear
    if (result.contents == null){
        Toast.makeText(context, text: "Error", Toast.LENGTH_LONG).show()
    }
    else {
        //Si ha ido bien, abrimos un dialogo
        dialog.setContentView(R.layout.resultado_devolver_layout)
        dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

        //Enlazamos con el diseño
        val imgQr: LottieAnimationView = dialog.findViewById(R.id.imgResultadoDevolver)
        val tvQr : TextView = dialog.findViewById(R.id.tvResultadoDevolver)

        var estado = 0
        var fecha = ""
        var idPedido = ""

        //Consultamos el pedido a devolver
        db.collection( collectionPath: "pedidos" )
            .whereEqualTo( field: "idPedido", result.contents )
            .get()
            .addOnSuccessListener { result ->
                for (prenda in result) {
                    //Recogemos su información
                    fecha = prenda.get("fechaCompra").toString()
                    estado = prenda.get("estado").toString().toInt()
                    idPedido = prenda.get("idPedido").toString()

                }
            }
    }
}

```

```

//Le damos un formato a la fecha del pedido
val sdf = SimpleDateFormat(pattern: "dd/MM/yyyy")
var fechaComprado = Date(sdf.parse(fecha).getTime())
var dia = fechaComprado.date
//Obtenemos la fecha limite de devolucion del pedido
fechaComprado.date = dia + 15

//Obtenemos la fecha actual
val fechaHoy = sdf.format(Date())
val fechaCompradoMeter = sdf.format(fechaComprado)

if(estado == 1){ //Si el estado del pedido es 1
    if (fechaHoy.toString() <= fechaCompradoMeter.toString()){ //Comprobamos que no ha pasado el limite
        imgQr.setAnimation(R.raw.tick)//Le damos la animacion
        imgQr.playAnimation()//La ejecutamos
        tvQr.text = "La prenda se ha devuelto correctamente"
        //Actualizamos el estado a 2 (pedido devuelto)
        db.collection(collectionPath: "pedidos").document(idPedido).update(field: "estado", value: 2)
    } else { //Si ha pasado el limite, indicamos que no se puede devolver
        imgQr.setAnimation(R.raw.cross)//Le damos la animacion
        imgQr.playAnimation()//La ejecutamos
        tvQr.text = "La prenda no se puede devolver, han pasado 15 dias desde su compra"
    }
} else if (estado == 2){ //Si el estado es 2, no se puede devolver, porque ya esta devuelta
    imgQr.setAnimation(R.raw.cross)//Le damos la animacion
    imgQr.playAnimation()//La ejecutamos
    tvQr.text = "La prenda no se puede devolver, ya ha sido devuelta"
} else { //Si no, no se puede devolver porque no se ha realizado su entrega
    imgQr.setAnimation(R.raw.cross)
    imgQr.playAnimation()
    tvQr.text = "La prenda no se puede devolver, todavia no se ha realizado su entrega"
}

```

Cuando escaneemos, crearemos un dialog, donde rescataremos sus dos componentes, el text view y la image view y, si el escaneo ha ido bien, hacemos una consulta en la tabla pedido, filtrando por el id del pedido que ha sido escaneado, y rescatamos el estado del pedido.

Si el estado del pedido es 1, significa que la prenda ya se ha entregado y que al devolverla no habría ningún problema, salvo que hayan pasado 15 días desde la compra, que ya no se podría devolver dicha prenda. Si se devuelve correctamente, en el dialogo saldría la animación de un tick verde diciendo que se ha devuelto, por el contrario, si ya han pasado los 15 días, saldría una animación de una cruz diciendo que no se puede devolver.

Si el estado del pedido es 2, significa que el pedido ya está devuelto por lo que saldría una animación con una cruz diciendo que no se puede devolver otra vez.

Si el estado del pedido es 0, significa que el pedido aún no se ha enviado por lo que no se puede devolver y saldría la animación de la cruz.

El siguiente aspecto a desarrollar será el **responder clientes**, en esta funcionalidad, tendremos una lista con todas las personas que nos han consultado algo, esto es gracias a la variable chat del usuario que se activa cuando estas personas nos escriben su primer mensaje.

```

    /**
     * Rellenamos la lista de contactos
     */
    private fun llenarArrayContacto() {

        //Consultamos todos los usuarios que tengan el tipo 1 (Usuarios normales)
        //y que chat sea 1 (Los que hayan hablado alguna vez con nosotros)
        db.collection( collectionPath: "usuarios")
            .whereEqualTo( field: "tipo", value: 1)
            .whereEqualTo( field: "chat", value: 1)
            .addSnapshotListener{ snapshot, e->
                listaContactos.clear()//Limpiamos la lista

                //Recogemos los usuarios
                for (contacto in snapshot!!) {

                    //Recogemos los datos del usuario
                    val idUsuario = contacto.get("idUsuario").toString()
                    val nombre = contacto.get("nombre").toString()
                    val email = contacto.get("email").toString()
                    val foto = contacto.get("foto").toString()
                    val pass = contacto.get("pass").toString()

                    //Lo añadimos a la lista de contactos
                    val u = Usuario (idUsuario,nombre,email, pass,foto)
                    listaContactos.add(u)
                }

                //Lo asignamos al adaptador
                recy.adapter = contactoAdapter
            }
    }
}

```

Esta será la consulta con la que rellenaremos la lista de usuarios, filtramos por el tipo de usuario y por la variable chat, obtenemos todos los datos de este usuario, lo añadimos a lista y lo mandamos al adaptador para que los muestre.

En la lista de contactos, también mostramos el último mensaje enviado en la conversación y la fecha de este.

```

Picasso.get().load(Uri.parse(ListaContactos[position].foto)).transform(CirculoTransformacion()).into(holder.imgItemCont)

var chatList = ArrayList<Chat>()
firebaseUser = FirebaseAuth.getInstance().currentUser

//Consultamos el chat ordenado por fecha
db.collection( collectionPath: "chat")
    .orderBy( field: "fecha")
    .addSnapshotListener{ snapshot, e->

        for (chat in snapshot!!) {
            //Rescatamos los mensajes si hemos mantenido una conversación
            if (chat.get("idEmisor").toString().equals(idUsuario) && chat.get("idReceptor").toString().equals(ListaContactos[position].idUsuario) ||
                chat.get("idEmisor").toString().equals(ListaContactos[position].idUsuario) && chat.get("idReceptor").toString().equals(idUsuario))

                //Rescatamos la información del chat
                val idEmisor = chat.get("idEmisor").toString()
                val idReceptor = chat.get("idReceptor").toString()
                val mensaje = chat.get("mensaje").toString()
                val fecha = chat.get("fecha")

                val c = Chat(idEmisor, idReceptor, mensaje, fecha.toString())
                //Anadimos ese objeto a la lista de chat
                chatList.add(c)
        }
    }
    //Mostramos el ultimo mensaje
    holder.tvItemContactoMensaje.text = chatList[chatList.size -1].mensaje
    //La fecha de ese mensaje
    val ano = chatList[chatList.size -1].fecha.substring(0, 4)
    val mes = chatList[chatList.size -1].fecha.substring(5, 7)
    val dia = chatList[chatList.size -1].fecha.substring(8, 10)

    val fechaMostrar = dia + "/" + mes + "/" + ano
    holder.tvItemContactoFecha.text = fechaMostrar
}

```

Esto lo hacemos en el adaptador, donde hacemos una consulta rescatando los mensajes que se han intercambiado en la conversación, los ordenamos por fecha, cogemos el último mensaje, su fecha y los mostramos en sus componentes.

Al pulsar en el usuario mostrado, accedemos a la conversación que tenemos con este y le pasamos el usuario.

```
//Recogemos el idUsuario del usuario activo con las SharedPreferences
val pref = activity?.getSharedPreferences(name: "Preferencias", Context.MODE_PRIVATE)

recy.layoutManager = LinearLayoutManager(context)

//Ponemos como idUsuarioEmisor nuestro id, porque somos la persona que nos hemos
//logueado y quienes vamos a mandar mensajes desde nuestra app
idUsuarioEmisor = pref?.getString(key: "idUsuario", defaultValue: "null").toString()

//Como idUsuarioReceptor ponemos el id del usuario que recibimos en esta clase, que será
//el id del usuario que se encontraba en la posición en la que hemos hecho click para abrir
//un chat con el
idUsuarioReceptor = u.idUsuario

//Recogemos y ponemos la foto y el nombre del usuario que recibimos (con el que vamos a hablar)
Picasso.get().load(Uri.parse(u.url)).transform(CirculoTransformacion()).into(imgChatFoto)
tvChatNombre.text = u.nombre
```

En la conversación, rescatamos el id del usuario emisor, que en este caso será el usuario que está usando ahora mismo la aplicación, también rescatamos el usuario receptor que será el id del usuario que recibe esta clase.

También, como esta clase recibe el usuario con el que se está hablando, rescatamos su nombre y su foto y las ponemos en sus componentes.

```
/**
 * Método que recibe, el idEmisor, idReceptor y el mensaje para poder guardarlo en la bbdd
 */
@RequiresApi(Build.VERSION_CODES.O)
private fun enviarMensaje(idEmisor: String, idReceptor: String, mensaje: String){
    //Creamos un hashMap con los valores del mensaje
    var hashMap : HashMap<String, String> = HashMap()
    hashMap.put("idEmisor", idEmisor)
    hashMap.put("idReceptor", idReceptor)
    hashMap.put("mensaje", mensaje)
    hashMap.put("fecha", LocalDateTime.now().toString())

    //Añadimos el chat(mensaje,idEmisor,idReceptor,fecha) a la bbdd
    db.collection(collectionPath: "chat").add(hashMap)
}
```

Cuando pulsamos en el botón de enviar el mensaje, primero miramos a ver si el mensaje que va a enviar está vacío, si no lo está, lo enviamos, llamando a este método, que recibe el id del emisor, del receptor y el mensaje.

Creamos un hashMap con los valores recibidos, además le añadimos la fecha, e insertamos el mensaje en la base de datos.

```

private fun leerMensaje(idEmisor: String, idReceptor: String){

    //Consultamos el chat y ordenamos por fecha
    db.collection( collectionPath: "chat")
        .orderBy( field: "fecha")
        .addSnapshotListener{ snapshot, e->
            chatList.clear()//limpiamos la lista

            for (chat in snapshot!!) {

                //Rescatamos los mensajes si hemos mantenido una conversación
                if(chat.get("idEmisor").toString().equals(idEmisor) && chat.get("idReceptor").toString().equals(
                    idReceptor
                ) || chat.get("idEmisor").toString().equals(idReceptor) && chat.get("idReceptor").toString().equals(
                    idEmisor
                )){

                    //Recogemos los datos del chat
                    val idEmisor = chat.get("idEmisor").toString()
                    val idReceptor = chat.get("idReceptor").toString()
                    val mensaje = chat.get("mensaje").toString()
                    val fecha = chat.get("fecha").toString()

                    val c = Chat(idEmisor, idReceptor, mensaje, fecha)
                    //Añadimos ese objeto a la lista de chat
                    chatList.add(c)
                }
            }

            val chatAdapter = ChatListAdapter(requireContext(), chatList)
            //Le asignamos el adaptador
            recy.adapter = chatAdapter
        }
    }
}

```

Cuando accedemos a esta clase, llamamos al método de leer el mensaje, en este método tenemos una consulta donde rescatamos todos los mensajes que han sido intercambiados por el emisor y el receptor, después rescatamos sus valores, creamos el objeto del chat y lo añadimos a la lista, posteriormente llamamos al adaptador.

El adaptador de esta clase tiene la misma peculiaridad que la clase de contactar con el administrador, ya que dependiendo de quién envíe el mensaje, este saldrá a un lado de la pantalla u al otro.

Esta ha sido la explicación del código, he resumido y he explicado los métodos más importantes de cada clase, intentando siempre explicar cómo funcionan dichas clases.

# Testeo y pruebas de la solución

## 1. Plan de pruebas (unitarias, integración, sistema y usuarios)

Las pruebas son un factor fundamental en cualquier aplicación, son imprescindibles ya que una vez obtenido el código ejecutable de un programa depurado lo máximo posible hay que comprobar exhaustivamente su funcionalidad, ya que es necesario probar si la solución diseñada genera los resultados correctos, para ello se deben realizar tantas pruebas como se consideren necesarias.

### - **Pruebas Unitarias:**

Las pruebas unitarias consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños test que validan el comportamiento de un objeto y la lógica.

Estos test suelen realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente se llevan a cabo por los propios desarrolladores de la aplicación.

Con estas pruebas se detectan errores que no se podrían detectar hasta fases más avanzadas de pruebas. Por lo que realizar estas pruebas supone al final, un ahorro de tiempo.

Hay dos tipos de enfoques principales:

- Enfoque estructural o de caja blanca: Se verifica la estructura interna del componente con independencia de la funcionalidad establecida para el mismo.
- Enfoque funcional o de caja negra: Se comprueba el correcto funcionamiento de los componentes del sistema de información, analizando las entradas y salidas y verificando si el resultado es el esperado.

### - **Pruebas de integración:**

En estas pruebas se examinan las interfaces entre grupos de componentes o subsistemas para asegurar que son llamados cuando es necesario y que los datos que se transmiten son los correctos.

A menudo se combinan los tipos de prueba unitarias y de integración, hay dos tipos de pruebas de integración.

- Integración incremental: Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.

Hay tres estrategias de integración, de arriba abajo, de abajo a arriba y estrategias combinadas.

- Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes.

- **Pruebas del sistema:**

Las pruebas del sistema tienen como objetivo ejercitarse profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen.

En esta etapa se pueden distinguir los siguientes tipos de pruebas, cada una con un objetivo diferente:

- **Pruebas funcionales:** Dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado.
- **Pruebas de comunicaciones:** Determinan que las interfaces entre los componentes del sistema funcionan adecuadamente.
- **Pruebas de rendimiento:** Consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos.
- **Pruebas de volumen:** Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos.
- **Pruebas de sobrecarga:** Consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndolo a cargas masivas.
- **Pruebas de disponibilidad de datos:** Consisten en demostrar que el sistema puede recuperarse ante fallos.
- **Pruebas de facilidad de uso:** Consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios.
- **Pruebas de operación:** Consisten en comprobar la correcta implementación de los procedimientos de operación.
- **Pruebas de entorno:** Consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de seguridad:** Consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

- **Pruebas de usuario:**

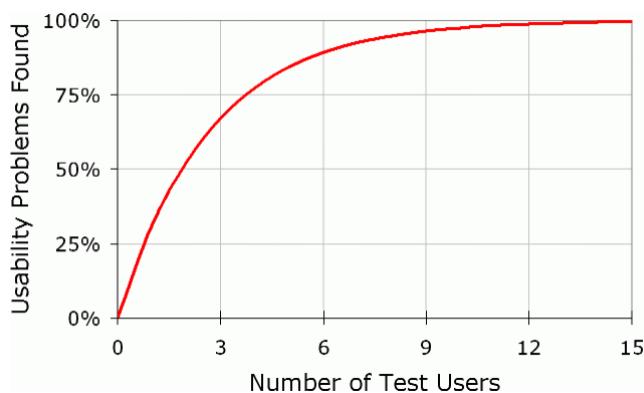
Estas pruebas son un método de investigación cualitativa que se basan en la observación y el análisis de cómo un grupo determinado de usuarios utiliza nuestro producto.

Durante la prueba vamos a observar el grado de eficacia, eficiencia y satisfacción con el que los usuarios de nuestro producto logran concretar objetivos específicos.

Estas pruebas nos brindan beneficios significativos para nuestros proyectos, ya que nos permiten detectar problemas de usabilidad por muy bajo costo, ya que este tipo de estudios es mucho menos costoso que otros.

Y lo más importante nos ayudan a recordar que **nosotros no somos el usuario**.

Una de las mejores cosas sobre las pruebas con usuarios es que, en el año 2000, Nielsen publicó un estudio donde demostró que solo con 5 usuarios se podían detectar el 85% de los problemas de usabilidad de una interfaz<sup>3</sup>. Entonces solo necesitamos 5 usuarios para encontrar la mayoría de fallos.



## Pruebas B&A

### Pruebas Unitarias:

- *Caso de prueba 1: Registro*

En esta prueba comprobaremos que el usuario se registra correctamente (en la autenticación de Firebase y en la base de datos de Firestore) y que cuando introduzca datos nulos o no válidos no le dejará registrarse y le mostrará un mensaje de error.

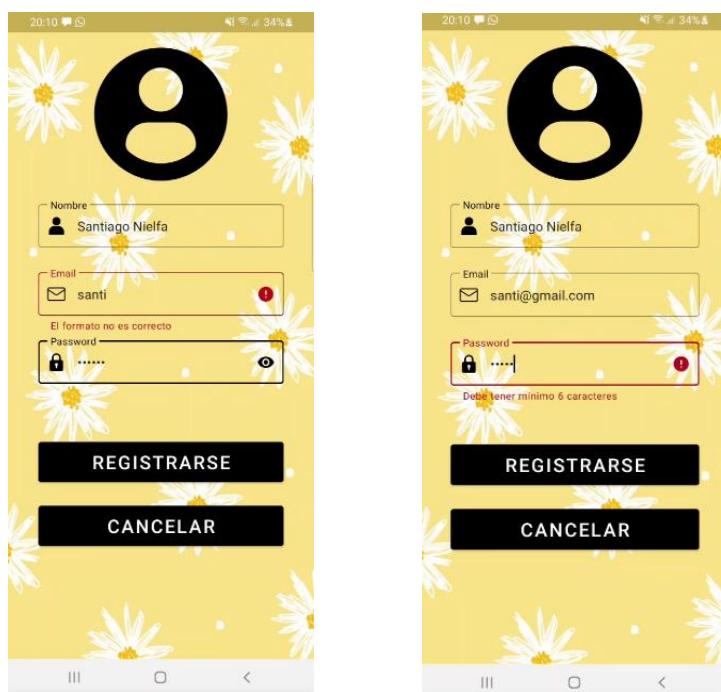
Hemos intentado registrarnos introduciendo campos nulos y nos muestra un mensaje de error en cada uno de los campos. Muestra errores en todos los campos. **El resultado es el esperado.**



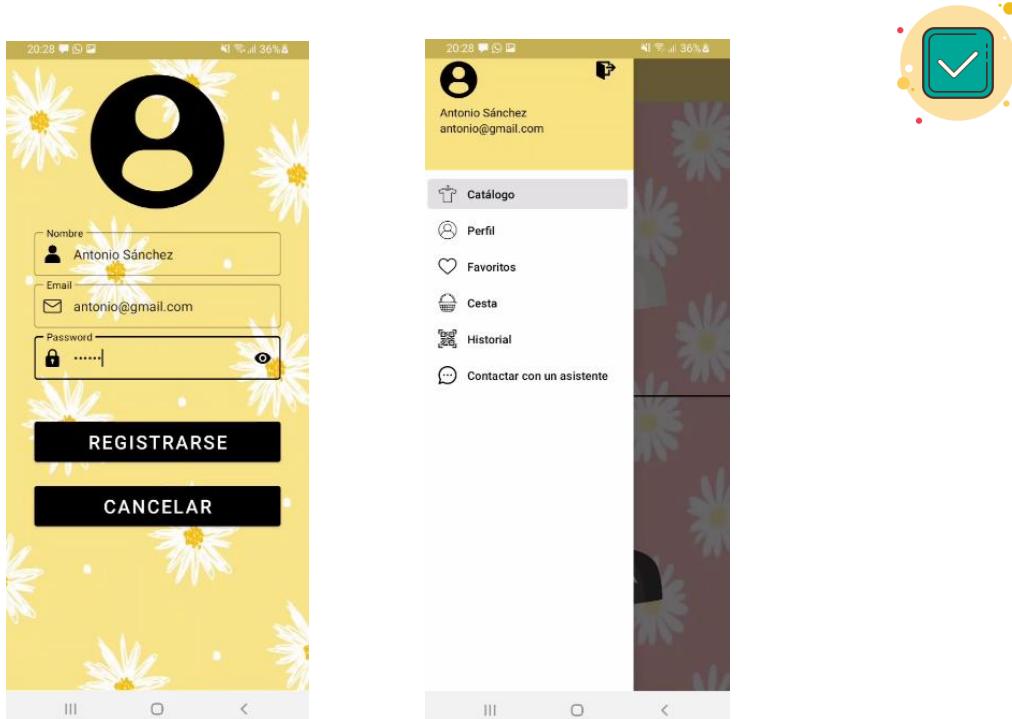
<sup>3</sup> <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>



Hemos intentado registrarnos introduciendo un email con un formato no válido e introduciendo una contraseña con una longitud no válida. Muestra errores en los campos no válidos. **El resultado es el esperado.**



Hemos intentado registrarnos introduciendo todos los valores válidos en todos los campos y sin introducir foto. La aplicación introduce una foto por defecto si el usuario no selecciona foto, nos registra correctamente y entramos a la aplicación. **El resultado es el esperado.**

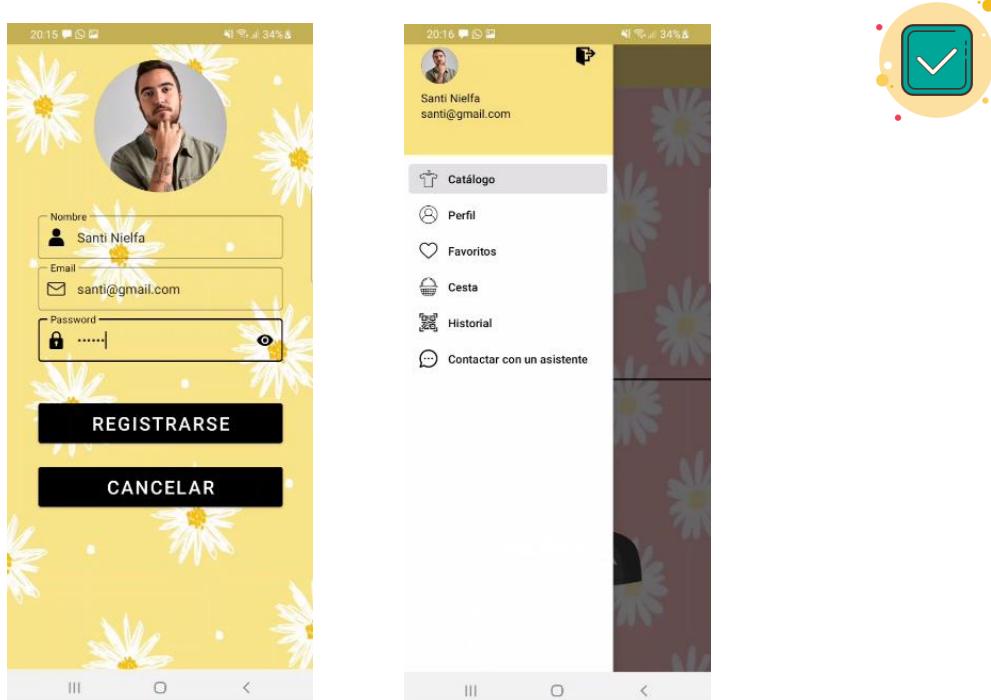


Comprobamos que el usuario ha sido insertado en la autenticación y en Firestore.

byabea-e5b76		
usuarios	usuarios	F0c7f5dY0zPJ... ...
<a href="#">+ Iniciar colección</a> cesta chat favoritos pedidos prendas tipo <a href="#">usuarios &gt;</a>	<a href="#">+ Agregar documento</a> 5n3cwRCHL7g7S2nZ0f72Dft4R8e2 F6vhyecPYv0B1EqLRpNqsMmVHhf2 <b>F0c7f5dY0zPJ... ...</b> McNKD147y9bYC1Ff63F9w5hwXyW2 U4sReETreAd0wo6cbm37D4njRpS2 U1NV1vm6qrdfvlnVGcL3tACpFA72 ZDX1Rj6mjKQS7U273zh0YpDgDcZ2 eD8t5GHkyceImQUmYfH1r6E6Kbr1 qofc9nWEFsXqILec0pLQ71c21Ph1	<a href="#">+ Iniciar colección</a> <a href="#">+ Agregar campo</a> chat: 0 email: "antonio@gmail.com" foto: "https://firebasestorage.googleapis.com/v0/b/byabea-e5b76.appspot.com/o/fotosUsuarios%2Ffdbb57e20-27f7-4deb-88f0-98430ca32a02?alt=media&token=009af310-63b5-4a90-ac20-e902fad93c37" idUsuario: "F0c7f5dY0zPJ... ..." nombre: "Antonio Sánchez" pass: "123456" tipo: 1

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario ↑
maria@gmail.com	✉	7 jun. 2021	7 jun. 2021	5n3cwRCHL7g7S2nZ0f72Dft4R8e2
santi@gmail.com	✉	8 jun. 2021	8 jun. 2021	F6vhyecPYvOB1EqLRpNqsMmVHf...
<b>antonio@gmail.com</b>	✉	8 jun. 2021	8 jun. 2021	F0c7f5dY0zPJpeg73z4bXsdYju2

Hemos intentado registrarnos introduciendo todos los valores válidos en todos los campos introduciendo foto. Nos registramos correctamente con nuestros datos y nuestra foto y entramos a la aplicación. **El resultado es el esperado.**



Comprobamos que el usuario ha sido insertado en la autenticación y en Firestore.

byabea-e5b76	usuarios	F6vhyecPYvOB1EqLRpNqsMmVHf2
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
cesta	5n3cwRCHL7g7S2nZ0f72Dft4R8e2	+ Agregar campo
chat	F6vhyecPYvOB1EqLRpNqsMmVHf2	chat: 0
favoritos	F0c7f5dY0zPJpeg73z4bXsdYju2	email: "santi@gmail.com"
pedidos	McNKD147y9bYC1FF63F9w5hwXyW2	foto: "https://firebasestorage.googleapis.com/v0/b/byabea-e5b76.appspot.com/o/fotosUsuarios%2Fa98f615-7180-45c3-b1fa-221c502802ad?alt=media&token=e584d613-7d75-4e1b-a314-d0ac38721d8d"
prendas	U4sReETreAdwo6cbm37D4njRpS2	idUsuario: "F6vhyecPYvOB1EqLRpNqsMmVHf2"
tipo	U1NV1vm6qrdf1nVGcL3tAcFA72	nombre: "Santi Nielfa"
usuarios >	ZDX1Rj6mjKQS7U273zh0YpDgDcZ2	pass: "123456"
	eD8t5GHkyceImQUmYfH1r6E6Kbr1	tipo: 1
	qofc9nWEFsXqILec0pLQ71c21Ph1	

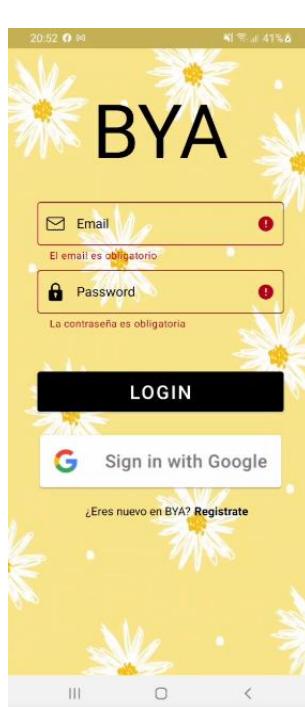
Identificador	Proveedores	Fecha de creacion	Fecha de acceso	UID de usuario ↑
maria@gmail.com	✉	7 jun. 2021	7 jun. 2021	5n3cwRCHL7g7S2nZ0f72Dft4R8e2
senti@gmail.com	✉	8 jun. 2021	8 jun. 2021	F6vhyecPYvOB1EqLRpNqsMmVH...
antonio@gmail.com	✉	8 jun. 2021	8 jun. 2021	F0c7f5dY0zPJPeg73z4bXsdYjul2

- *Caso de prueba 2: Login*

En esta prueba comprobaremos que el usuario es capaz de iniciar correctamente con su correo y su contraseña, en el caso de que estos no sean los correctos, nos saltará un mensaje informativo, y en el caso de introduzca campos vacíos que también nos salga un mensaje de error.

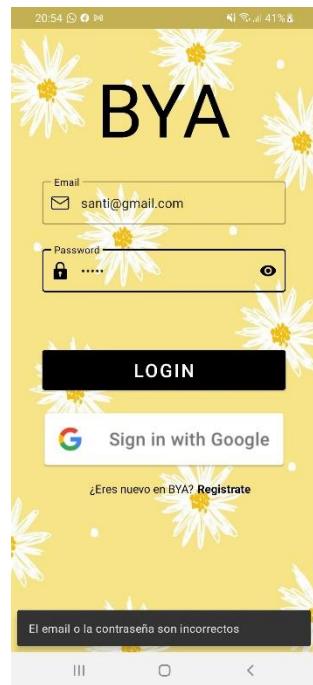
También comprobaremos que inicia sesión correctamente a través de Google.

Hemos intentado iniciar sesión sin llenar los campos. Nos han saltado varios mensajes de error diciendo que los campos están vacíos. **El resultado es el esperado.**



Hemos intentando iniciar sesión introduciendo datos erróneos. Nos ha saltado un mensaje informándonos de que los datos introducidos no son correctos. **El resultado es el esperado.**

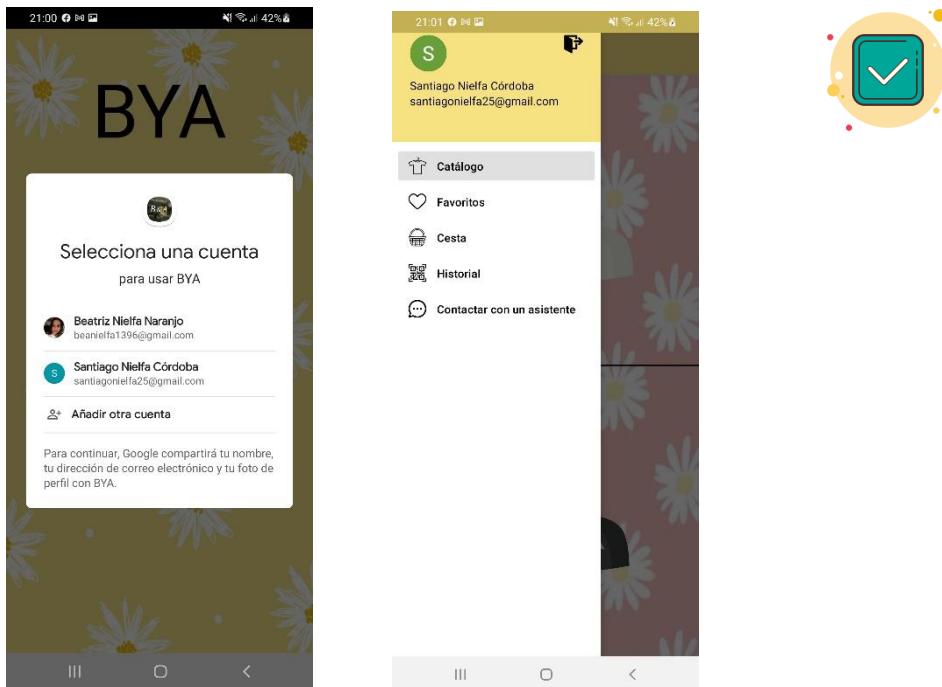




Hemos intentando iniciar sesión introduciendo datos correctos. Hemos iniciado sesión sin problemas. **El resultado es el esperado.**



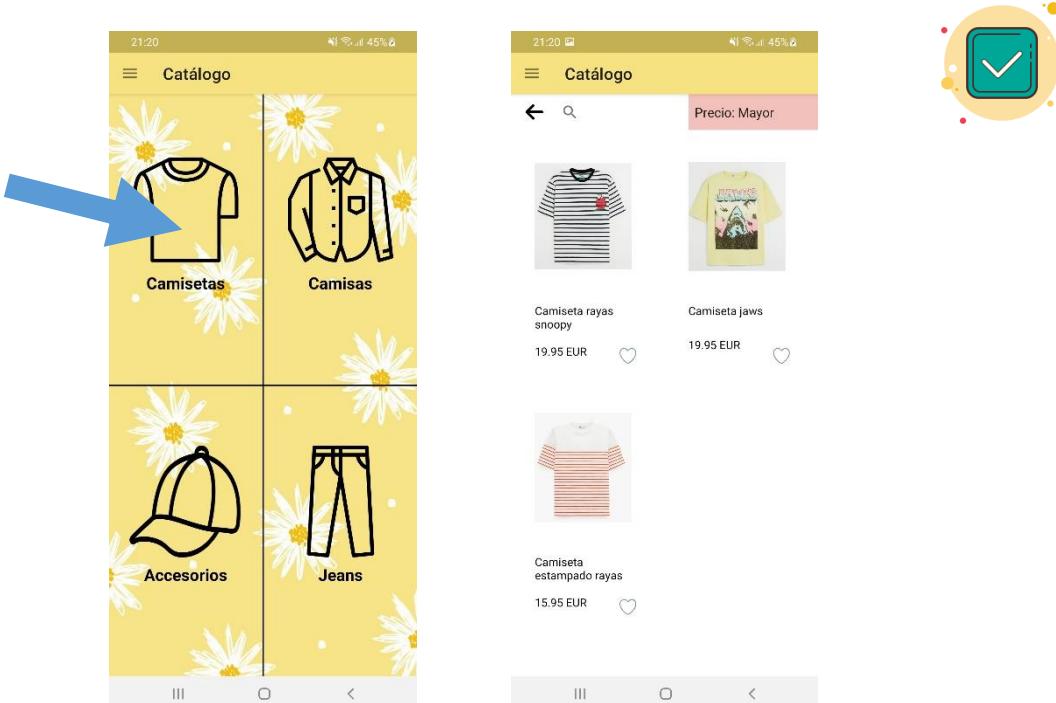
Hemos intentando iniciar sesión a través de Google. Hemos iniciado sesión sin problemas. **El resultado es el esperado.**



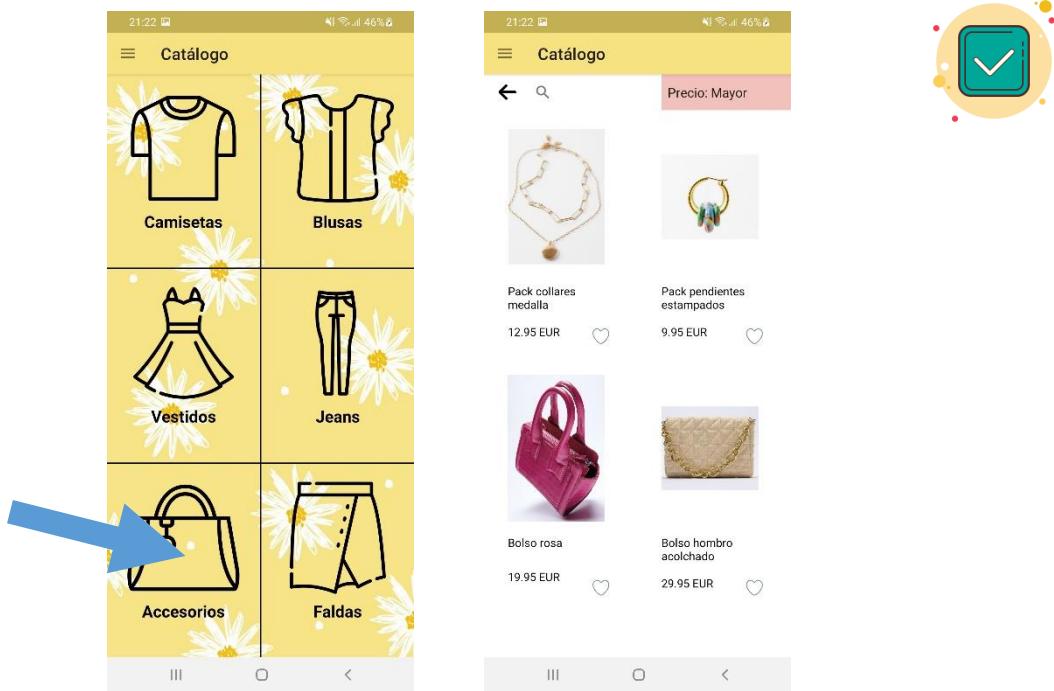
- *Caso de prueba 3: Catálogo*

En esta prueba comprobaremos que el usuario visualiza todos los productos y además que se filtran por el tipo de prenda seleccionada.

Hemos hecho clic en la opción camisetas de hombres. Visualizamos todas las camisetas de hombres. **El resultado es el esperado.**



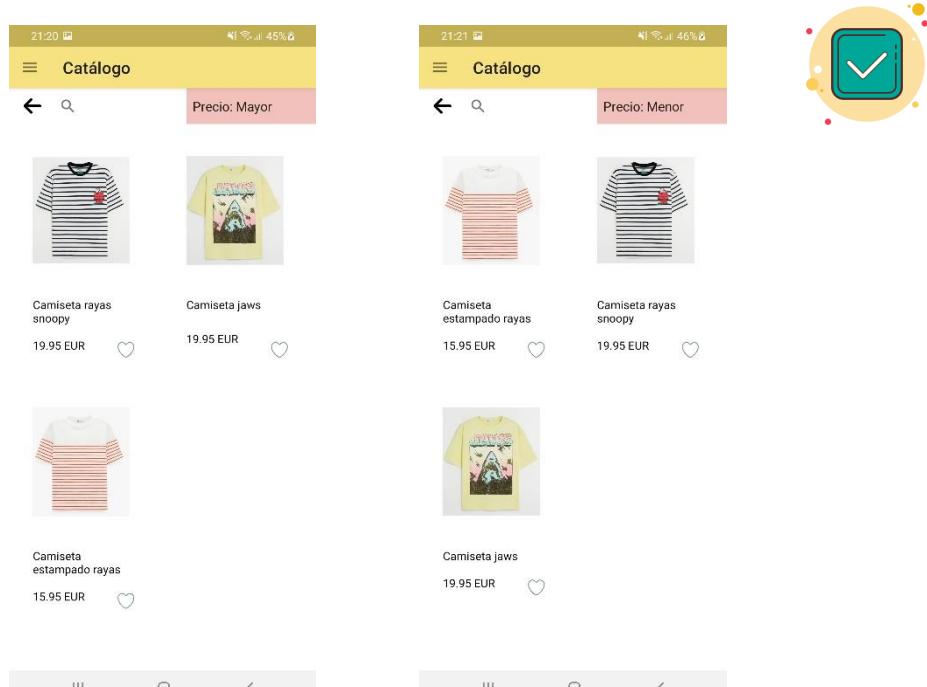
Hemos hecho clic en la opción accesorios de mujeres. Visualizamos todas las camisetas de hombres. **El resultado es el esperado.**



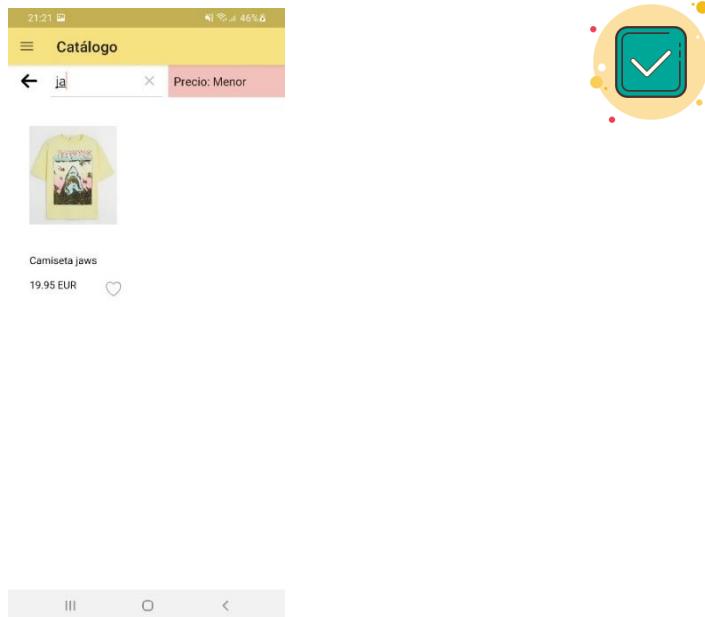
- *Caso de prueba 4: Filtros*

En esta prueba comprobaremos que el usuario puede filtrar por precio y por el nombre de la prenda.

Hemos cambiado el filtro del precio. Cuando lo ponemos a mayor se nos ordenan las prendas de mayor precio a menor precio y viceversa cuando ponemos menor en el filtro. **El resultado es el esperado.**



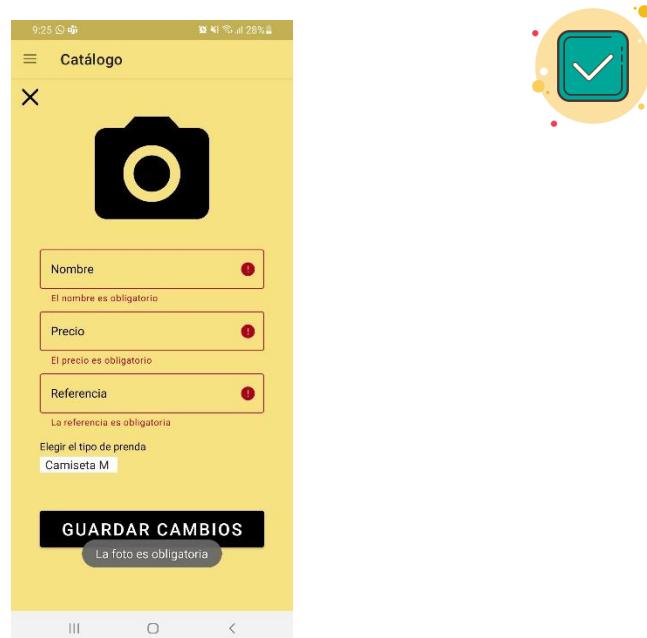
Hemos filtrado por nombre. Cuando empezamos a escribir el nombre de una camiseta nos aparecen las prendas que contienen esas letras. **El resultado es el esperado.**



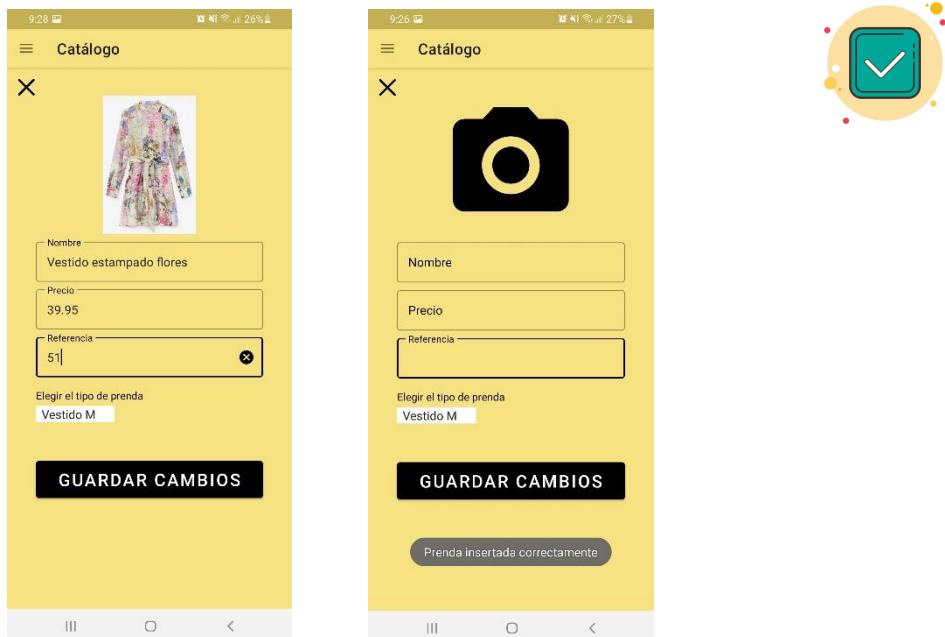
- *Caso de prueba 5: Subir prenda nueva*

En esta prueba comprobaremos que el administrador puede subir una prenda correctamente y que cuando introduzca datos nulos o no válidos no te deje añadir la prenda y salte un mensaje de error.

Hemos intentado añadir una prenda con los campos vacíos (incluida la foto). Nos salen mensajes de errores en todos los campos diciendo que están vacíos, y también un mensaje en la pantalla informando de que hay que introducir una foto. **El resultado es el esperado.**



Hemos intentado añadir una prenda con los campos rellenados correctamente. Nos inserta la prenda sin problemas. **El resultado es el esperado.**



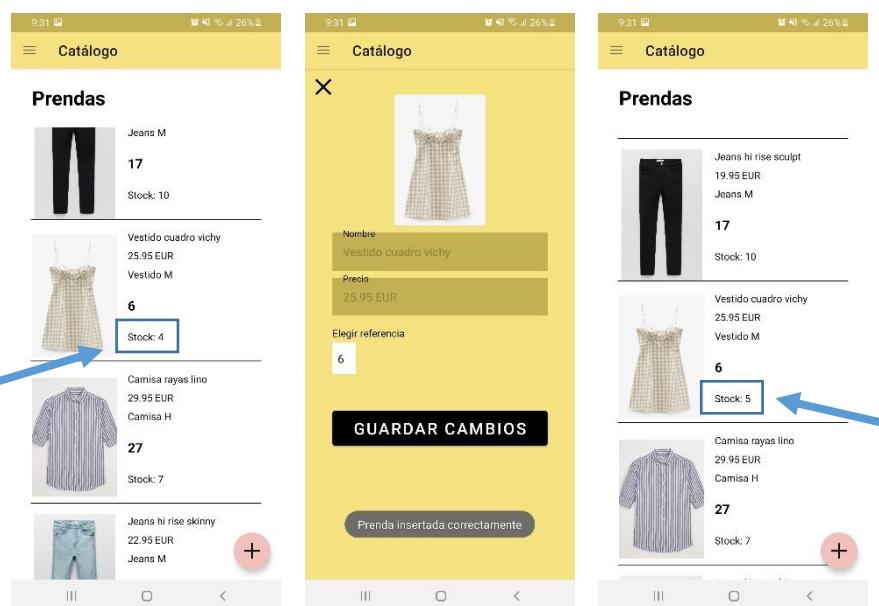
Comprobamos que la prenda se ha añadido correctamente en Firestore.

+ Iniciar colección	+ Agregar documento	+ Iniciar colección
cesta	5112f066-a912-4f9f-b873-e2	+ Agregar campo
chat	51d847fd-4a3b-47ee-b8b9-40	foto: "https://firebasestorage.googleapis.com/v0/b/byabea-e5b76.appspot.com/o/fotosPrendas%2F806ae603-28e2-452d-9e4e-025f70b5122c?alt=media&token=39d0eb4c-8120-463d-93f61bdbba604662"
favoritos	55f264b1-dd33-48d7-880f-23	idPrenda: "ab66464b-15cf-4cf8-b8e5-64f1ab646b84"
pedidos	5cf543e8-caec-4623-b5d2-67	idTipo: "2"
<b>prendas</b>	62d490bc-f202-4cf8-92be-1d	nombre: "Vestido estampado flores"
tipo	7c2382a6-bbf9-4204-9491-13	precio: "39.95"
usuarios	8a5d2946-c439-4ec8-8d2e-5e	referencia: "51"

- *Caso de prueba 6: Reponer stock*

En esta prueba comprobaremos que el administrador puede añadir stock correctamente. En este caso de prueba el fallo humano está muy controlado ya que no hay que introducir datos, solo seleccionar la referencia de una lista.

Hemos intentado añadir stock de una prenda seleccionando su referencia en una lista donde se cargan todas las referencias existentes. El stock se ha añadido correctamente. **El resultado es el esperado.**



- Caso de prueba 7: Manipular prenda existente

En esta prueba comprobaremos que el administrador puede editar y borrar una prenda correctamente, cuando edite e introduzca datos nulos tendría que salir un mensaje de advertencia y cuando intente borrar debería salir un mensaje de confirmación por si se había equivocado e iba a eliminar la prenda por error.

Hemos intentado editar una prenda introduciendo datos nulos. No nos deja editarla y nos saltan mensajes de advertencia. **El resultado es el esperado.**



Hemos intentado editar una prenda introduciendo datos correctos. La prenda se edita correctamente. **El resultado es el esperado.**

The first screenshot shows a list of items in the catalog. A blue arrow points to the blouse's price and size information. The second screenshot shows the edit screen for the blouse, with fields for Name (Camiseta amarilla volantes), Price (7.99), Reference (36), and Type (Camiseta M). The third screenshot shows the updated catalog with the edited blouse information, accompanied by a green checkmark icon.

Prenda	Nombre	Precio	Referencia	Tipo de prenda
Camisetas volantes	Camiseta amarilla volantes	7.99 EUR	36	Camiseta M
Jeans hi rise sculpt		19.95 EUR		Jeans M
Vestido cuadro vichy		25.95 EUR		Vestido M
Camisa rayas lino				

Hemos intentado borrar una prenda. Nos salta un dialogo de confirmación y seleccionando "Sí" nos borra la prenda. **El resultado es el esperado.**

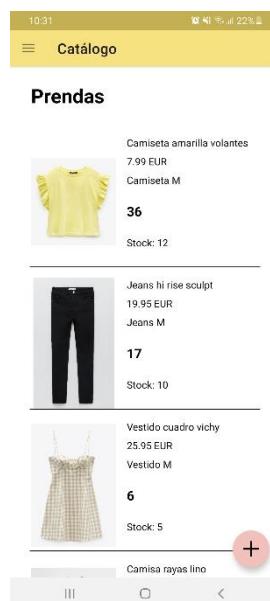
The first screenshot shows a list of items in the catalog. The second screenshot shows a confirmation dialog box asking if the user wants to delete the existing item. The third screenshot shows the catalog after the deletion, with a message at the bottom stating "Prenda eliminada con éxito". A blue arrow points to this success message. A green checkmark icon is also present.

Prenda	Nombre	Precio	Referencia	Tipo de prenda
Blusa estampada volantes	Blusa estampada volantes	25.95 EUR	10	Blusa M
Gorra verde		15.95 EUR		Accesorio H
Falda lunares volantes		22.95 EUR		Falda M
Vestido mini estampado		25.95 EUR		Vestido M

- Caso de prueba 8: Listado de prendas subidas

En esta prueba comprobaremos que desde el administrador podemos ver la lista de todas las prendas que hemos subido con anterioridad.

Hemos intentado ver el listado de prendas. El listado de prendas se muestra correctamente. **El resultado es el esperado.**



- Caso de prueba 9: Añadir a la cesta

En esta prueba comprobaremos que a través del detalle de la prenda y seleccionando la talla, podemos añadir dicha prenda a la cesta.

Hemos intentado añadir una prenda a la cesta sin seleccionar la talla. Nos muestra un mensaje avisándonos de que la talla es obligatoria. **El resultado es el esperado.**



Hemos intentado añadir una prenda a la cesta seleccionando la talla. La prenda se añade a nuestra cesta correctamente. **El resultado es el esperado.**

The image shows two screenshots of a mobile application interface. The left screenshot is titled 'Catálogo' (Catalog) and displays a yellow t-shirt with a 'JAWS' graphic. Below it, the product details are shown: 'Camiseta jaws' at '19.95 EUR'. A heart icon indicates it's favorited. The right screenshot is titled 'Cesta' (Cart) and shows the same yellow t-shirt added to the cart. The cart summary says 'Camiseta jaws' at '19.95' with size 'M'. A large green checkmark icon with a yellow glow is positioned above the cart screen. At the bottom of each screen is a navigation bar with three icons: a list, a search, and a back arrow.

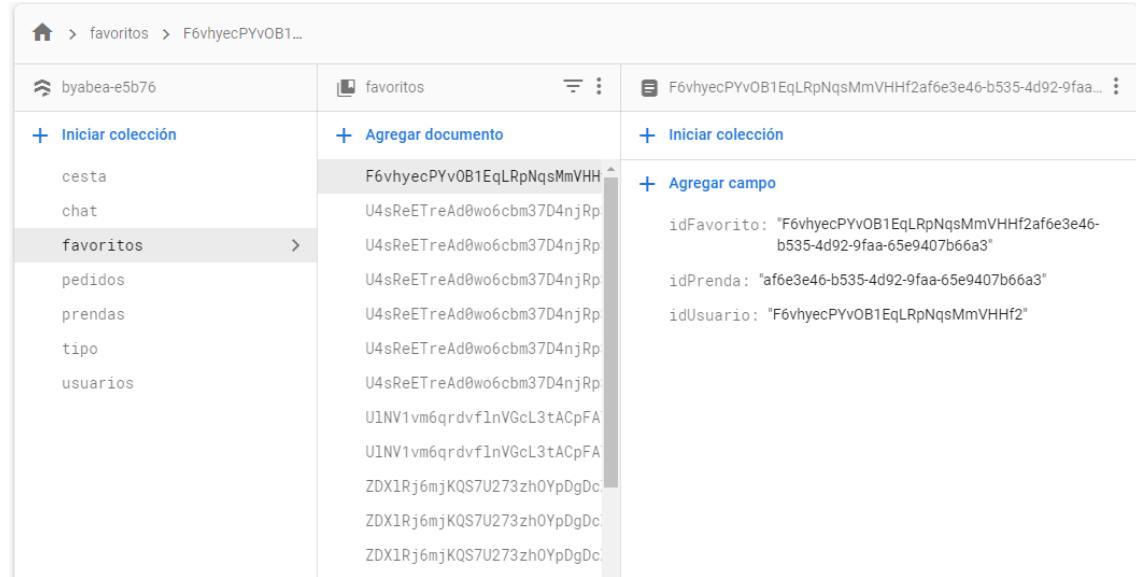
#### - Caso de prueba 10: Añadir a favoritos

En esta prueba comprobaremos que podemos añadir y eliminar prendas a favoritos a través del listado de prendas y del detalle.

Hemos intentado añadir una prenda a favoritos a través del listado de prendas. La prenda se añade a favoritos correctamente. **El resultado es el esperado.**

The image shows two screenshots of a mobile application interface. The left screenshot is titled 'Catálogo' (Catalog) and displays two shirts: a striped button-down and a patterned short-sleeve. Below them are their details: 'Camisa rayas lino' at '29.95 EUR' and 'Camisa estampado' at '22.95 EUR', each with a heart icon. The right screenshot is also titled 'Catálogo' and shows the same two shirts. The patterned shirt now has a red heart icon below it, indicating it has been favorited. A large green checkmark icon with a yellow glow is positioned above the catalog screen. At the bottom of each screen is a navigation bar with three icons: a list, a search, and a back arrow.

Comprobamos que la prenda se añade correctamente a la tabla de favoritos de Firestore.

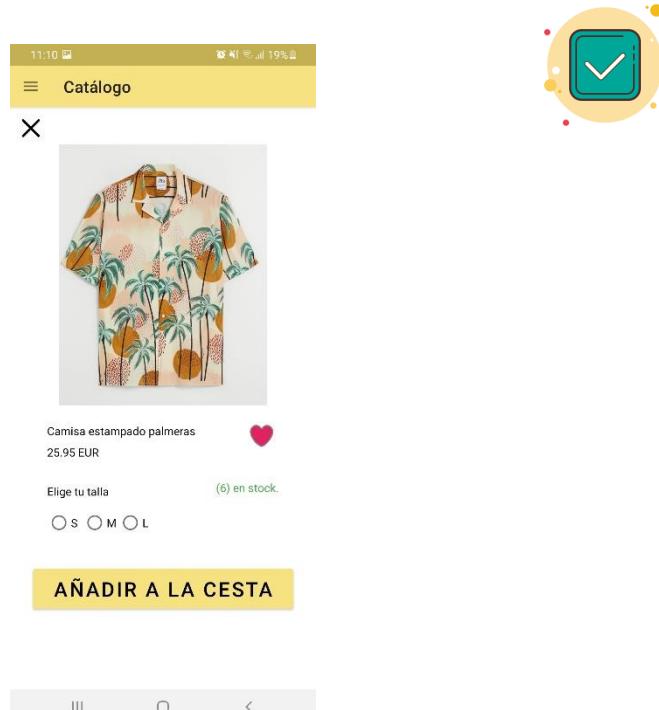


The screenshot shows the Firebase Firestore interface. On the left, there's a sidebar with collections: cesta, chat, favoritos (which is selected), pedidos, prendas, tipo, and usuarios. In the center, under the 'favoritos' collection, there's a sub-section for 'Añadir documento'. A new document is being created with the following fields:

- idPrenda: "af6e3e46-b535-4d92-9faa-65e9407b66a3"
- idUsuario: "F6vhyecPYvOB1EqLRpNqsMmVHHf2"
- idFavorito: "F6vhyecPYvOB1EqLRpNqsMmVHHf2af6e3e46-b535-4d92-9faa-65e9407b66a3"

The 'favoritos' field in the document itself also contains the ID of the newly created document.

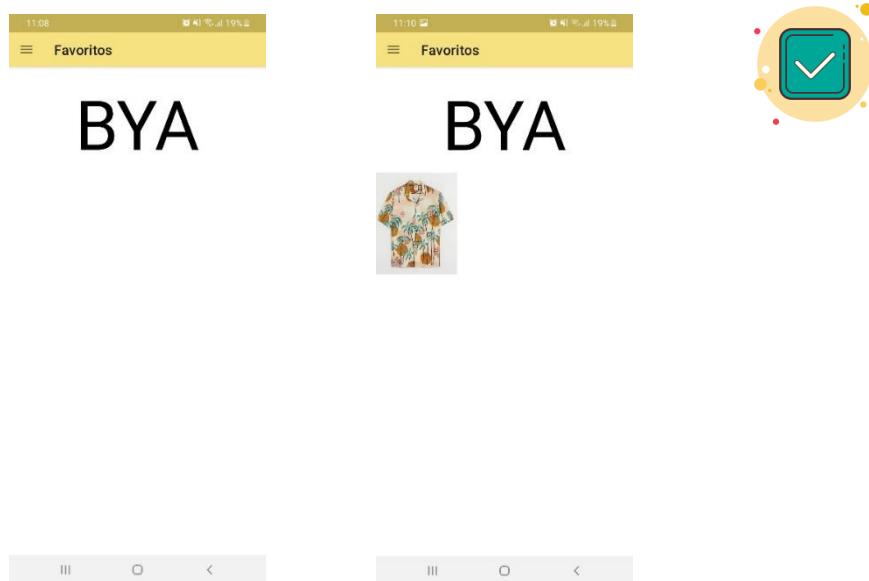
Hemos intentado acceder al detalle de una prenda que está en favoritos. El corazón de la prenda se muestra en rojo, lo que significa que esa prenda está añadida favoritos. **El resultado es el esperado.**



- Caso de prueba 11: Listar favoritos

En esta prueba comprobaremos que tendremos acceso al listado total de prendas que hemos añadido previamente a favoritos.

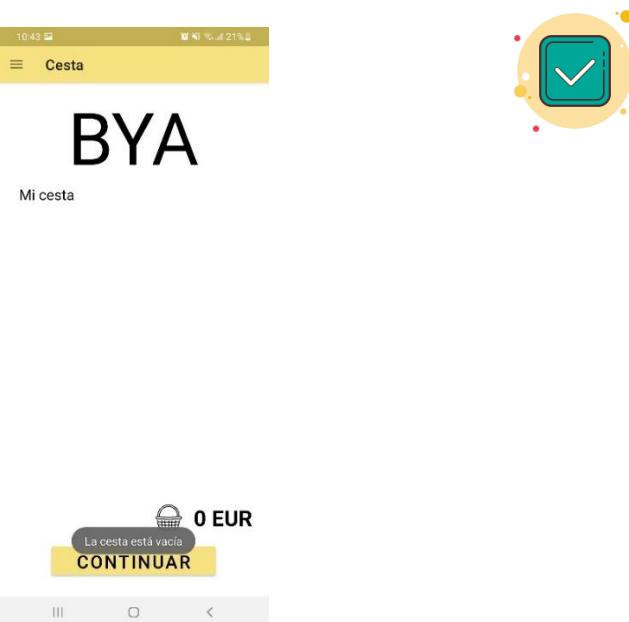
Hemos intentado acceder al listado de prendas favoritas. Tenemos acceso al listado de favoritos. **El resultado es el esperado.**



- Caso de prueba 12: Realizar pedido

En esta prueba comprobaremos que, en la cesta, los productos que al final no nos interesen los podemos eliminar, y una vez tengamos todos los productos que queramos comprar, finalizar esta compra añadiendo la ubicación y los datos de la tarjeta. Por otro lado, si la cesta está vacía no se podrá realizar ningún pedido.

Hemos intentado seguir con el pedido con la cesta vacía. No se puede seguir con el pedido. **El resultado es el esperado.**



Hemos intentado eliminar prendas de la cesta. Las prendas se eliminan y tanto la lista de la cesta como el nuevo precio se actualizan correctamente. **El resultado es el esperado.**

The image shows two side-by-side screenshots of a mobile application interface. Both screens have a yellow header bar with the time (10:43) and battery level (21%). The header bar also contains a menu icon and the word "Cesta".

**Left Screenshot (Initial State):**

- Mi cesta:** Shows a blue baseball cap labeled "Gorra bordado snoopy" at 15.95 EUR, size M. A red circle with a minus sign is positioned to its right.
- Mi cesta:** Shows a yellow t-shirt labeled "Camiseta jaws" at 19.95 EUR, size M. A red circle with a minus sign is positioned to its right.
- Total:** 35,9 EUR
- Buttons:** CONTINUAR, a navigation bar with three icons (menu, home, back).

**Right Screenshot (After Removal):**

- Mi cesta:** Shows the same yellow t-shirt labeled "Camiseta jaws" at 19.95 EUR, size M. A red circle with a minus sign is positioned to its right.
- Total:** 19,95 EUR
- Buttons:** CONTINUAR, a navigation bar with three icons (menu, home, back).

A large orange circular icon with a checkmark and radiating lines is positioned above the second screenshot, indicating a successful action.

Hemos intentado, una vez tenemos la cesta definitiva, seguir con el pedido, a la hora de seleccionar la ubicación, la dejaremos vacía. Nos coge la ubicación que tenemos en ese momento. **El resultado es el esperado.**

The image shows a screenshot of a mobile application interface. The top bar shows the time (12:04), battery level (18%), and signal strength. The header bar has a menu icon and the word "Cesta".

**X**

Seleccione su ubicación

A map is displayed, showing various locations such as "Panificadora IMEDIO", "Alba Maquinaria Agrícola SL", "Casa de La Virgen", "Bar Cande", "Parque", and "Bar Restaurante la Plaza". The map includes zoom controls (+, -).

**CONTINUAR**

A navigation bar with three icons (menu, home, back) is at the bottom.

An orange circular icon with a checkmark and radiating lines is positioned above the map.

Hemos intentado seleccionar la ubicación donde queremos que nos llegue el pedido. Nos coge la ubicación seleccionada. **El resultado es el esperado.**



Hemos intentado introducir datos erróneos en la tarjeta. Nos saltan mensajes de advertencia. **El resultado es el esperado.**



The image contains three side-by-side screenshots of a mobile application interface for selecting a payment method. Each screenshot shows a yellow header bar with the word "Cesta" and a red "X" icon. Below the header, the text "Seleccione un método de pago" is displayed, followed by two payment method icons: "VISA" and "MasterCard".

- Screenshot 1:** Shows a "NÚMERO DE TARJETA" input field with the number "BYA". A red error message "La tarjeta debe contener 16 dígitos" is displayed above the input field.
- Screenshot 2:** Shows a "NÚMERO DE TARJETA" input field with the number "12". A red error message "La tarjeta debe contener 16 dígitos" is displayed above the input field.
- Screenshot 3:** Shows a "NÚMERO DE TARJETA" input field with the number "123". A red error message "El CSV debe contener 3 dígitos" is displayed below the input field.

Below the payment method selection, the total amount "19.95 EUR" is shown. At the bottom of each screenshot is a yellow "PAGAR" button. In the first two screenshots, a black callout box above the "PAGAR" button says "Por favor seleccione un método de pago".



Hemos intentado introducir datos válidos en la tarjeta. Hemos finalizado la compra y el stock ha bajado y el pedido se ha añadido a la base de datos. **El resultado es el esperado.**



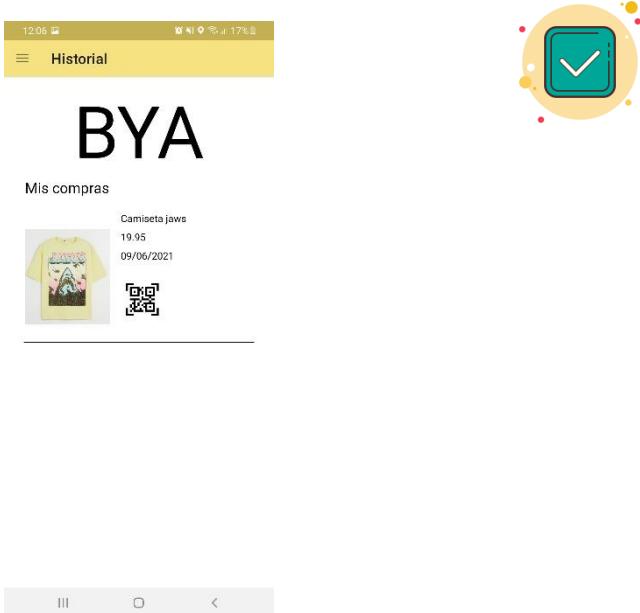
Comprobamos que el pedido ha sido añadido en Firebase.

A screenshot of the Firebase Realtime Database console. The left sidebar shows a navigation path: home &gt; pedidos &gt; 00f41c29-4c39-... . The main area displays a table with three columns. The first column contains database references: "byabea-e5b76", "+ Iniciar colección", "cesta", "chat", "favoritos", "pedidos" (which has a dropdown arrow), "prendas", "tipo", and "usuarios". The second column shows a list of documents under the "pedidos" collection, with the first few items being "00f41c29-4c39-46cf-9900-6b4c855ecf57", "461f505d-dac1-43c6-98b0-4522", and "50dc2dbd-dd1a-4c46-af04-849c". The third column shows the detailed data for the document "00f41c29-4c39-46cf-9900-6b4c855ecf57", which includes fields like "estado": 0, "fechaCompra": "09/06/2021", "idPedido": "00f41c29-4c39-46cf-9900-6b4c855ecf57", "idPrenda": "26740bd7-ad47-4bf4-9f54-fdb9fc9c536", "idUsuario": "F6vhyecPYvOB1EqLRpNqsMmVHHf2", "latitud": "38.85451628387631", "longitud": "-3.7723754346370697", and "talla": "M".

- Caso de prueba 13: Ver Historial

En esta prueba comprobaremos que tendremos acceso al listado total de historial de prendas que hemos comprado.

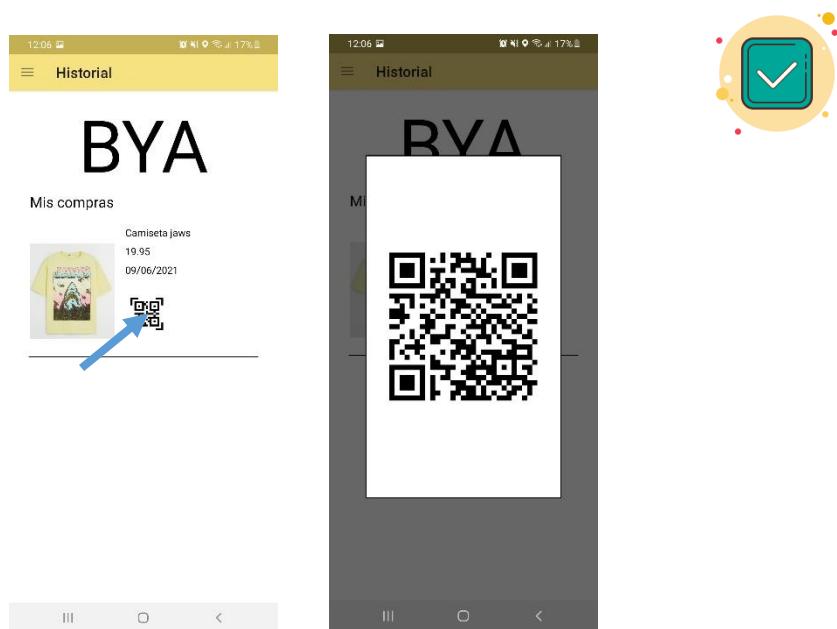
Hemos intentado acceder al listado del historial de prendas. Tenemos acceso al listado. **El resultado es el esperado.**



- Caso de prueba 14: Generar QR

En esta prueba comprobaremos que podremos generar un QR con la información del pedido de una prenda.

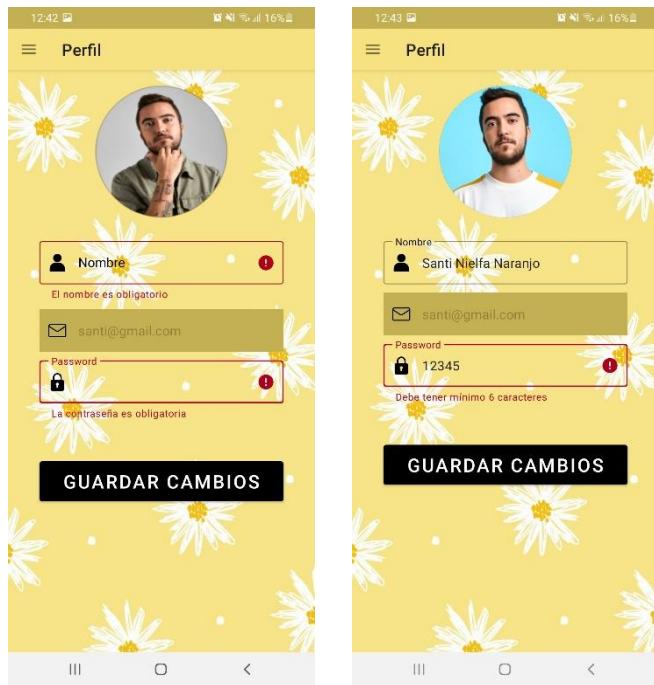
Hemos intentado generar un QR a través del listado del historial de prendas. Se nos genera correctamente un diálogo con el QR. **El resultado es el esperado.**



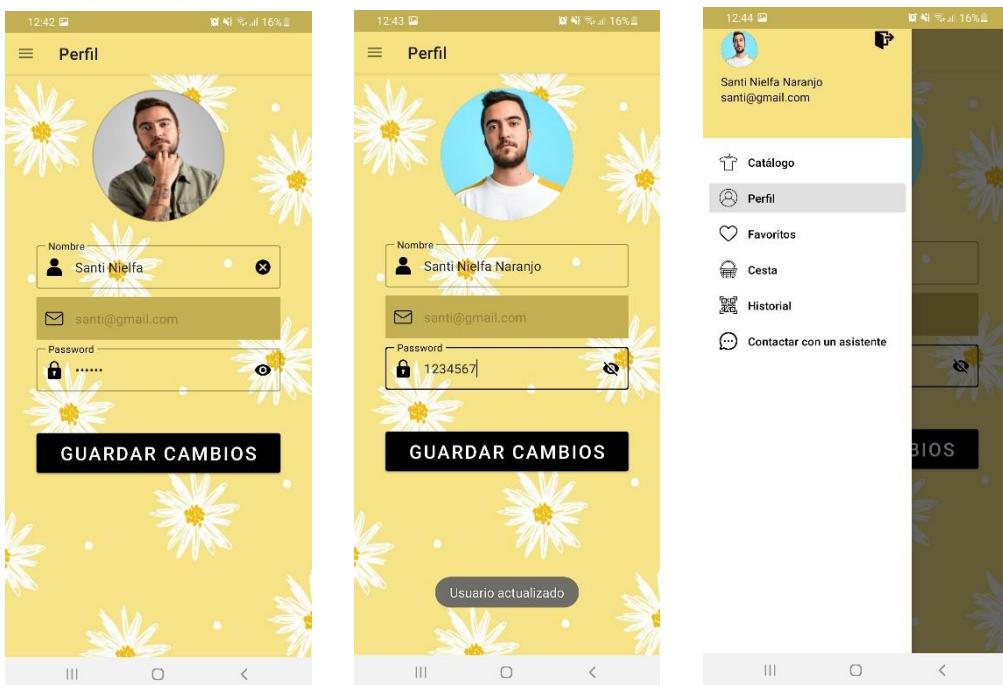
- Caso de prueba 15: Editar perfil

En esta prueba comprobaremos que podremos editar correctamente el perfil, controlando que el usuario introduzca campos nulos.

Hemos intentado editar el perfil introduciendo campos vacíos y con datos no válidos. No nos deja editarlo y nos saltan campos avisándonos. **El resultado es el esperado.**



Hemos intentado editar el perfil introduciendo campos válidos. Podemos editar el perfil correctamente. **El resultado es el esperado.**



- Caso de prueba 16: Contactar con el asistente

En esta prueba comprobaremos que podremos contactar correctamente con el asistente.

Hemos intentado contactar con el asistente mandándole un mensaje. Podremos enviar el mensaje correctamente y este se almacenará en la base de datos. **El resultado es el esperado.**



Comprobamos que los mensajes se han guardado correctamente en la base de datos.

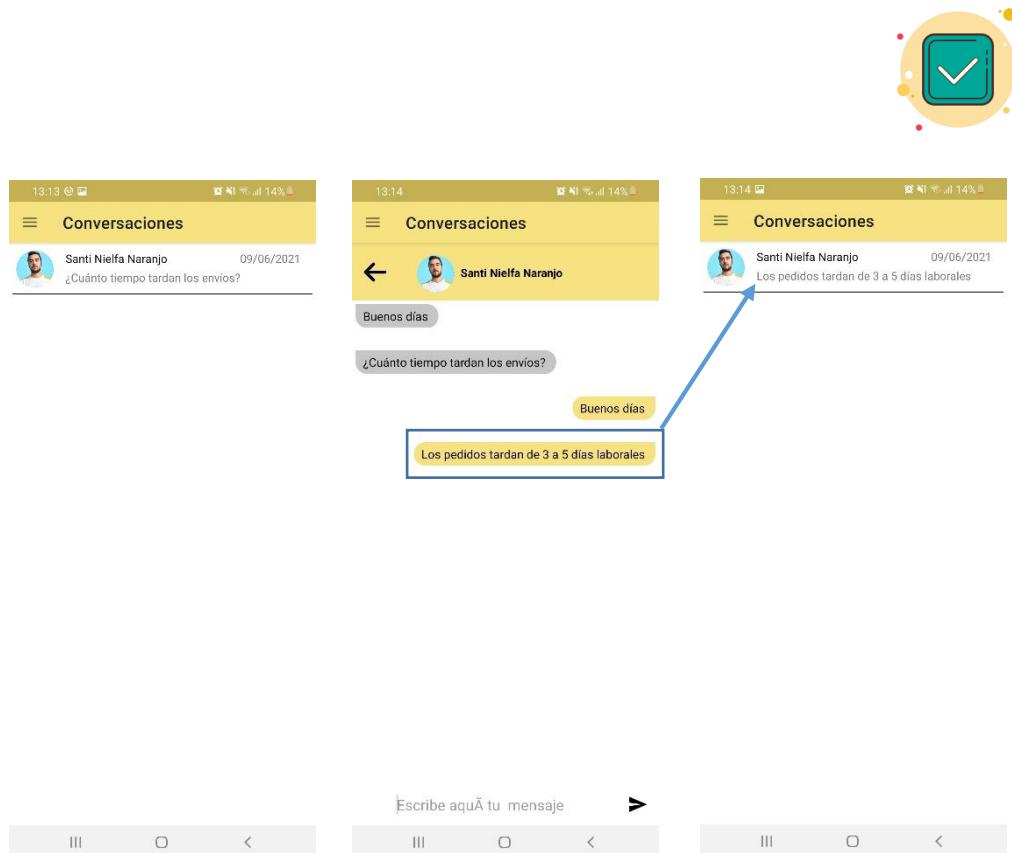
byabea-e5b76	chat	9JKQdsp0FZ8UX8pq7msK
+ Iniciar colección	+ Agregar documento	+ Iniciar colección
cesta	9JKQdsp0FZ8UX8pq7msK >	+ Agregar campo
chat >	bElp3wsLUwsujm0Mwe61	fecha: "2021-06-09T13:11:54.495"
favoritos	ufMqniD2CqmqJ4PvSwdK	idEmisor: "F6vhycPYvOB1EqLRpNqsMmVHHf2"
pedidos	yDjZi4pFISAu4ot8SkRK	idReceptor: "eD8t5GHkyceImQUmYfH1r6E6Kbr1"
prendas		mensaje: "¿Cuánto tiempo tardan los envíos?"
tipo		
usuarios		

- Caso de prueba 17: Responder chat

En esta prueba comprobaremos que al usuario administrador le ha llegado correctamente el mensaje enviado por el usuario y este puede responder sin problemas.

Por otro lado, también comprobaremos que en el listado de conversaciones se guarda correctamente el último mensaje enviado.

Hemos intentado acceder a la lista de personas que nos han hablado. Hemos visto que la conversación aparece y podemos responder el mensaje correctamente, además hemos comprobado que se muestra correctamente el último mensaje de la conversación. **El resultado es el esperado.**



- Caso de prueba 18: Devolver prenda a través del QR

En esta prueba comprobaremos que el administrador, al escanear el QR de una prenda, si todo es correcto, puede devolverla. Hay la posibilidad de que ya esté devuelta o que el pedido no haya sido enviado todavía, entonces mostrará un mensaje de error.

Hemos intentado devolver una prenda que no ha sido enviada. Hemos visto que la prenda no se puede devolver ya que todavía no ha sido enviada. **El resultado es el esperado.**

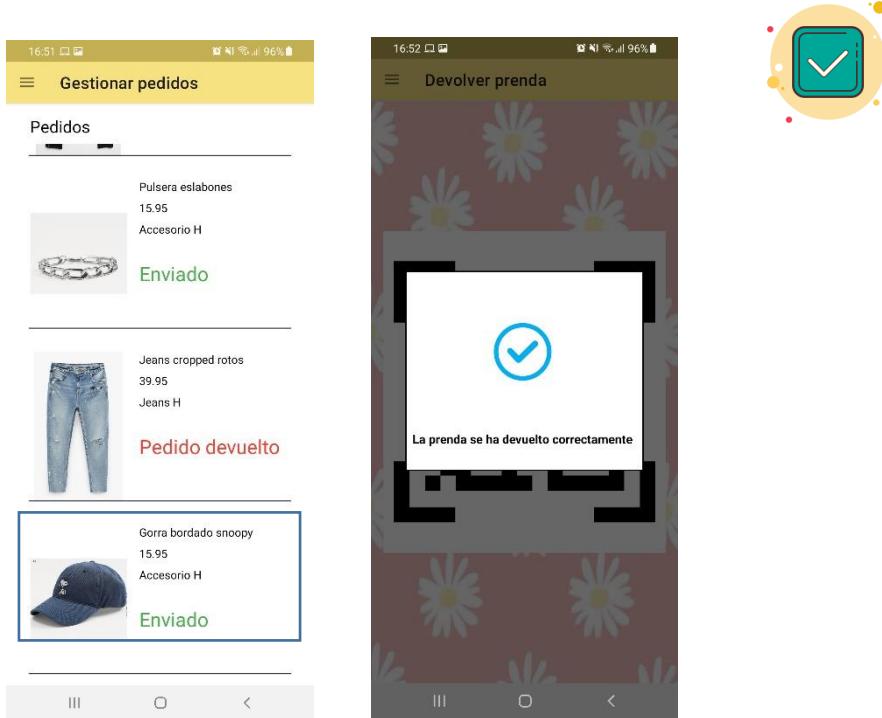




Hemos intentado devolver una prenda ya devuelta. Hemos visto que la prenda no se puede devolver ya que ya ha sido devuelta. **El resultado es el esperado.**

Two screenshots from a mobile application. The left screenshot shows the 'Gestionar pedidos' (Manage Orders) screen with three items listed: 'Pulsera eslabones' (bracelet), 'Jeans cropped rotos' (damaged jeans), and 'Gorra bordado snoopy' (Snoopy embroidered cap). The status for the bracelet is 'Enviado' (sent), while the jeans and cap are both labeled 'Pedido devuelto' (returned order). The right screenshot shows an error message: 'La prenda no se puede devolver, ya ha sido devuelta' (The item cannot be returned, it has already been returned). A yellow circular icon with a green checkmark and a small decorative trail is overlaid on the top right of the second screen.

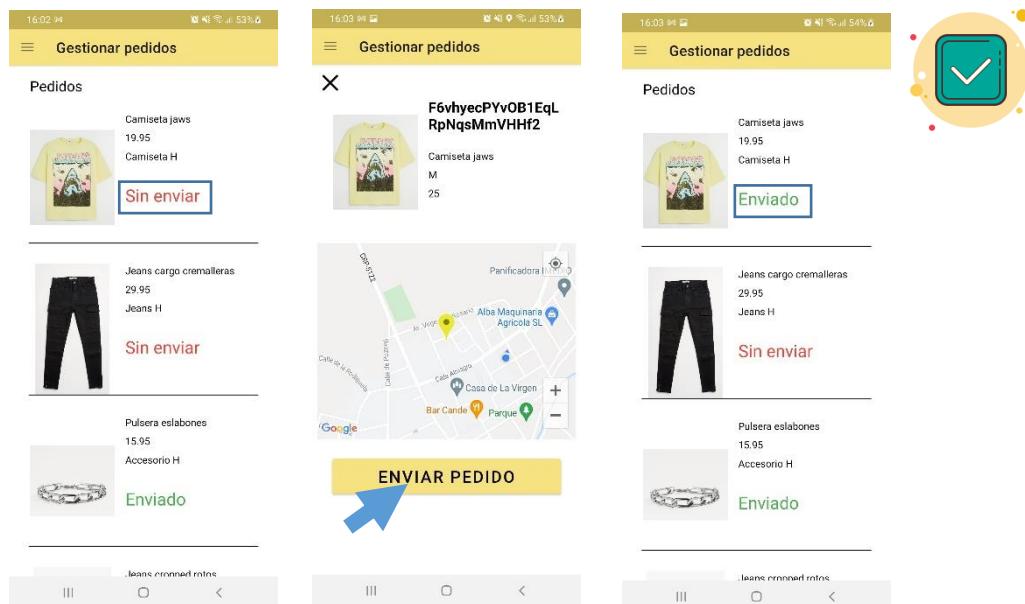
Hemos intentado devolver una prenda ya enviada y que no ha sido devuelta. Hemos visto que la prenda se devuelve correctamente y que cambia su estado en la base de datos. **El resultado es el esperado.**



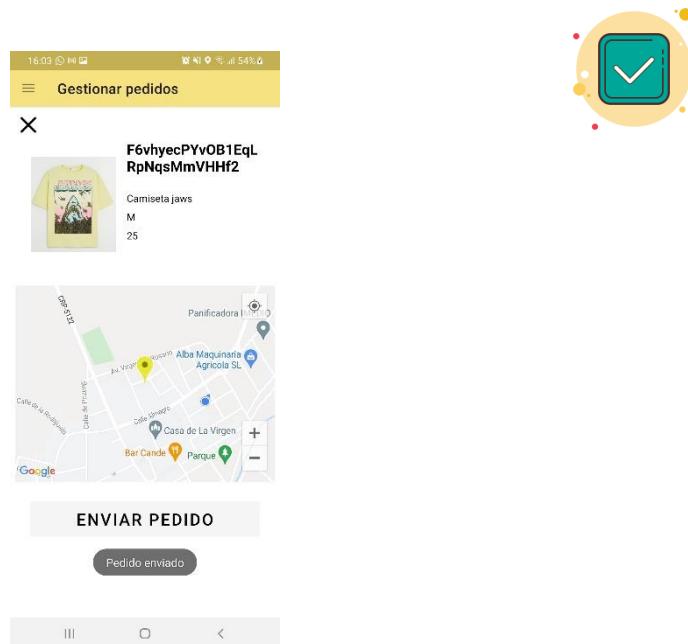
- *Caso de prueba 19: Gestionar pedidos*

En esta prueba comprobaremos que el administrador tiene acceso a los pedidos. Puede enviar los que aún no están enviados y no puede enviar los que ya han sido enviados.

Hemos intentado enviar un pedido que no ha sido enviado aún. El pedido se envía correctamente. **El resultado es el esperado.**



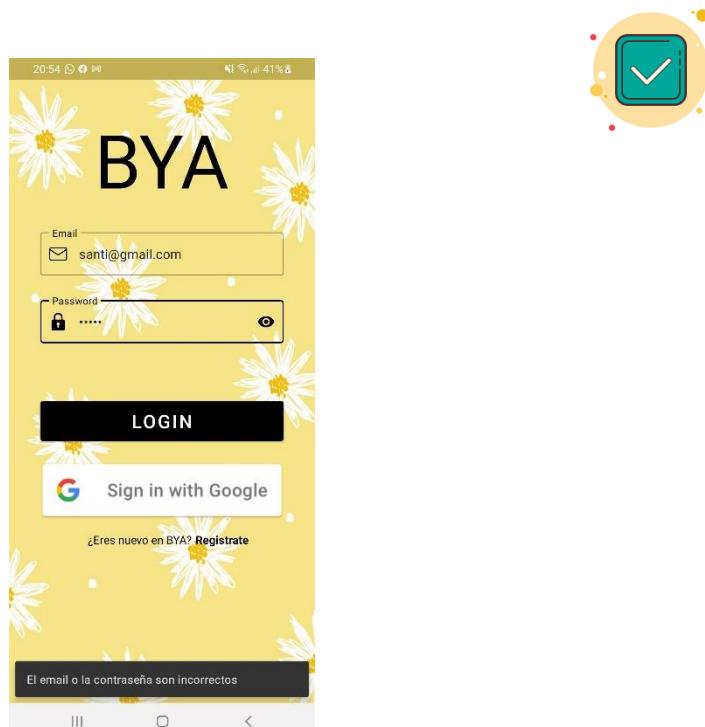
Hemos intentado enviar un pedido que ya ha sido enviado. El botón de enviar aparece deshabilitado y el pedido no se puede volver a enviar. **El resultado es el esperado.**



### Pruebas del sistema:

- *Caso de prueba 1: Seguridad*

En esta prueba comprobaremos que si el usuario no está registrado o no tiene cuenta de Google no puede acceder a la aplicación de ninguna de las maneras.



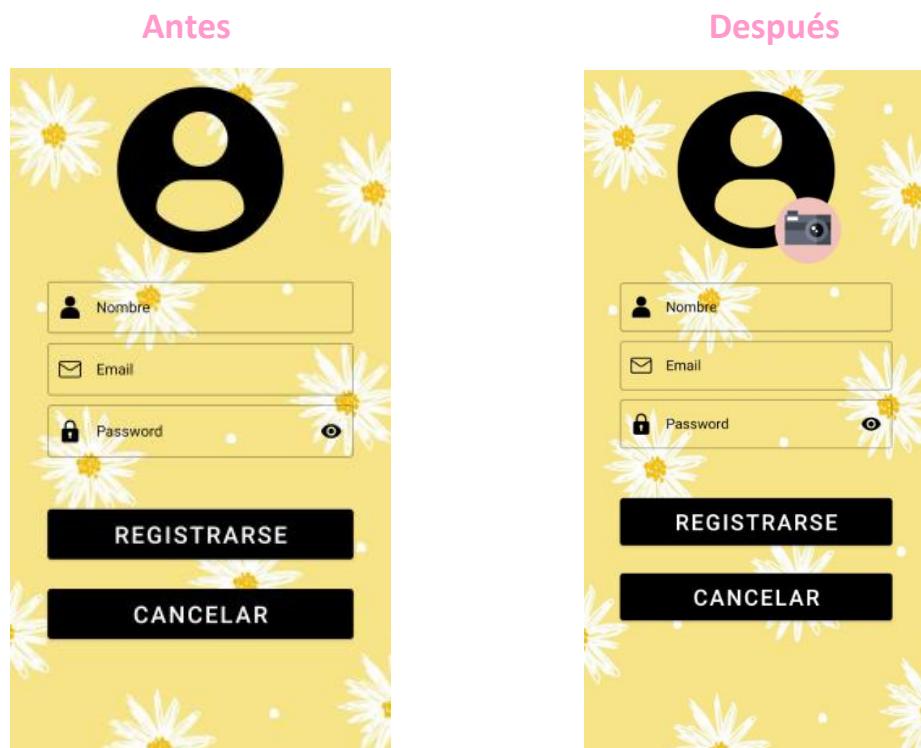
Efectivamente, **un usuario no autenticado nunca podrá acceder a la aplicación.**

- *Caso de prueba 2: Facilidad de uso*

En esta prueba comprobaremos que la aplicación es fácil de usar e intuitiva, para ello, la aplicación ha sido probada exhaustivamente por varias personas para comprobar su usabilidad.

Gracias a estas pruebas nos hemos percatado de que podríamos mejorar la funcionalidad de la aplicación añadiendo algunos botones para volver a la pantalla anterior.

Además, también nos hemos dado cuenta de que la elección de la foto no es intuitiva por lo que hemos decidido poner una imagen de una foto para que sea más fácil intuir que ahí hay que poner una foto.



## 2. Solución a problemas encontrados

Durante el transcurso del desarrollo de la aplicación no han surgido muchos problemas remarcables, los únicos problemas que podría destacar son los que respectan a la usabilidad.

Es cierto que he intentado hacer la aplicación lo más intuitiva y sencilla posible, pero siempre se escapan pequeños detalles, y en mi caso, han sido los que he explicado

anteriormente, los botones de vuelta atrás y el botón de selección de la foto y la solución ha sido rápida y sencilla.

Por otro lado, en las pruebas unitarias no he tenido casi problemas, ha salido todo bien y como me lo esperaba.

Por último, si es cierto que el problema más notable que he tenido, es que no había usado ni Firebase ni Kotlin nunca. Con respecto a Firebase, me he tenido que formar mirando muchos tutoriales y vídeos, pero por lo que respecta a Kotlin me ha resultado bastante sencillo adaptarme a este lenguaje nuevo, ya que no es muy diferente a Java.

# Lanzamiento y puesta en marcha

## 1. Aspectos relevantes del despliegue y puesta en marcha del sistema

En este apartado hablaremos sobre los aspectos más relevantes del despliegue, tanto del servidor como de nuestra aplicación.

Cuando hablamos de despliegue nos referimos a todas las actividades que hacen que un sistema de software esté disponible para su uso.

Estas actividades son:

- **Lanzamiento:** La actividad de publicación se deriva del proceso de desarrollo completado y, a veces, se clasifica como parte del proceso de desarrollo en lugar del proceso de implementación. Incluye todas las operaciones para preparar un sistema para el compilado y la transferencia a los sistemas informáticos en los que se ejecutará en producción.
- **Instalación y activación:** La instalación implica establecer algún tipo de comando, acceso directo, script o servicio para ejecutar el software. Para sistemas complejos puede implicar la configuración del sistema, posiblemente haciendo preguntas al usuario final sobre su uso previsto, o preguntándoles directamente cómo les gustaría que se configuren. Y la activación es la actividad de iniciar el componente ejecutable de software por primera vez.
- **Desactivación:** La desactivación es la actividad inversa a la activación y se refiere al cierre de cualquier componente que ya se esté ejecutando de un sistema.
- **Desinstalación:** La desinstalación es el inverso a la instalación, es la eliminación de un sistema que ya no es necesario.
- **Actualización:** Este proceso reemplaza una versión anterior de todo o parte de un sistema de software con una versión más reciente.
- **Actualización incorporada:** La automatización de los procesos para instalar actualizaciones va totalmente automática hasta iniciada y controlada por el usuario.
- **Seguimiento de las versiones:** Los sistemas de seguimiento de la versión ayudan al usuario a encontrar e instalar actualizaciones a los sistemas de software.

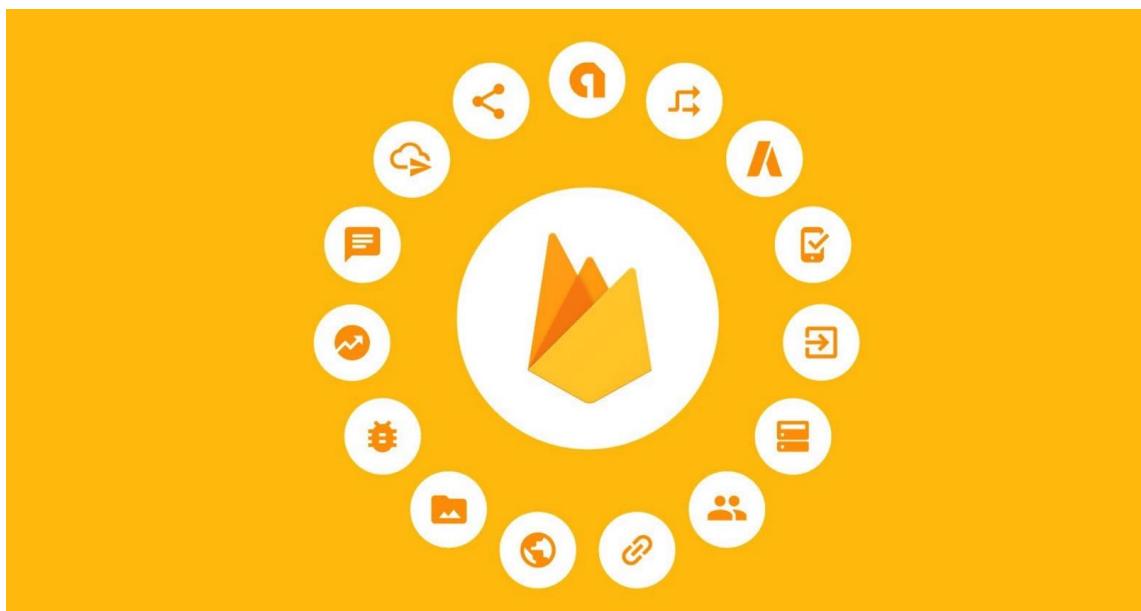
- **Adaptación:** La actividad de adaptación también es un proceso para modificar un sistema de software que se ha instalado anteriormente, se diferencia de la actualización en que las adaptaciones son iniciadas por eventos locales, mientras que la actualización es una consecuencia de la disponibilidad de la nueva versión.

## Despliegue de B&A

En el caso del lanzamiento de B&A, a la hora de desplegar, podemos dividir el lanzamiento en dos partes:

- **La parte del servidor (Firebase).**

En lanzamiento en la parte del servidor (ya que usamos Firebase) es bastante sencilla, no porque su lanzamiento sea fácil, sino porque lo hace Firebase por nosotros, **desde el momento de su creación, Firebase ya se encuentra desplegado**, nosotros lo único que tenemos que hacer es configurarlo y meternos en la consola ya que el único requisito que tenemos que cumplir es tener una cuenta de Google.



La configuración de Firebase es muy sencilla, para empezar, le tenemos que dar un nombre a nuestro proyecto.

X Crear un proyecto(paso 1 de 3)

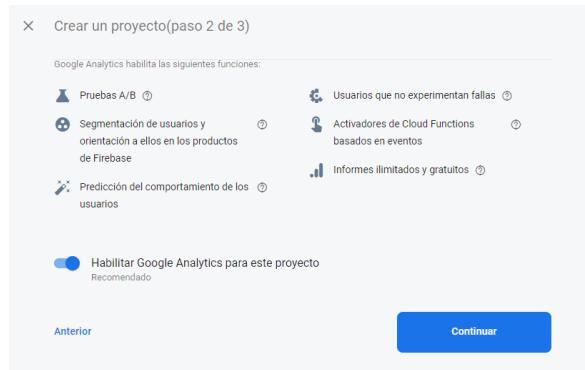
Comencemos con el nombre de tu proyecto<sup>®</sup>

Nombre del proyecto  
**BYABEA**

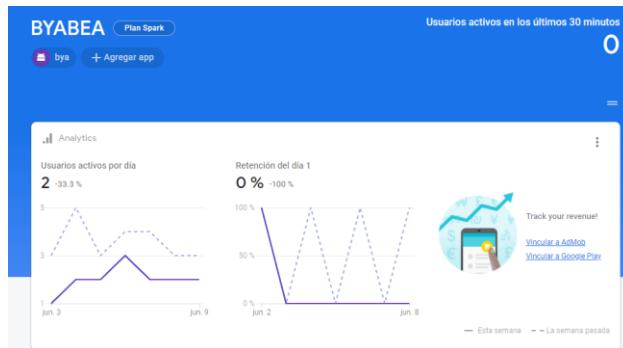
byabea-bb34a

Continuar

Una vez tenemos el nombre del proyecto, el siguiente paso será habilitar las analíticas de Google, requisito indispensable si queremos ver los gráficos y las estadísticas de nuestra aplicación.



Por último, tendremos que crear el proyecto y sincronizarlo con nuestro proyecto de Android. Una vez hecho esto ya tendremos toda la configuración del servidor y las analíticas listas para usar.



Ya tenemos Firebase sincronizado con el proyecto, ahora tenemos que implementar y configurar todas las funciones que queramos añadir a nuestra configuración de Firebase. En nuestro caso, le vamos a añadir la autenticación, Firestore y Storage.

En el caso de la autenticación, solo tendremos que habilitar esta función en el menú lateral de la izquierda. Una vez habilitada esta función, tendremos que habilitar los métodos de autenticación que vamos a querer usar, en nuestro caso, el de correo/contraseña y Gmail.

Proveedores de acceso	
Proveedor	Estado
Correo electrónico/contraseña	Habilitada
Teléfono	Inhabilitado
Google	Habilitada
Play Juegos	Inhabilitado

The screenshot shows the Firebase Authentication console under the 'Users' tab. It displays a table of user information:

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
maria@gmail.com	Email	7 jun. 2021	7 jun. 2021	5n3cwRCHL7g792nZ0f72Dft4R8e2
santi@gmail.com	Email	8 jun. 2021	10 jun. 2021	F6vhyecPYvOB7EqLRpNqzHmVH...
antonio@gmail.com	Email	8 jun. 2021	8 jun. 2021	FOc7f5dY0zPUpeg73z4bXedYjuJ2
kevin@gmail.com	Email	7 jun. 2021	7 jun. 2021	McNKD147y9bYClF63F9wShwXy...
josefa@yahoo.es	Email	6 jun. 2021	7 jun. 2021	U4sReETreAd0wo6cbm37D4nRjs2
santiagoseffa25@gmail.c...	Google	20 may. 2021	8 jun. 2021	UINV1vm6qsfvfnV0cL3tAcPfA72

También tendremos que habilitar la base de datos de firestore, para ello, nos vamos a su opción en el menú de la izquierda y pulsamos en comenzar, tendremos que elegir en que región queremos alojar la base de datos, una vez hecho esto, ya podremos usar esta funcionalidad.

The screenshot shows the Firestore database interface. On the left, there's a sidebar with a tree view of collections: 'cesta' (selected), 'chat', 'favoritos', 'pedidos', 'prendas', 'tipo', and 'usuarios'. The main area shows a document in the 'cesta' collection with the ID '0eb93f20-bf45-4831-b44e-b0f983b2a18f'. The document contains the following data:

```

idCesta: "0eb93f20-bf45-4831-b44e-b0f983b2a18f"
idPrenda: "af6e3e46-b535-4d92-9faa-65e9407b66a3"
idUsuario: "ZDXIRj6mjKQS7U273zhOYpDgDcZ2"
talla: "L"

```

Por último, la última funcionalidad que vamos a añadir va a ser el storage, para activarlo, será la misma mecánica que con firestore, nos vamos a su opción en el menú lateral de la izquierda, le damos a comenzar, seleccionamos un alojamiento para esta base de datos y ya estará listo para usar.

The screenshot shows the Google Cloud Storage interface. At the top, it says 'gs://byabea-e5b76.appspot.com'. Below that, there are tabs for 'Files', 'Rules', and 'Usage'. A note says 'Protege tus recursos de Storage contra los abusos, como fraudes de facturación o suplantación de identidad.' There is a button to 'Configurar la Verificación de aplicaciones'. The main area shows a table of files:

Nombre	Tamaño	Tipo	Modificación más reciente
fotosPrendas/	—	Carpeta	—
fotosUsuarios/	—	Carpeta	—

Una vez hayamos configurado estas opciones, **Firebase estará lanzado, configurado y listo para su uso.**

- **La parte del cliente**

Para desplegar la parte del cliente, tenemos que subirla al PlayStore para que esté a disposición de todos los usuarios del mercado.



Para subirla tenemos que seguir los siguientes pasos.

- **Acceder a Google Play Developer Console**



Para ello, tenemos que registrarnos en Google Play, aceptar el acuerdo de distribución para desarrolladores, pagar la cuota de registro, que será de un pago único de 25 \$ que, convertidos a euros, serán 20.57 €.

Una vez realizado el pago, tendremos que llenar información básica sobre nuestra cuenta, como el nombre de desarrollador.

- **Añadir la aplicación**

A continuación, ya podremos subir la aplicación a la cuenta, para ello, tenemos que subir la APK que anteriormente hemos generado, tendremos que seguir estos pasos:

- Seleccionar en el menú la opción de añadir una nueva aplicación.
- Elegir un idioma predeterminado.
- Darle un nombre a la aplicación.
- Subir la APK

- **Definir la App**

Una vez subidos los archivos APK debemos pasar a cumplimentar la ficha de la aplicación en Google Play. Aquí debemos indicar, desde la descripción completa de la aplicación, hasta el texto de promoción, el ícono o los pantallazos que vamos a mostrar a los usuarios interesados.

- **Establecer el precio y la distribución**

Finalmente, ahora tenemos que decidir si nuestra aplicación será gratuita o de pago, si elegimos de pago, para poder cobrar por el producto, debemos disponer de una cuenta de pago válida, estas cuentas se consiguen a través de un acuerdo independiente con un procesador de pagos.

En mi caso B&A será gratuita.

Hecho esto, **ya tendremos la aplicación del cliente desplegada y lista para usar.**

## 2. Manual de uso

Un manual de uso es un documento que busca brindar la existencia a los sujetos que usan una aplicación. En este manual de usuario hemos intentado apelar a un lenguaje ameno y simple, para llegar a la mayor cantidad posible de receptores.

Haciendo clic en esta imagen, tendremos acceso al manual de uso de la aplicación de B&A.



# Valoración y conclusiones

Desde el momento en el que entre en primero, ya le tenía cierto respeto al proyecto final, porque sabía que al fin y al cabo este momento llegaría.

Han sido unos meses duros, ya que nunca había realizado una documentación tan completa y extensa, y la inexperiencia me ha hecho dudar en algún momento y plantearme todo tipo de cosas. No sabía si iba a llegar a la fecha, porque en muchos momentos he pensado que me había pasado con las funcionalidades de la aplicación, además también me he tenido que formar en Kotlin y Firebase.

Las primeras entregas fueron de documentación, me costó ir cogiendo el ritmillo, pero finalmente creo que hasta le cogí el gusto a ir documentando el proyecto, además me ha venido muy bien a la hora de desarrollar la aplicación ya que cuando llegó este momento fui con una idea bastante clara sobre las cosas que tenía que hacer. Acabé quemada con el prototipado, pero viéndolo ahora, es cierto que después ahorra muchísimo tiempo y es un mal que hay que pasar. Después pasé al desarrollo, momento de plasmar la documentación y todo lo aprendido de Kotlin y Firebase, esta etapa la verdad es que fue de altibajos, influía mucho si me salían las cosas o no, un día estaba eufórica y al siguiente desanimada. Finalmente llegó la etapa de volver a la documentación, aquí la verdad es que volví un poco perdida porque hacía mucho tiempo que no tocaba esta documentación, pero me costó poco volver a la rutina.

Durante el transcurso de estos años, he aprendido mucho de todas y cada una de las asignaturas que he tenido, y por ello he intentado plasmarlo todo en este proyecto. Por ejemplo, la asignatura de PSP me ha venido muy bien a la hora de planificar el proyecto ya que gracias a sus trabajos y a sus exámenes he aprendido a pensar como una verdadera programadora, para el prototipado he recordado todo lo aprendido en desarrollo de interfaces, todo lo que José Alberto nos enseñaba y todas esas prácticas con sus complejos diseños, también me ha servido de mucha ayuda acceso a datos a la hora de planificar y desarrollar la base de datos, obviamente programación multimedia y dispositivos móviles también me ha servido de mucha ayuda, ya que el cliente de la aplicación está en su totalidad desarrollado en Android, a pesar de no haber cursado esta asignatura este año, lo aprendido el curso pasado ha sido más que suficiente para tener una base firme con la que poder realizar el proyecto. Hay asignaturas que no he nombrado, pero os aseguro que también me han ayudado mucho a formarme y están presentes en este proyecto.

Finalmente quería destacar que todo esto ha sido una experiencia muy positiva para mí, y que sin lugar a dudas me va a ayudar bastante en mi futuro profesional, escribo esto con muchos nervios y sin saber la nota final, pero pase lo que pase lo escribo orgullosa porque sé que he dado todo de mí.

# Bibliografía y recursos utilizados

Se conoce como bibliografía a la relación o lista de un conjunto de libros o escritos utilizados (en este caso, recursos de internet) como material de consulta o soporte documental para la investigación y la elaboración de un trabajo escrito o una monografía.

Las bibliografías otorgan validez y rigurosidad a los trabajos de investigación monográficos, académicos, científicos o eruditos, pues demuestran que su autor se preocupó por rastrear fuentes que pudieran sentar las bases de su investigación, así como orientarla o aportarle valor.

Suele ubicarse al final del texto y su objetivo es presentar el soporte documental con el que contó el trabajo, mostrando el repertorio de textos consultados.

A continuación, voy a mostrar los enlaces toda la bibliografía utilizadas en este proyecto.

- [https://es.wikipedia.org/wiki/Requisito\\_funcional](https://es.wikipedia.org/wiki/Requisito_funcional)
- [https://es.wikipedia.org/wiki/Requisito\\_no\\_funcional](https://es.wikipedia.org/wiki/Requisito_no_funcional)
- <https://www.infor.uva.es/~mlaguna/is1/apuntes/2-requisitos.pdf>
- <https://www.obsbusiness.school/blog/como-hacer-un-estudio-de-mercado-en-4-pasos>
- <https://mglobalmarketing.es/blog/marketing-para-empresas-como-se-define-el-target/#La edad un aspecto clave en la definicion del target>
- <https://www.infoautonomos.com/plan-de-negocio/analisis-dafó/#que-es-dafó>
- <https://instintobinario.com/diagrama-de-casos-de-uso/>
- <https://docs.github.com/es/github/managing-your-work-on-github/about-project-boards>
- <https://aws.amazon.com/es/devops/continuous-integration/#:~:text=La%20integraci%C3%B3n%20continua%20es%20una,ejecutan%20versiones%20y%20pruebas%20autom%C3%A1ticas.>
- <https://gfourmis.co/gitflow-sin-morir-en-el-intento/>
- <https://nvie.com/posts/a-successful-git-branching-model/>
- <https://guides.github.com/features/wikis/>
- <https://www.gladysbegnedji.com/estimar-los-costos-del-proyecto/>
- <https://medium.com/administrador-de-proyectos/estimar-los-costos-5da1e1e44b35#:~:text=Estimar%20los%20costos%20consiste%20en%20desarrollar%20una%20estimaci%C3%B3n%20aproximada%20de,completar%20las%20actividades%20del%20proyecto.&text=No%20olvidar%20los%20costos%20relacionados,Tiempo%20del%20director%20de%20proyecto>
- <https://www.recursosenprojectmanagement.com/duracion-de-una-tarea/>
- <https://www.recursosenprojectmanagement.com/estimacion-del-coste-del-proyecto/>
- <https://thedigitalprojectmanager.com/es/guia-estimacion-presupuesto-costos/#basics>
- <https://app.estimate.com/#/project/-MYozYKVJ-kbF0EGUGJv>
- <https://www.semantic-systems.com/semantic-noticias/articulos-tecnologicos/gestion-de-riesgos-en-proyectos-de-implantacion-de-software/>
- <https://www.unir.net/ingenieria/revista/riesgos-laborales-informatica/>
- <https://developer.android.com/guide/topics/ui?hl=es>

- <https://www.efectodigital.online/single-post/2018/04/18/dise%C3%B1o-de-interfaz-de-usuario-ui>
- <https://wiki.uqbar.org/wiki/articles/diagrama-de-clases.html>
- <https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml>
- [https://www.ecured.cu/Diagrama de Clase](https://www.ecured.cu/Diagrama_de_Clase)
- <http://elvex.ugr.es/decsai/java/pdf/3c-relaciones.pdf>
- <https://ocw.unican.es/pluginfile.php/914/course/section/1022/Sistemas-Informacion-5.pdf>
- <https://aws.amazon.com/es/nosql/>
- [https://es.wikipedia.org/wiki/Arquitectura\\_de\\_software](https://es.wikipedia.org/wiki/Arquitectura_de_software)
- <http://www.itlp.edu.mx/web/java/Tutorial%20de%20Java/Intro/carac.html>
- [https://es.wikipedia.org/wiki/Kotlin\\_\(lenguaje\\_de\\_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Kotlin_(lenguaje_de_programaci%C3%B3n))
- <https://codigoonclick.com/caracteristicas-de-kotlin-para-android/>
- <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html>
- <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-flutter/>
- <https://www.programaenlinea.net/flutter-desarrollo-movil/>
- <https://softwarecrafters.io/xamarin/xamarin-forms-apps-nativas-introduccion>
- <https://www.oscarblancarteblog.com/2018/07/17/spring-boot-relacion-los-microservicios/>
- <https://www.ecured.cu/Laravel>
- <https://openwebinars.net/blog/que-es-laravel-caracteristicas-y-ventajas/>
- <https://www.nextu.com/blog/6-cosas-que-debes-saber-de-amazon-web-services-y-sus-beneficios/>
- <https://aws.amazon.com/es/what-is-aws/>
- <http://www.carlospes.com/minidiccionario/pruebas.php>
- <https://www.yeeply.com/blog/que-son-pruebas-unitarias/>
- <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/unitarias>
- <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/integracion>
- <https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/pruebas/sistema>
- <https://eugeniacasabona.medium.com/pruebas-con-usuarios-1-qu%C3%A9-cu%C3%A1ndo-y-para-qu%C3%A9-testteamos-7c3a89b4b5e7>
- [https://es.wikipedia.org/wiki/Despliegue\\_de\\_software#cite\\_note-1](https://es.wikipedia.org/wiki/Despliegue_de_software#cite_note-1)
- <https://www.antevenio.com/blog/2016/09/como-colgar-una-aplicacion-en-google-play/>
- <https://www.significados.com/bibliografia/>