UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Catálogo interactivo desarrollado en Java/Android

MEMORIA PRESENTADA POR:

Francisco Cerdán Guerrero

TUTOR:

Oscar Vega Gisbert

GRADO DE INGENIERÍA INFORMÁTICA

Convocatoria de defensa: Septiembre 2017

Resumen

El presente documento ha sido realizado por Francisco Cerdán Guerrero, estudiante del Grado en Ingeniería Informática de la Escuela Politécnica Superior de Alcoy. En esta memoria se recorrerá el proceso de realización del proyecto, desde su origen hasta su etapa final de mantenimiento, los diferentes objetivos que se han ido cumpliendo y las decisiones tomadas en cada uno de los problemas que se encontraron durante el mismo.

El trabajo consiste en el desarrollo de una aplicación para dispositivos Android, programada en lenguaje JAVA, la cual se basa en un catálogo interactivo, enfocada al uso por parte de los comerciales de los que dispone una empresa, permitiendo así agilizar su labor haciendo el proceso más sencillo tanto como para el comercial como para el cliente.

Abstract

This document has been written by Francisco Cerdán Guerrero, student of the Computer Engineering Degree at Polytechnics Superior School of Alcoy. In this report, the process of the development will be traced, from its beginning to its final stage of maintenance, also the different objectives that have been fulfilled and the decisions taken in each problem encountered.

The work it's the development of an JAVA application for Android devices, which consists of an interactive catalog, focused on de use of a company commercials, allowing to speed up their work by making the simpler process as much for the commercial as for the client.

Resum

El següent document ha sigut realitzat per Francisco Cerdán Guerrero, estudiant del Grau en Enginyeria Informàtica a l'Escola Politècnica Superior d'Alcoi. En aquesta memoria es recorrerà el procés de realització del projecte, des del seu origen fins a la seva etapa final de manteniment, els diferents objectius que s'han anat complint i les decisions preses en cada un dels problemes que es van trovar durant el mateix.

El treball es tracta del desenvolupament d'una aplicació per a dispositius Android, programada en llenguatje JAVA, la cual consisteix en un catàleg interractiu, enfocat al l'us per part dels comercials dels que disposa una empresa, aconseguint així agilitzar la seva tasca fent el procés mes senzill tant com per al comercial com per al client.

Índice

1. Introducción

1.1 ANTECEDENTES DEL PROYECTO

Finalizando el grado en ingeniería informática, tanto por cuenta propia, como mediante las prácticas en empresa, se me brindó la oportunidad de desarrollar programas a un nivel más profesional, enfocados a resolver problemas reales, por lo que pude descubrir el gran abanico de oportunidades que se desplegaba ante él: Las aplicaciones móviles

Hoy en día el uso de los teléfonos inteligentes o “smartphones”, como son conocidos, es algo que se ha generalizado hasta que, salvo alguna excepción, la totalidad de la población utiliza. Estos terminales no tienen nada que ver con antiguos teléfonos móviles, son pequeños ordenadores capaces de realizar cualquier clase de trabajo, equipados con una interesante variedad de sensores como cámaras, GPS, acelerómetros… lo que se traduce una capacidad de desarrollar herramientas casi ilimitada. Sabiendo esto, la posibilidad de desarrollar una aplicación a medida y personalizada para una empresa y función en particular es muy interesante, ya que el teléfono se convierte en un pequeño ayudante que facilitará y hará más eficientes tareas rutinarias de trabajo.

1.2 OBJETIVOS

Teniendo en cuenta las ventajas de las aplicaciones móviles se decidió crear una aplicación que consistiese en un catálogo interactivo como idea principal, destinada a agilizar la tarea de los comerciales de una empresa. Posteriormente, tras conocer con más detalle el proceso de trabajo de un comercial, se le han incorporado más funciones con el fin de cubrir el máximo de necesidades posibles, evitando así que el comercial necesite más medios para poder realizar una transacción con un cliente, englobando todas las fases de la operación en un solo dispositivo.

A todo esto, se le añaden otros requisitos indispensables que ha de tener la aplicación.

- **Facilidad de uso:** No todos los comerciales están igual de familiarizados con las nuevas tecnologías, por lo que uno de los puntos en los que se ha concentrado la aplicación ha sido en tener un uso intuitivo, con menús sencillos, procurando realizar el trabajo en la menor cantidad de pasos posible.
- **Compatibilidad:** La aplicación se ha diseñado también con el objetivo de que funcione adecuadamente en la franja más amplia posible de dispositivos posibles, para ello se han debido implementar características para que se pueda usar en terminales con recursos muy limitados y se han cumplido requisitos para los más modernos.

- **Robustez:** Para garantizar que la aplicación esté siempre disponible, también se ha analizado el uso de la misma y se han tomado medidas para garantizar la mayor disponibilidad posible.

2. Metodología

2.1 ANÁLISIS DEL PROCESO COMERCIAL

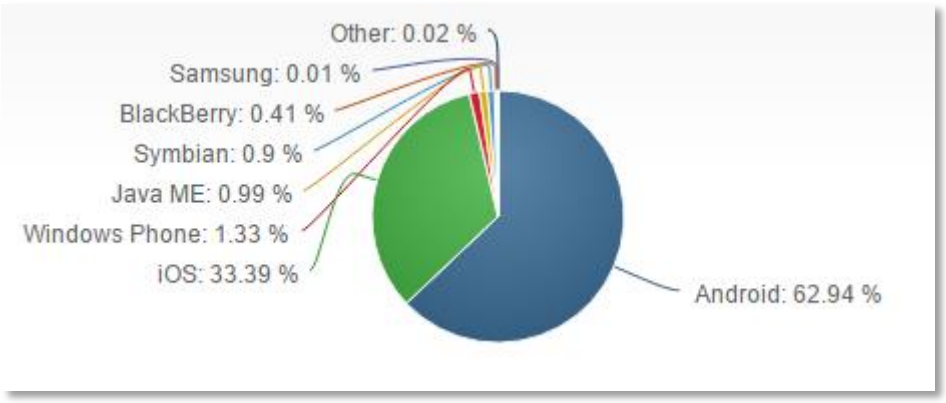
Una vez con la idea del programa preconcebida, se analizó el proceso que se pretende agilizar con la aplicación, las diferentes funciones que debería tener y, por último, los posibles problemas que deberían surgir durante el desarrollo. El proceso es el siguiente:

1. **Presentación al cliente de los diversos productos:** Se debe implementar un sistema de galería/catálogo que sea atractivo, dinámico y vistoso para que el cliente tenga una buena impresión. También se ha de implementar la función de creación de pedido desde el mismo sistema, para que el cliente pueda decidir qué producto desea añadir al pedido desde el mismo instante en el que lo ve.
2. **Revisión del pedido por parte del cliente:** En cualquier momento desde la creación del pedido, hasta el paso final donde se cierra, el cliente debe que ser capaz de poder editar el contenido, ya que no todos los presupuestos son iguales y muchas veces se necesitan realizar ajustes en un último momento.
3. **Toma de datos del cliente:** Para formalizar el pedido, serán necesarios la totalidad de datos personales del cliente, evitando de este modo la falta de información. Se tuvieron especialmente en cuenta las formas de pago más utilizadas.
4. **Finalización del pedido:** Cuando se finaliza el pedido, es importante que todo quede archivado y exista la posibilidad de volver a generar las hojas de pedido, o poder implementar documentos reportando el listado de pedidos realizados, la suma de su valor, otro listado de clientes existentes, etc.

Pese a parecer un proceso muy sencillo y lineal, se comprobó en muchas ocasiones que la aplicación necesitaba una alta disponibilidad de los datos y funciones, por lo que en cualquier momento era necesario acceder a cualquiera de las tres primeras partes del proceso citadas anteriormente. Cada cliente dispone de un tiempo disponible diferente para presentarle el producto, algunos clientes son conocidos, otros son nuevos… por lo que pese a ser el mismo procedimiento, puede variar el orden de realización de los pasos.

2.2 MEDIOS ESCOGIDOS

La aplicación se desarrolló para sistemas Android, debido a que este sistema desde hace mucho tiempo tiene la mayor cuota de mercado de dispositivos móviles. Además, es usado todo tipo de gamas de dispositivos, por lo que se pudo facilitar nuevos terminales Android a los comerciales que no disponían de él, y escoger la gama de precios más conveniente.



1. Ventas de dispositivos por Sistema Operativo de Enero a abril de 2017

Como se aprecia en la miniatura, existe también otro sistema (iOS) con una muy buena cuota de mercado, pero se descartó debido a que implicaría un sobrecoste tanto en los terminales, considerados todos de gama alta, como en la formación para desarrollar el programa.

2.3 LENGUAJES ESCOGIDOS

Una vez decidida la plataforma sobre la que se va a trabajar, es conveniente elegir el lenguaje de programación adecuado. En el caso de Android, se despliega un amplio abanico de posibilidades donde elegir. Existen soluciones de pago, como puede ser Xamarin, de Microsoft, que permite desarrollar aplicaciones en C#, o Corona SDK, que lo permite en LUA, entre otros. Para evitar sobrecostes y partiendo de que se tienen conocimientos anteriores, se optó por **JAVA**, ya que ofrece la opción de desarrollar de manera gratuita.

JAVA es un lenguaje multiplataforma, el cual suele encabezar las listas de los lenguajes más usados. Por otra parte, es el lenguaje empleado a lo largo de los cuatro años de formación en el grado en ingeniería informática, por lo que resulta el idóneo para trabajar en este caso, además

tiene una curva de aprendizaje relativamente fácil que dará la posibilidad de aprender a medida que se solucionan problemas.

2.4 ENTORNO DE DESARROLLO

Como entorno de desarrollo se escogió Android Studio, herramienta desarrollada por la compañía JetBrains, creadora también de otros excelentes entornos de desarrollo. Android Studio se trata de la herramienta recomendada por la misma documentación Android cuando se desea desarrollar alguna aplicación, trabaja con el lenguaje JAVA, dispone de una enorme comunidad de usuarios que ofrecen soporte, y es actualizado muy frecuentemente, por lo que lo hace un entorno puntero para desarrollar aplicaciones Android y el ideal para el proyecto.

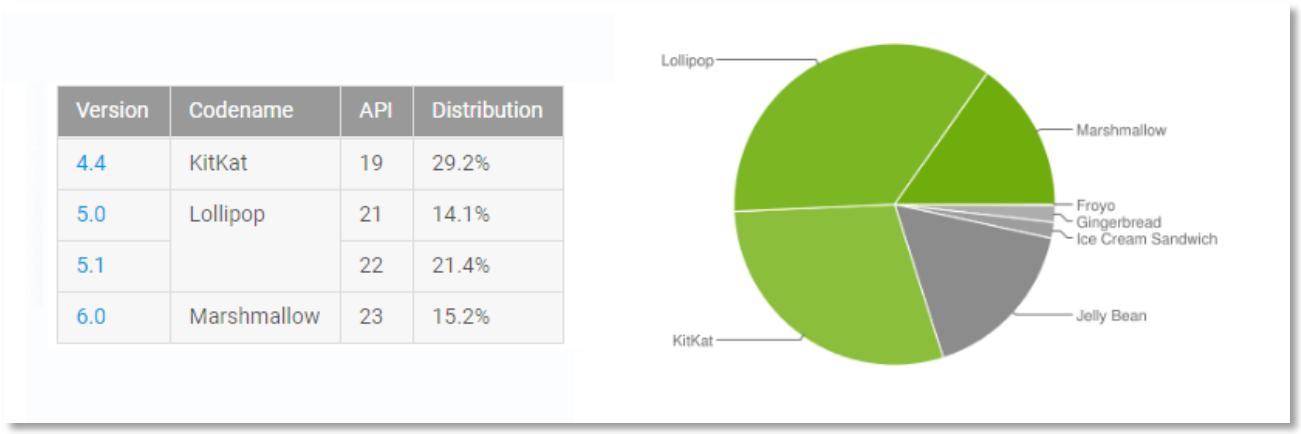
Entre otras funciones, dispone de un emulador integrado, el cual permite ir comprobando el desarrollo de la aplicación sin la necesidad de tener un dispositivo Android al alcance, además de que se puede simular cualquier dispositivo, sea un móvil o tableta, con diferentes características, ya sea el tamaño de la pantalla, la memoria de la que dispone, o, muy importante, la versión de Android en la que se desea ejecutar la aplicación. Esta función ha sido de vital importancia a la hora de preparar la aplicación para un gran rango de versiones, las cuales se analizarán en el siguiente punto.

Otra característica muy interesante es que dispone de versión tanto de versión Linux como Windows, facilitando de esta manera el trabajar en la aplicación en diferentes equipos.

2.5 VERSIONES DE ANDROID

Al haber una cantidad tan extensa de terminales, existe con ella una variación muy grande de versiones, debido a que, limitados por las características, muchos dejan de recibir actualizaciones y quedan estancados en una versión. Es por ello que no se puede limitar el proyecto a una versión en concreto, ya que en algunas ocasiones surgirían problemas de compatibilidad.

Para solventar este problema se comprobó cuáles son las versiones más utilizadas en la actualidad, las cuales van desde la **4.4 KitKat** hasta la **6.0 Marshmallow**, teniendo el rango entre estas un **79.9%** de la cuota de mercado.



Version	Codename	API	Distribution
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%

2. Gráfica que muestra las versiones más utilizadas de Android (en verde) en 2017

Si se compara la versión más antigua con la más reciente, estéticamente hay variaciones muy notables, por lo se ha decidido mantener la estética de la versión 4 para evitar problemas de compatibilidad, ya que las versiones más recientes siguen aceptando esos estilos. Esto limita algunas funciones más modernas que hacen la aplicación visualmente más vistosa, **pero se gana compatibilidad,** que es uno de los objetivos que se buscan.

Respecto a las versiones más nuevas, el único problema que surgió consiste en un cambio en las **políticas de permisos** de Android. Estas políticas son las que deciden qué acciones puede realizar la aplicación sobre el teléfono, y se han de aceptar aquellas necesarias para su correcto funcionamiento. Es el mecanismo de Android para advertir de qué puede hacer la aplicación sobre

el teléfono, sensores, o datos personales, entre otros. En este caso, la aplicación solo manipula el almacenamiento SD. En las versiones inferiores a Android 6, estos permisos se aceptaban durante la instalación de la aplicación, mientras que en Android 6 y posteriores se han de aceptar en tiempo de ejecución, por lo que se tuvo que realizar ese ajuste para poder generar PDF y guardarlo en la memoria (es la finalidad de dicho permiso).

2.6 BASE DE DATOS

Debido a la necesidad de una alta disponibilidad, se decide trabajar sobre una base de datos **SQLite**. Este tipo de base de datos trabaja sobre un archivo y no requiere de conexión a ningún tipo de red, por lo que cada dispositivo dispone de la suya propia accesible en cualquier momento.

La particularidad de esta base de datos es su ventaja y desventaja al mismo tiempo: Al ser un único archivo, no se requiere conexión a ningún tipo de red o servidor, por lo que es accesible en cualquier momento desde el dispositivo. Por otro lado, al trabajar sobre un archivo, es imposible implementar funciones para controlar los datos que aparecen en cada terminal, como por ejemplo consultar la cantidad de artículos que quedan en stock, o gestionar los artículos del catálogo remotamente. Analizando pros y contras, se llegó a la conclusión de que era preferible para el comercial disponer de la base en todo momento, por lo que se decidió usar SQLite y para actualizar la base, se le enviarían actualizaciones periódicas del programa con la base de datos actualizada que él mismo podría instalar.

Para administrar la base de datos, se ha empleado el programa **SQLite Studio**, el cual es gratuito y de código abierto bajo la licencia GPLv3.

2.7 PLANIFICACIÓN POR OBJETIVOS

La planificación del proceso de desarrollo se ha realizado en su totalidad **por objetivos**, estableciendo un objetivo inicial y añadiendo otros nuevos que surgían a medida que se desarrollaba el trabajo. Cada objetivo se añadía a una lista y se le asignaba una prioridad, para en el momento en el que se completaba uno, o se llegaba a un bloqueo, había otro objetivo disponible con el cual empezar a trabajar (en ocasiones cuando se emplea un gran esfuerzo durante mucho tiempo en un punto, es conveniente cambiar de tarea y despejarse para retomarlo con las ideas más claras)

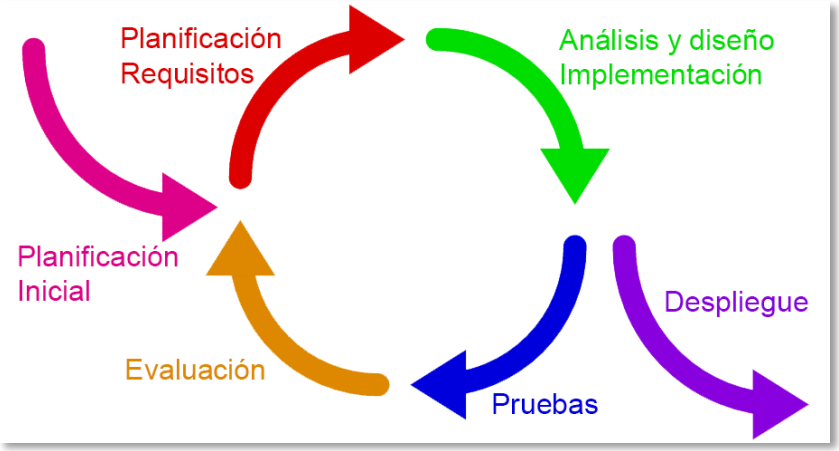
Para la planificación por objetivos, se empleó la herramienta **Trello**, una aplicación web que utiliza un sistema de fichas, el cual permite organizar todos los objetivos y darles una prioridad. Una vez los objetivos se cumplen, las fichas quedan archivadas. Se puede reorganizar todo con un sencillo gesto de ratón y tiene cliente de sincronización con el teléfono móvil, por lo que es

ideal para añadir nuevas tareas en cualquier momento y lugar. Las listas están ordenadas por prioridad, mayor prioridad cuanto más a la izquierda y más alto en la lista.

3. Captura de pantalla de la aplicación web Trello y de cómo se ha utilizado para la planificación por objetivos

2.8 MODELO INCREMENTAL / ITERATIVO

Un modelo de trabajo que tiene muy buena sinergia con la planificación por objetivos es el modelo incremental. Se trata de un sistema iterativo que repite ciclos, aumentando pequeñas partes de código cada vez, por lo que se puede definir como un conjunto de iteraciones incrementales. Una vez preparado el planteamiento inicial y los requisitos del proyecto (como se ha hecho hasta ahora), empieza un bucle de planificación, análisis, implementación, pruebas y evaluación, realizando este ciclo para cada objetivo que se quiere realizar. Pese a ser un sistema repetitivo con bastantes pasos, es muy efectivo cuando se busca cierta estabilidad, ya que antes de empezar el siguiente paso se está seguro de que el anterior está correctamente realizado y evaluado. La ventaja es que no se suelen emprender acciones demasiado arriesgadas y por lo general, se avanza sobre seguro a lo largo del proyecto, pudiendo deshacer un fallo fácilmente.



4. Traducción del modelo incremental / iterativo según Wikipedia

Finalmente, el programa se despliega cuando ya se encuentra en una versión utilizable o más estable, aunque en la práctica siempre se vuelve a entrar al ciclo, ya que el software siempre requiere un mantenimiento a lo largo de su vida útil.

3. Desarrollo y resultados

En este capítulo se va a explicar el proceso completo acerca del desarrollo de la aplicación, desde el primer boceto hasta los resultados finales, separando cada apartado de la aplicación y detallando todo el proceso de creación y pruebas realizadas. Se ilustrarán las funciones con diagramas para comprender mejor su funcionamiento.

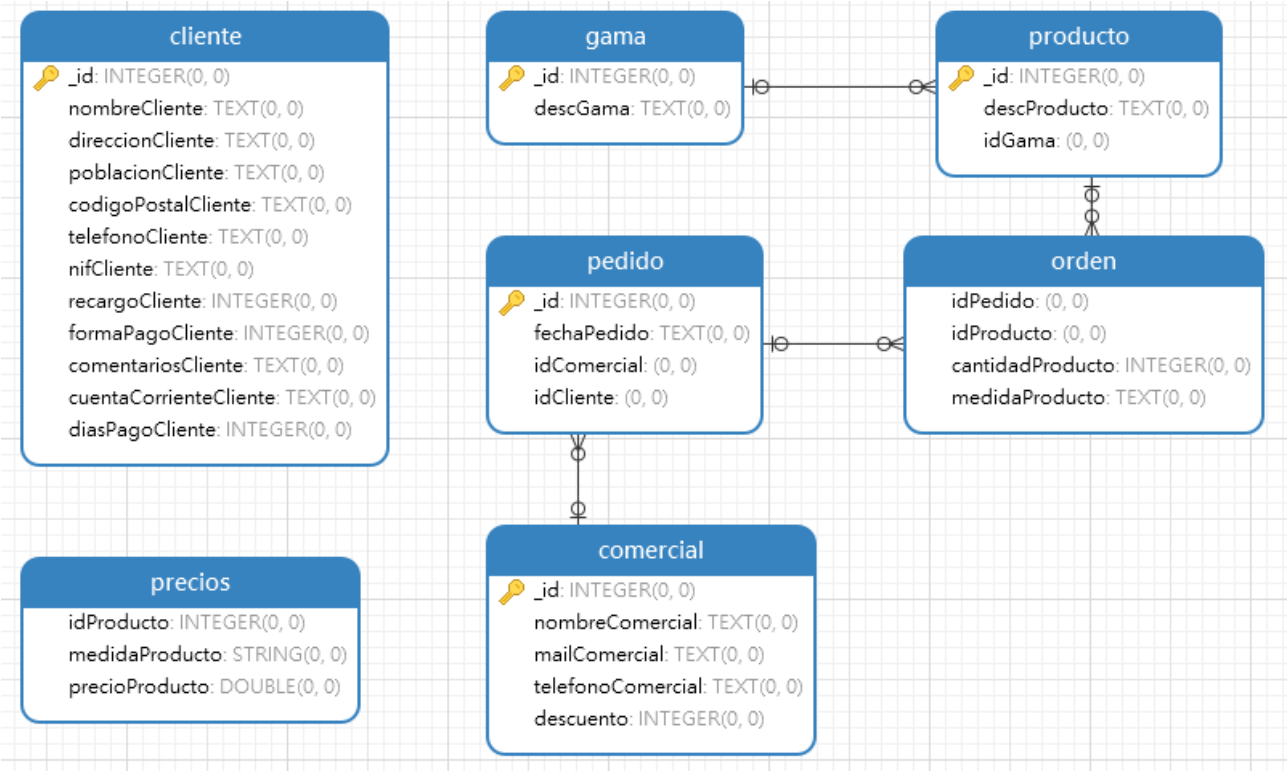
3.1 DISEÑO DE LA BASE DE DATOS

El primer esfuerzo para iniciar el proyecto consistió en plasmar la rutina de trabajo de los comerciales en una base de datos, para ello se realizaron diversas reuniones y ajustes, obteniendo como resultado una base de datos sencilla pero ajustada a las necesidades, evitando así incrementar su complejidad y perjudicando el tiempo de desarrollo. Las tablas que se diseñaron fueron las siguientes:

- **Cliente:** Almacena los datos personales del cliente y las formas de pago del mismo.
- **Comercial:** Contiene los datos del comercial que usa la aplicación, para cuando se genera una hoja de pedido, saber quién ha sido el autor. También almacena el descuento que puede aplicar sobre un pedido, ya que puede variar de un comercial a otro.
- **Gama:** Tabla que contiene las diferentes categorías de productos que existen.
- **Pedido:** Contiene los datos de cada pedido, con referencias al cliente, comercial y órdenes.
- **Orden:** Indica las cantidades y variaciones de un determinado producto en cada pedido (un color o una talla, por ejemplo).
- **Precios:** Aquí se guardan las relaciones entre los productos y sus variaciones, con el precio de cada una de ellas.
- **Producto:** Por último, en esta tabla se guardan todos los productos con su identificador, relacionados con la tabla gama, para saber a qué categoría pertenecen

3.2 DIAGRAMA ENTIDAD-RELACIÓN

Tras el diseño de la base de datos para el programa, el diagrama entidad-relación resultante es el siguiente:



5. Diagrama entidad-relación de la aplicación

Como se aprecia, tanto el cliente como los precios no están relacionados directamente con ninguna tabla. Esto se debe a haber realizado ajustes en el procedimiento que se realiza al usar la aplicación: **Si se quedaba relacionado obligatoriamente el cliente con el pedido, no se podrá crear un pedido sin haber dado de alta al cliente, caso que muchas veces no ocurría en la rutina de un comercial,** ya que ésta consistía en realizar rápidamente el pedido al cliente y después, con más tranquilidad, dar de alta sus datos y relacionarlo con el pedido (la aplicación está programada para no poder generar un pedido sin cliente). Algo parecido sucede con los precios, por lo que no se relacionaron con los productos con el fin de utilizar productos sin precio (descatalogados, por ejemplo). Hay que tener en cuenta que se está reemplazando un proceso que se realiza con un catálogo, papel y bolígrafo, por lo que es muy importante mantener la simplicidad, y reducir el número de pasos lo máximo posible.

El resto de relaciones sigue un patrón bastante usual:

- Una **Gama** puede disponer de uno o varios productos
- Un **Producto** sólo puede estar en una **Gama** y en varias **Órdenes** de pedido
- Un **Pedido** puede contener una o varias **Órdenes**
- Un **Comercial** puede tener uno o varios **Pedidos** a su cargo.

3.3 DISEÑO DE LAS CLASES DE LA APLICACIÓN

Una vez moldeada la base de datos, se procede a empezar a implementar las clases que formarán la aplicación.

3.3.1 ACTIVIDADES

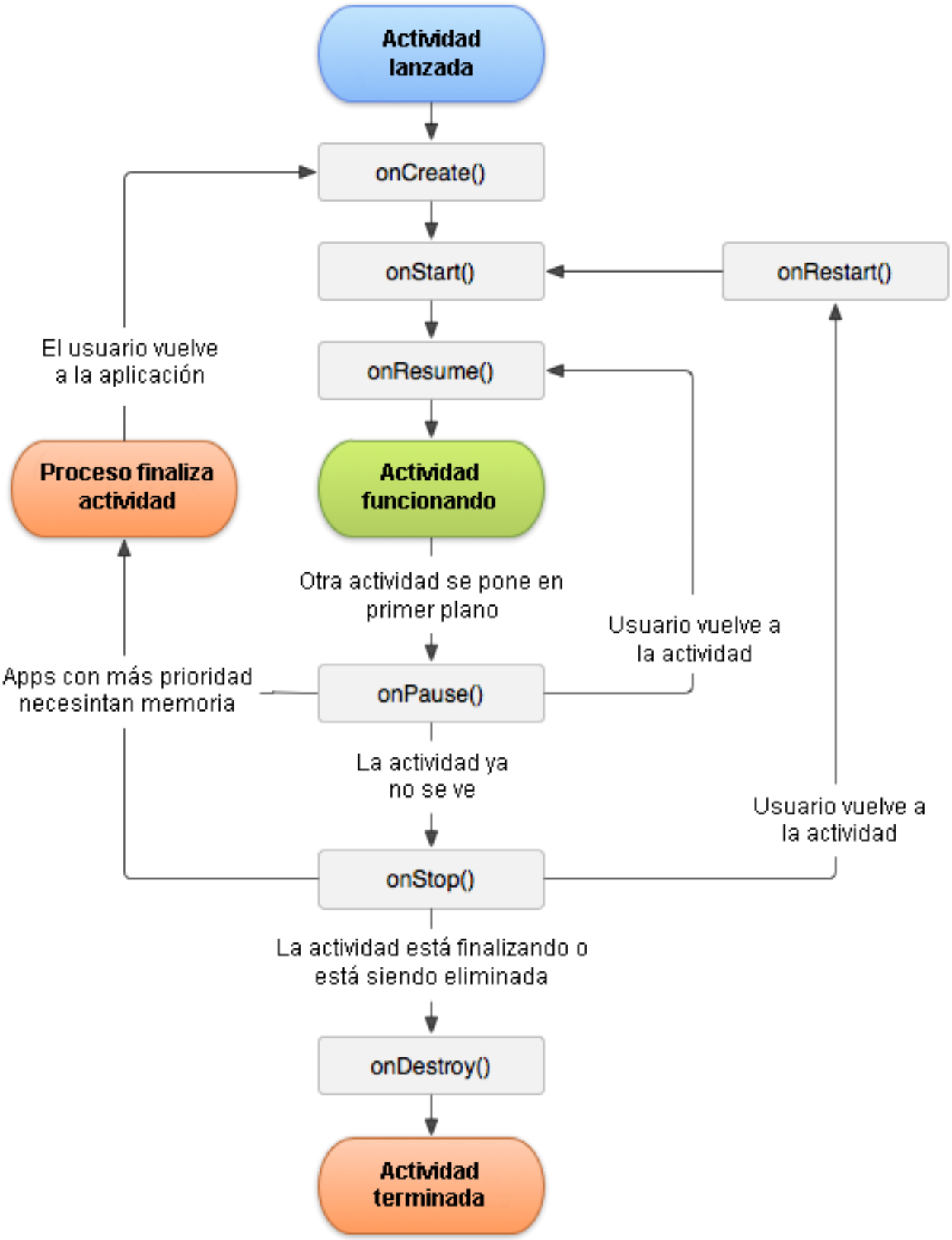
En Android, se emplean las llamadas **Actividades**. Una Actividad, es el resultado de una combinación de una **clase JAVA** y una **plantilla XML**, produciendo la combinación de ambas la representación gráfica e interactiva que se aprecia en la pantalla del dispositivo: **Una actividad se encarga de crear la "ventana" que interactúa con el usuario.**

Las actividades en un sistema Android son gestionadas como una **pila de actividades**. Cuando una nueva actividad se crea, se sitúa **encima** de la pila y pasa a ser la actividad **activa**. La actividad anterior permanece por debajo, pausada, y no volverá a ejecutarse hasta que la actual finalice.

Fundamentalmente una actividad tiene cuatro estados:

- Si está la actividad en primer plano, la actividad está **activa**
- Si una actividad ha dejado de estar en primer plano, pero sigue siendo visible (por ejemplo, cuando aparece una ventana emergente), la actividad está **pausada**. Cuando está pausada la actividad todavía posee todas sus propiedades, como si estuviese activa, pero podría ser finalizada en una situación en la que la aplicación necesitase más memoria disponible
- Cuando una actividad se tapa por completo por otra, está **detenida**, como en el caso anterior, sus datos siguen disponibles siempre y cuando no se necesite tomar sus recursos.

La actividad dispone de sus propios métodos para gestionar su ciclo de vida, el cual se encuentra entre dos actividades, **onCreate** y **onDestroy**. A continuación, se muestra un diagrama que explica el recorrido de una actividad desde que se crea hasta que finalmente se destruye.



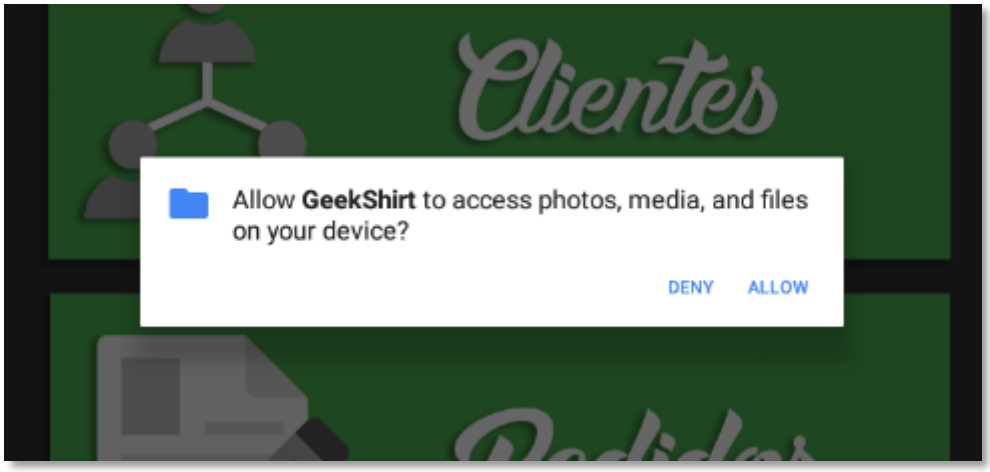
6. Ciclo de vida de una actividad. Traducción del original de la documentación de Android

Una vez comprendido el funcionamiento de las actividades se dispone a hacer un diseño a mano alzada de lo que serán finalmente las interfaces gráficas con las que el usuario final trabajará, intentando como máxima prioridad minimizar el número de pasos. Finalmente se obtienen el siguiente número de actividades.

NOTA: Con la finalidad de hacer una mejor demostración de las diferentes funciones que caracterizan el programa, la aplicación estéticamente se ha desarrollado entorno a una empresa ficticia llamada "GeekShirt" cuyo mercado principal son las camisetas. La aplicación es exportable para cualquier otro tipo de empresa cambiando los datos de la misma, los productos, logotipos, etc.…

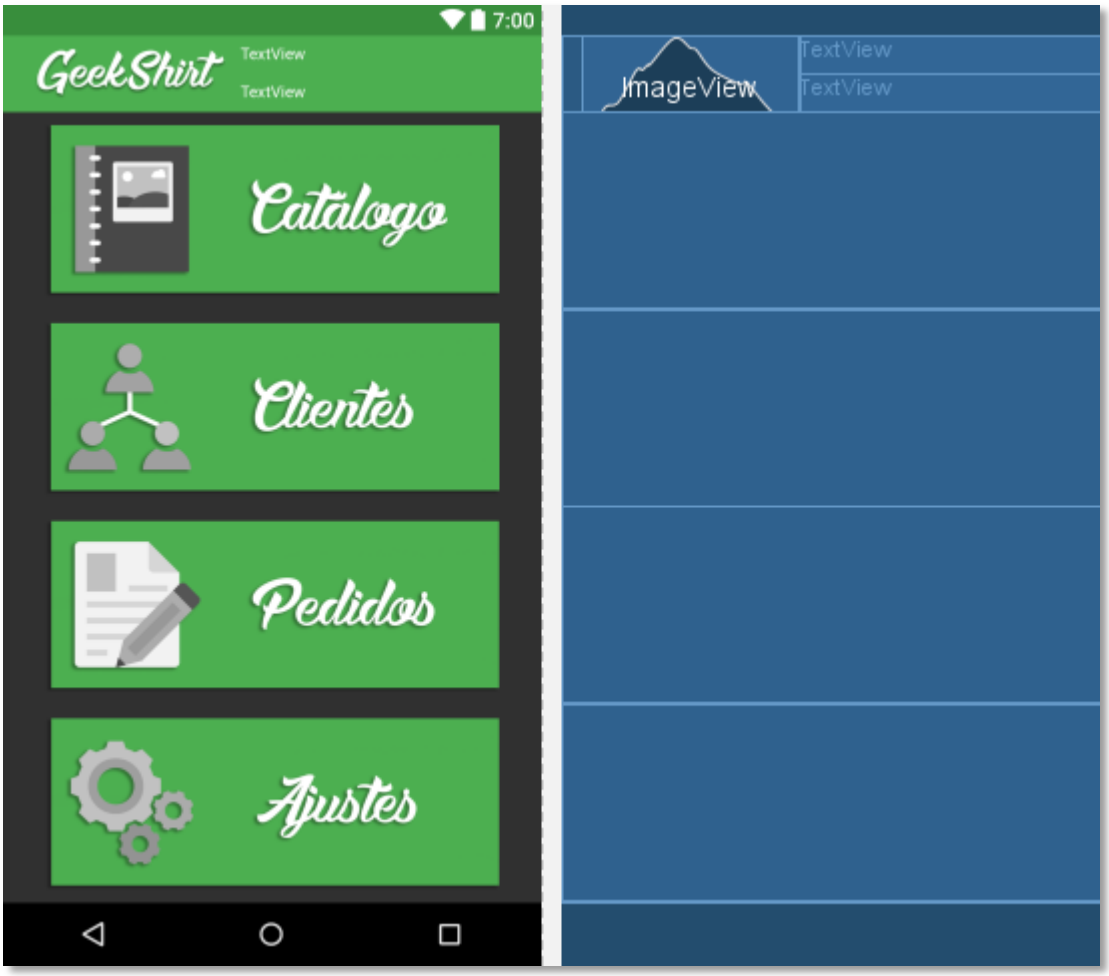
3.3.1.1 ACTIVITY_MAIN

Se trata de la **actividad principal**, que se cargará en primer lugar cuando se inicie la aplicación. Realiza diferentes comprobaciones durante su arranque, tales como si se trata del primer arranque, si hay algún pedido activo, la versión de Android en la que se está ejecutando, etc.



7. Aplicación durante el primer inicio en Android 6, preguntando por los permisos

En el caso de utilizar Android 6, pedirá permisos de almacenamiento en esta actividad. Una vez aceptados quedarán memorizados y no se volverán a solicitar.



8. Representación gráfica de la actividad principal "activity_main"

En las actividades en las que se cree necesario, se muestra una barra superior la cual indica en todo momento el nombre del comercial que está utilizando la aplicación y el pedido (si lo hubiese), con su número correspondiente.

9. Zoom de la barra superior y la información que proporciona

3.3.1.2 ACTIVITY_SELECTOR_CLASE

Esta actividad se mostrará tras pulsar el botón “Catálogo” desde la actividad principal, en ella se mostrarán las categorías que se habiliten para ello. Para el caso particular que se ha implementado, se muestran cuatro categorías diferentes de camisetas. Dependiendo de la cantidad de categorías, se podría hacer una variación a una lista que realizase un “scroll” o desplazamiento vertical, o anidar esta misma actividad, permitiendo así la creación de subcategorías. En este caso, al tratarse de cuatro categorías, con una plantilla lineal en formato vertical es suficiente para mostrar las categorías en pantalla.

10. Representación gráfica de la actividad "activity_selector_clase"

Tanto en esta actividad, como en la anterior, en lugar de usar los botones planos de Android, con el fin de añadir una mejora estética, se están utilizando unos botones especiales llamados **Image Buttons** que, como su propio nombre indica, son imágenes (creadas previamente) que poseen la particularidad de funcionar a modo de botón cuando se pulsa sobre ellas.

3.3.1.3 ACTIVITY_GALERIA

La actividad de la galería, es el corazón de la aplicación. Esta actividad sustituye al catálogo que porta el comercial en sus visitas, y permite conocer cada producto, su precio y las variaciones del mismo. Pese a ser una de las partes con más código, se ha conseguido un funcionamiento muy sencillo y cómodo para el usuario final.

El acceso a la galería se produce al pulsar el botón de cualquier categoría de la actividad anterior. Cuando se está creando la galería, se le indica el botón que ha pulsado, para que cargue la categoría correspondiente.

Sabiendo la categoría, se cargan en memoria todos los productos que contiene la misma y se establece el primero de la lista como el producto seleccionado. Cuando un producto es seleccionado, se muestran todos sus detalles. De todos estos accesos a la base de datos con el fin de obtener toda esta información, se encarga la clase **DatabaseHelper**, de la que se hablará posteriormente.

Una vez terminada la inicialización, se muestra el primer objeto de la categoría, del cual se puede obtener sus variaciones pulsando en los diversos controles, o cargar un producto diferente pulsando sobre el “scroll” de productos.

11. Transición al pulsar un producto

El “scroll” o deslizante situado en la parte inferior, se llena con los productos de la categoría se ha pulsado. Para ello, es indispensable que los artículos estén correctamente repartidos en las carpetas que dispone el programa a modo de biblioteca. Los productos se deben introducir en la base de datos cada uno con su número identificativo, y este número es el que generará la estructura de la biblioteca donde consulta la aplicación para buscar las imágenes:

/assets/img/número de categoría/número de producto

El número identificativo del producto empieza con su número de categoría, en un número de 3 cifras, por lo que habrá un máximo de 100 productos por categoría en principio, siendo esta capacidad modificable mediante el código en cualquier momento.

Para este caso particular de aplicación (tienda de camisetas) se han generado a este número dos variaciones más, uno para la camiseta femenina, y otro para la imagen que se muestra en el “scroll” de selección, quedando, por ejemplo, el producto **8** de la categoría **1** de la siguiente manera:

12. Formato de guardado de un producto en la biblioteca

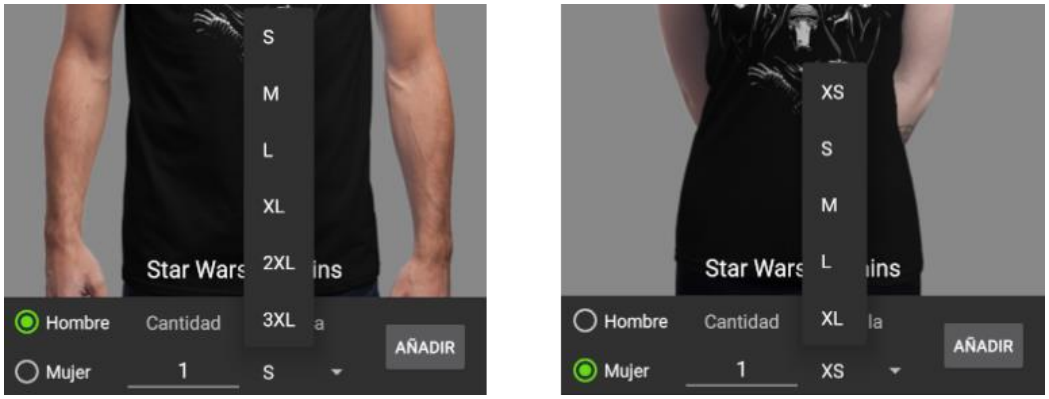
Creando un estándar como este, se consigue que la aplicación sea capaz de encontrar cualquier producto de cualquier categoría, si previamente se ha introducido correctamente en la base de datos. En este caso particular la galería sabrá que estas imágenes pertenecen al producto 8 de la categoría 1 y sus variaciones. Este método también obliga a introducir correctamente los productos para evitar así fallos en su visualización e inconsistencia en los datos.

13. Carga de una variación del artículo

De este modo la aplicación siempre sabe el artículo que se está cargando gracias al “scroll” inferior, e independientemente de la variación que se seleccione, la galería buscará esa variación del artículo cargado y la cargará en pantalla, quedando así automatizado el proceso para todos y cada uno de los productos dados de alta en la base de datos.

También se puede apreciar que no solo se está cargando la imagen sino la variación completa del artículo, cambiando así los selectores de talla (No son las mismas para hombre que para mujer) y el precio del artículo.

Se dispone de una barra para controlar las variaciones del pedido, como se ha visto en la miniatura anterior. En el grupo de **Radio Botones** de la izquierda, se controla si la variación del artículo es la masculina o la femenina. En el centro-izquierda existe un campo para escribir la cantidad del artículo que se desea añadir al pedido, el cual al pulsarlo desplegará un teclado numérico para editarlo cómodamente. En el centro derecha está el selector de talla, el cual varía, junto con las imágenes, si la versión seleccionada es la masculina o la femenina, ya que en este caso particular las tallas masculinas llegan a tallas de mayor tamaño y lo contrario sucede con las femeninas, llegando a tallas más pequeñas. Internamente, en la base de datos, está implementada cada una de ellas, con su precio asignado.

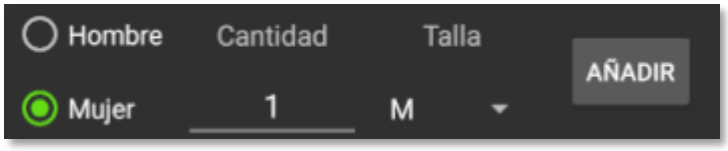


14. Muestra de los "Spinner" para las tallas masculinas y femeninas

Estos selectores, tienen un pulsador el cual detecta la talla seleccionada, cargando en pantalla el precio que se le haya asignado a esa talla de esa variación en la base de datos. Nuevamente, con la intención de automatizar todo el proceso y asegurar que cada talla tiene su precio asignado, se ha de introducir manualmente en la base de datos para que muestre el precio correctamente.

15. Muestra de una variación en la talla, repercutiendo en el indicador del precio

El último elemento de la barra de control, es el botón “añadir”. Inicialmente la galería disponía de un modo de demostración cuando era obligatorio tener el perfil del cliente agregado para tener el pedido. Como el comercial no siempre podía haber dado de alta al cliente antes de realizar un pedido, la función demostración se eliminó porque se pasaba a tener un pedido activo siempre. La peculiaridad de este modo consistía en que se eliminaba de la plantilla el botón, por lo que quedaba más amplia y agradable a la vista, pudiendo manejar los controles para ver las diferentes variaciones del producto, pero sin poder agregarlos al pedido.



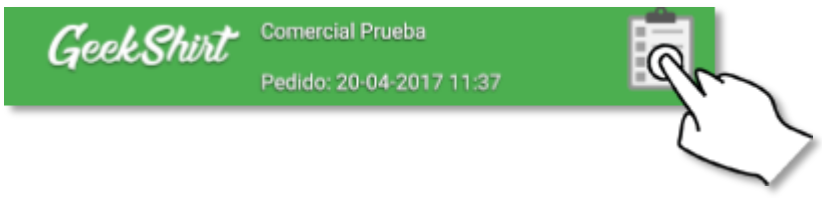
16. Barra de control de la galería

Con el modo de funcionamiento normal, la función de este botón es agregar al pedido un artículo con las variaciones que se hayan establecido previamente en la barra de control. Cuando El artículo es insertado, se muestra otra de las herramientas gráficas de Android llamada **Toast**, la cual consiste en un mensaje flotante que se aparece durante un pequeño periodo de tiempo. En este caso una herramienta ideal para mostrar el aviso que los artículos se han agregado satisfactoriamente al pedido.

17. Captura de un mensaje flotante o "Toast" de Android

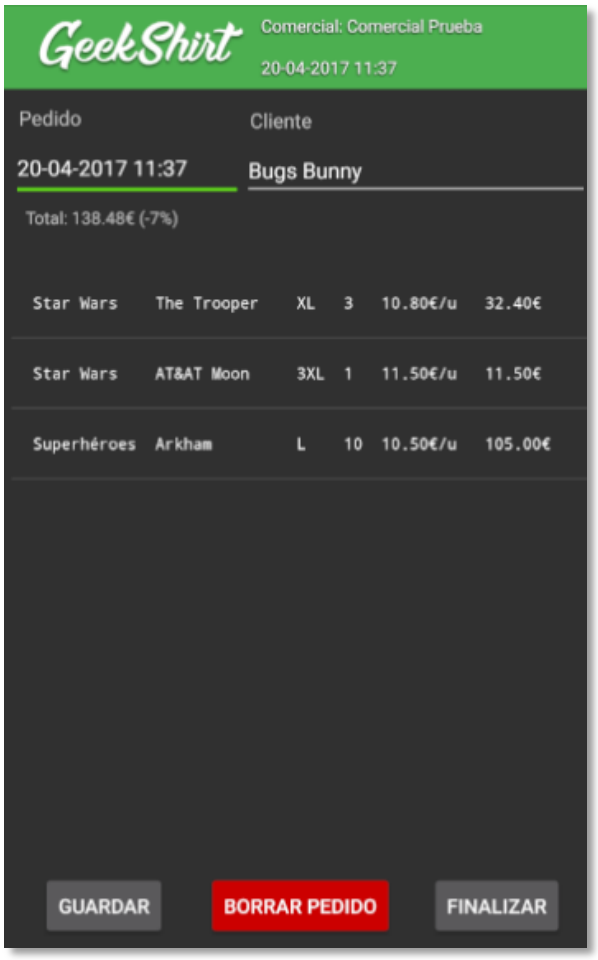
El último elemento de la galería es un pequeño botón que aparece en la parte derecha de la barra superior. Igual que el botón “añadir”, éste no siempre estaba activo, pero posteriormente pasó a ser un elemento fijo en la galería.

Este botón otorga a la galería la función de abrir una actividad que muestra el estado del pedido actual. Es el mismo acceso que se realiza desde el botón del menú principal, pero se implementó aquí además por si un cliente o el mismo comercial quería saber qué cantidad de productos llevaba acumulada y el total de ellos en cualquier momento de la visita.



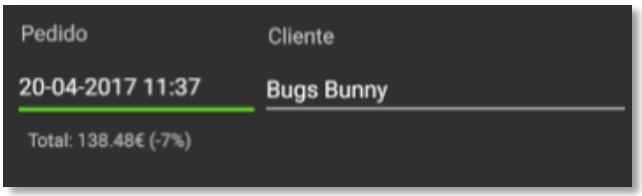
18. Barra superior de la galería

3.3.1.4 ACTIVITY_EDITOR_PEDIDO



19. Vista general de la actividad de edición del pedido

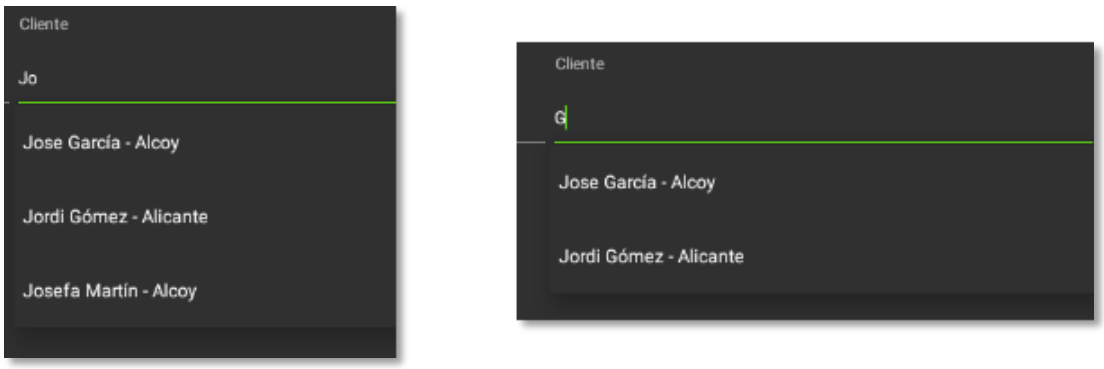
Esta actividad es la que permite administrar el pedido y su contenido. Al iniciarse, consulta la base de datos para devolver el pedido activo actualmente y mostrar su contenido. Tiene implementada la barra superior para mostrar información acerca del comercial y el pedido.



20. Zoom de los controles del editor de pedido

El primer bloque de controles que aparece, contiene tres campos de texto dinámicos. El primero de ellos indica el nombre del pedido. Para evitar problemas de duplicidades, se decidió que el nombre se generase automáticamente con la hora y fecha, siendo la diferencia mínima entre dos pedidos de un minuto, situación poco probable ya que en ese periodo de tiempo es difícil que un comercial genere otro pedido y lo sobrescriba.

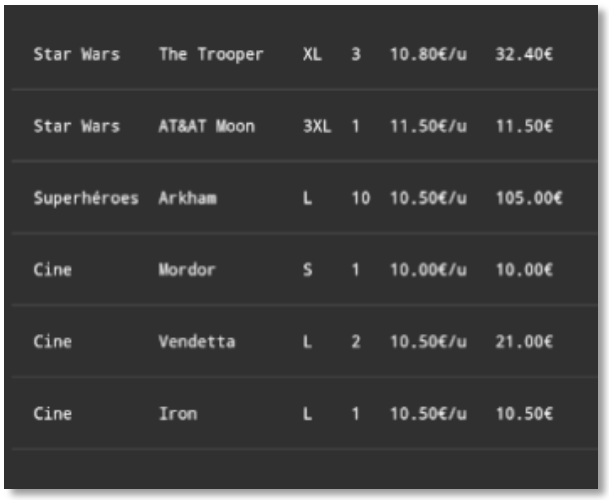
A continuación, existe otro campo de entrada de texto, en este caso se introduce el nombre del cliente al que se le está realizando el pedido. Este campo contiene una función de autocompletado, el cual mostrará los nombres que coincidan con el texto conforme se vaya escribiendo (para más ayuda, muestra la localidad, por si se diera el caso particular de tener dos clientes con el mismo nombre). Mediante este proceso, la aplicación se asegura de que el cliente seleccionado sea el correcto, ya que, si el nombre no coincide con ningún cliente existente en la base de datos, no permitirá asignarlo al pedido ni realizar la función de finalizado, nuevamente con el fin de aumentar la fiabilidad y consistencia de los datos.



21. Muestra del autocompletado de la aplicación, tanto por nombre como por apellido

Como último componente dentro del primer bloque, existe un pequeño texto que se muestra si el pedido tiene algún contenido dentro, indicando la suma de todo el valor y aplicando el descuento que el comercial haya podido establecer previamente. Se ha situado en la parte superior por comodidad, ya que cuando se accede desde la galería (por ejemplo, cuando el cliente quiere ver lo que lleva acumulado) está fácilmente visible, evitando así tener que perder tiempo localizando ese texto.

El siguiente elemento de esta actividad, es la lista de artículos. Se trata de una lista básica de Android, poblada durante la carga de la actividad con las diferentes órdenes de productos que contiene el pedido, mediante una consulta a la base de datos.



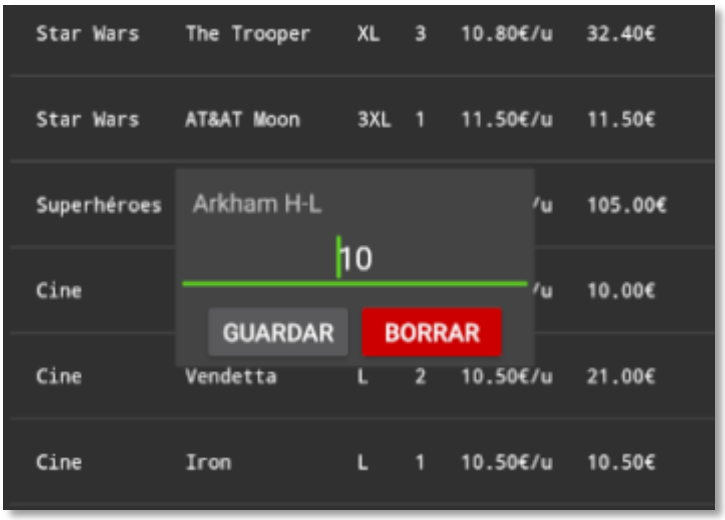
Star Wars	The Trooper	XL	3	10.80€/u	32.40€
Star Wars	AT&AT Moon	3XL	1	11.50€/u	11.50€
Superhéroes	Arkham	L	10	10.50€/u	105.00€
Cine	Mordor	S	1	10.00€/u	10.00€
Cine	Vendetta	L	2	10.50€/u	21.00€
Cine	Iron	L	1	10.50€/u	10.50€

22. Ejemplo del listado de productos de un pedido

El criterio a la hora de separar los artículos de la lista, es por sus variaciones (en este caso el tallaje) ya que cada uno de ellos tiene particularidades como el tamaño y si el corte es masculino o femenino, por lo que aparecerán los productos solicitados agrupados de esa manera. **La lista se ha configurado con una fuente monoespaciada y formateada de manera que nunca llegue a solaparse una tabulación con la siguiente**, teniendo en cuenta la longitud de los nombres de las categorías y artículos, consiguiendo de esta manera que quede más agradable a la vista y sea más sencillo encontrar un artículo o característica la lista.

Teniendo en cuenta también que el cliente tiende a cambiar y ajustar el pedido conforme se ha creado se han implementado dos medidas que hacen más sencillo e intuitivo su uso:

1. Cuando en la galería se agrega un producto con una cantidad determinada. Si se vuelve a pulsar sobre el botón añadir, la misma función del botón comprobará si está ese artículo en el pedido y, en caso afirmativo sumará la nueva cantidad a la ya existente.
2. En la vista del pedido, se ha implementado un pulsador para la lista. Este pulsador cargará la variación del producto seleccionada y permitirá variar su cantidad o eliminar el artículo. Una vez pulsado el botón de guardado o eliminado, la actividad al completo se refresca con los datos modificados.



23. Popup de modificación del artículo

Como se puede observar en la miniatura 23, cuando se pulsa sobre un artículo de la lista, ésta genera un “pop-up” permitiendo elegir entre modificar la cantidad de artículos en el pedido o borrar esa variación del mismo. Ambos botones tienen implementada una función de refresco que actualizará la lista a los nuevos valores.

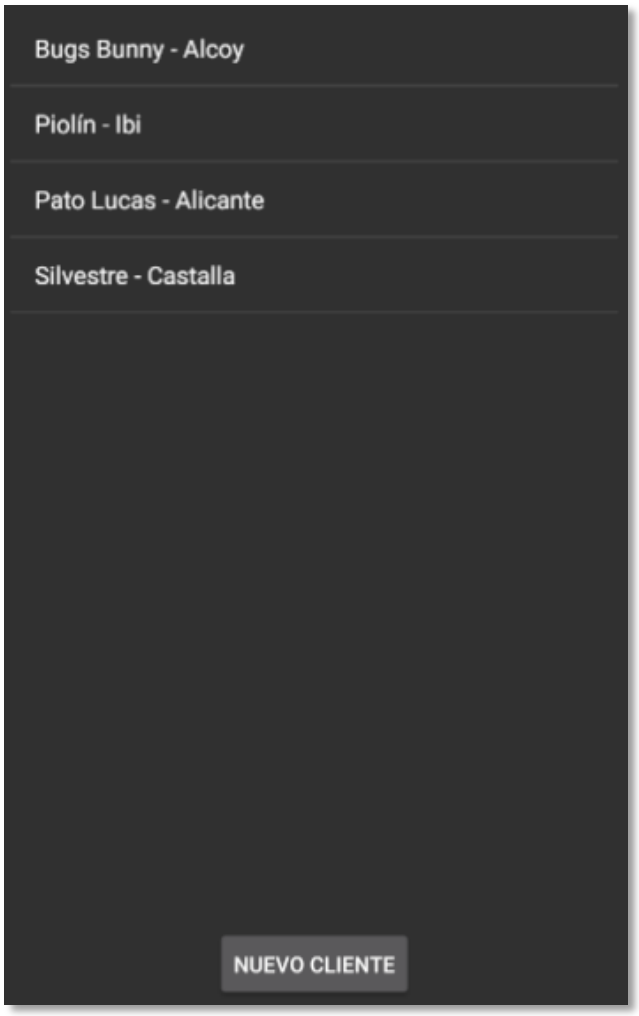
Por último, en la parte inferior de la actividad, existe otra pequeña barra que permitirá controlar el pedido.

24. Botones para modificación del pedido

- El botón **GUARDAR** tiene la función de guardar las modificaciones realizadas al pedido. Aunque el contenido del pedido se guarda en el acto, otras modificaciones como el cliente propietario del pedido, sí deberán actualizarse manualmente. Teniendo en cuenta que por defecto se crean sin cliente, esta función se asegura de que cuando se actualiza el cliente de un pedido, éste exista en la base de datos. Si no existiese, se mostrará un pequeño aviso en forma de **Toast** y el pedido seguirá sin cliente, por lo que será necesario darlo de alta e introducirlo correctamente para poder finalizar el pedido
- El botón **BORRAR** eliminará el pedido. Automáticamente se genera uno nuevo en blanco, ya que al borrarlo la aplicación retorna a la actividad principal y ésta, al comprobar que no hay pedido activo, creará uno vacío. Como se ha comentado anteriormente, esta función se implementó posteriormente para agilizar el trabajo del comercial, por lo que nunca estará la aplicación en un estado “sin pedido activo”.
- El botón **FINALIZAR** archivará el pedido, asignándole un nuevo número identificativo y a las órdenes asociadas al mismo, ya que por defecto se utiliza el número **0** como identificador para el pedido activo. El botón comprobará los pedidos que existen archivados y le asignará un número nuevo de acuerdo a los existentes, por lo que el pedido “activo” desaparece y se creará uno nuevo al volver a la actividad principal como sucede con el botón de borrado. Tras archivar el pedido, se genera el PDF del mismo (el proceso del cual se explica más adelante) y se abre una nueva actividad, en la que el sistema Android buscará la aplicación para leer PDFs por defecto y abrirá el documento generado. De no existir aplicación, la aplicación igualmente se cerrará, pero guardando el archivo PDF en memoria.

3.3.1.5 ACTIVITY_SELECTOR_CLIENTE

Como siguiente opción en el menú principal, se dispone del apartado “Clientes”. Pulsando ese botón en la actividad principal, se inicia una nueva actividad que mostrará un listado de los clientes existentes (previamente consultados a la base de datos) y, de manera similar a la actividad anterior, generará una lista formateada con el nombre del cliente y su población para localizarlo con mayor facilidad. Pulsar sobre cualquier elemento de la lista, iniciará una actividad para editarlo.

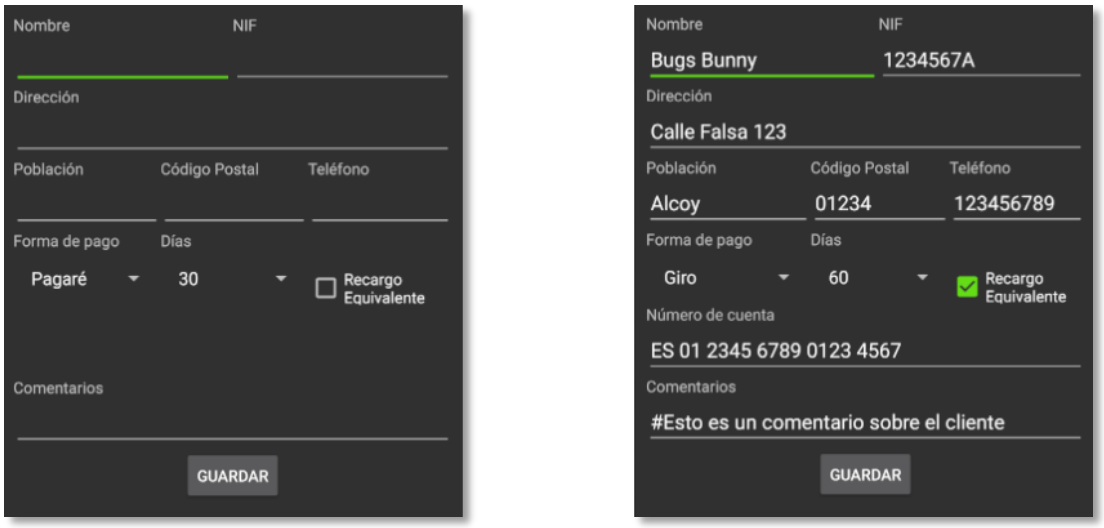


25. Imagen de la actividad "activity_selector_cliente" con algunos clientes de ejemplo

Como se ve en la miniatura 25, no se dispone de la barra superior (no es necesaria, ya que es irrelevante aquí la información que proporciona), ni tampoco se dispone de una opción de borrado para evitar la pérdida de datos, ya que los clientes con la herramienta de trabajo de los comerciales.

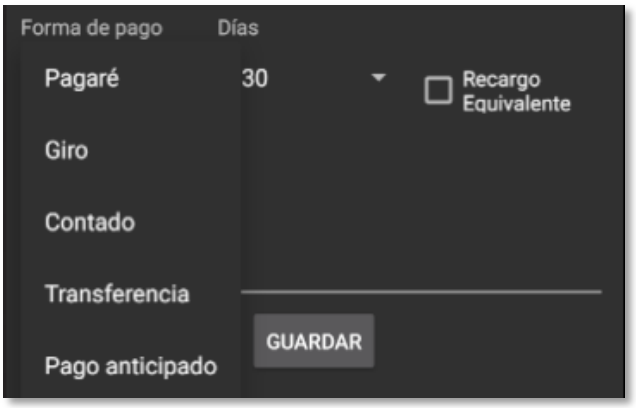
3.3.1.6 ACTIVITY_EDITOR_CLIENTE

Partiendo de la actividad anterior, tanto como si se crea un nuevo cliente, como si se modifica un cliente existente (pulsando sobre él en la lista), en esta actividad se editan los diferentes datos personales y campos de interés de un cliente.



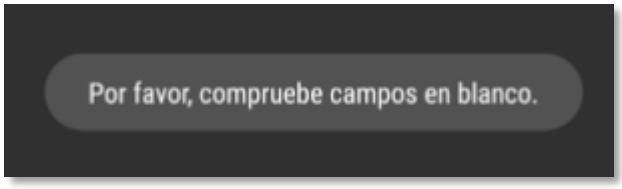
26. Ventana de edición del cliente al crear uno nuevo (izquierda) y al editar uno existente (derecha)

El editor, también tiene la particularidad de adaptarse según la forma de pago. Por ejemplo, dejará visible un campo para introducir una cuenta bancaria si se selecciona la opción de pago “Giro”, o generará un campo con la cuenta corriente de la empresa en el PDF si se selecciona la opción “Transferencia”. El resto de opciones no tienen comportamientos diferentes salvo activar o desactivar el campo días (que siempre será 30, 60 o 90). Tanto las formas de pago como los periodos fueron previamente revisados y consultados con los comerciales que le darían el uso final a la aplicación.



27. Menú desplegable de las formas de pago

Adicionalmente, se han implementado comprobaciones para que, a la hora de modificar y guardar los datos del cliente, parte de éstos sean obligatorios, para evitar tener campos en blanco y evitar futuros problemas con los datos. En caso de dejar algún campo en blanco, aparecerá un mensaje **Toast** indicando de que hay campos por revisar.



28.Mensaje tipo "Toast" informando de que se han campos sin completar

Por último, se dispone de un campo de comentarios, en este caso opcional, que se usará para cualquier información adicional del cliente y que no exista el campo adecuado para introducirla.

3.3.1.7 ACTIVITY_MENU_AJUSTES

Esta actividad se encarga de representar la última sección del menú principal llamada “Ajustes”, en la que se abre un abanico de opciones para configurar la aplicación.

29. Vista general de la actividad "Ajustes"

En esta actividad se decidió dejar la barra superior, ya que de este modo el comercial puede ver de un rápido vistazo el descuento que tiene aplicado sobre el pedido, de modo que si se pulsa el botón “editar descuento” aparecerá un pop-up que permitirá editarlo hasta un máximo de un 13% (cantidad máxima aplicada por los comerciales en este caso, modificable en el código si se desea)

30. Pop-up de edición de descuento

En cuanto a los demás botones, se diferencian por las siguientes funciones:

- **Reporte pedidos:** Genera un documento PDF con todos los pedidos archivados existentes en la base de datos, la cuantía de cada uno de ellos y sus datos de una manera abreviada.
- **Reporte clientes:** De igual manera que con los pedidos, se genera un reporte con todos los clientes y sus datos personales, volcado a un documento PDF convenientemente formateado. Esta función se implementó para poder conservar la información de los clientes, ya que cuando se realiza alguna actualización de la aplicación que incluya modificación de la base de datos, ésta ha de ser borrada y se perdían los datos (una desventaja de usar SQLite, como se ha mencionado anteriormente). De este modo se podían conservar los datos de cada usuario por si en un futuro se necesitasen volver a utilizar, poder insertarlos de nuevo.
- **Editar comercial:** Inicia la actividad que aparece en el primer arranque de la aplicación, solo que en este caso en vez de introducir por primera vez los datos personales del comercial, solamente se modifican, por ejemplo, si se han introducido mal los datos la primera vez, o el terminal cambia de comercial.
- **Archivo pedidos:** Abre una nueva actividad que muestra los pedidos archivados (como se ha explicado anteriormente, una vez finalizado, el pedido se archiva), con la finalidad de poder consultar pedidos anteriores, eliminarlos definitivamente, o generar de nuevo el PDF del pedido.

Como se puede comprobar, en esta parte de la aplicación no se ha cuidado tanto la estética y se han dejado los botones planos sobre una plantilla **Linear Layout**. Esto se debe a que, por una parte, es el comercial el que solamente ve esta parte de la aplicación, y por otra,

es una actividad sujeta a continuos cambios y que muy probablemente acabará creciendo con nuevas opciones y funciones, por lo que es buena idea dejar la plantilla lo más sencilla posible para no tener problemas cuando vaya creciendo (de este modo, se va adaptando automáticamente).

3.3.1.8 ACTIVITY_EDITOR_COMERCIAL

Esta actividad está formada por tres campos de entrada de texto cuyo cometido es almacenar los datos personales del comercial que utilizará la aplicación, para mostrarlos posteriormente en el PDF que se genera tras un pedido o en la barra superior de la interfaz.

31. Vista de la actividad "activity_editor_comercial"

Como sucede con la actividad de edición de clientes, no permitirá dejar un campo en blanco, por lo que obligatoriamente se han de rellenar los tres.

Cada vez que la aplicación se inicia, se comprueba si hay activo algún comercial en la base de datos. En caso negativo, como sería en el primer arranque, abrirá automáticamente esta ventana para introducir uno, ya que de lo contrario no se podrá utilizar la aplicación.

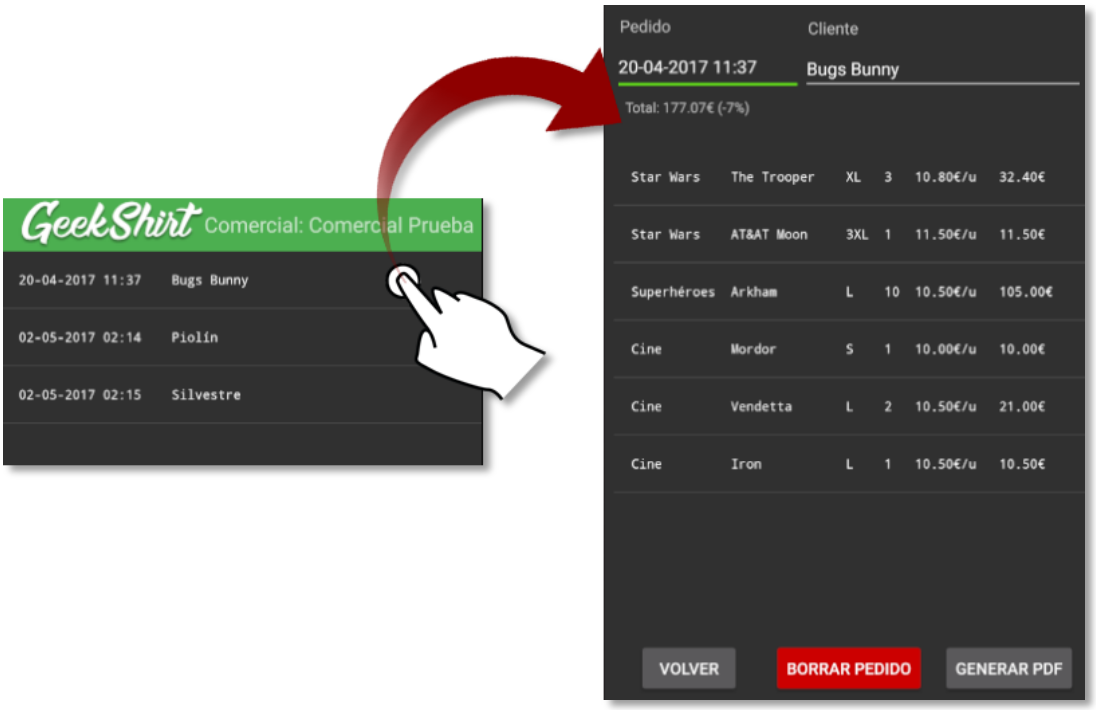
3.3.1.9 ACTIVITY_ARCHIVO_PEDIDOS

Por último, en esta actividad, se muestra una lista del tipo **ListView** como las que ya se han visto con anterioridad en la actividad que muestra los clientes, o la que detalla el contenido de un pedido. En este caso, mostrará los pedidos generados anteriormente y que se han archivado con el fin de poder volver a consultarlos o generar el PDF de nuevo.

32. Listado de pedidos archivados

De una manera también debidamente formateada, se muestra en pantalla cada pedido de la siguiente manera: Por una parte, la fecha (y al mismo tiempo nombre identificativo) del pedido, y por otra parte el nombre del cliente al que pertenece. Quedando de este modo de una manera bastante clara sin que dé lugar a confusión.

Si se pulsa encima de cualquier pedido, se iniciará la ya conocida actividad de edición del pedido, con las funciones que le corresponden.



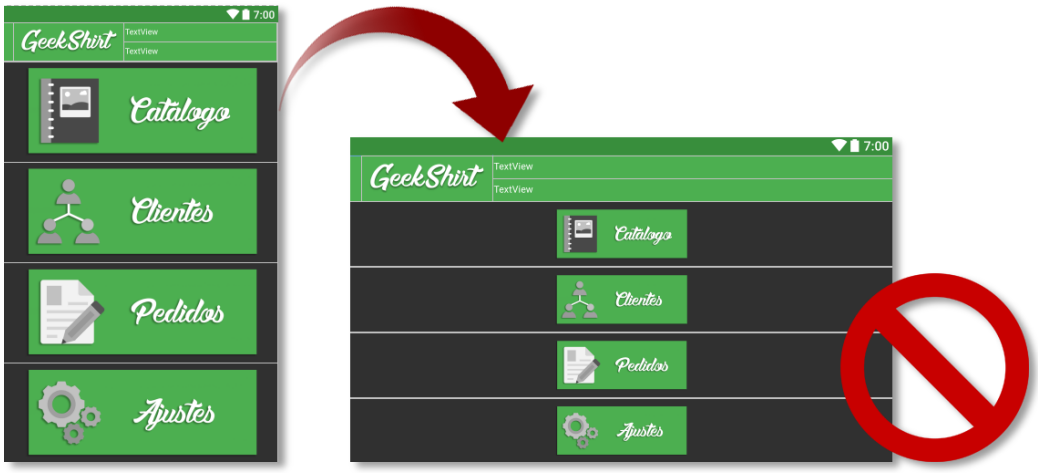
33.Carga de un pedido archivado

3.3.2 AJUSTES ESPECIALES

Por norma general, Android es muy flexible cuando se trata de ajustar las plantillas a la pantalla, si no se trata de plantillas demasiado complejas, suelen adaptarse bien a las pantallas más comunes (una densidad media o alta de píxeles por pulgada y resoluciones de 720p en adelante). Sin embargo, en este proyecto han habido algunas actividades que se han tenido que adaptar debidamente al modo horizontal, como es el caso de la galería o el menú principal, permitiendo así una mayor comodidad al comercial al poder elegir cómo hacer sus presentaciones.

Tras observar el uso de la aplicación, se decidieron adaptar tres plantillas manualmente. Por una parte, eran las tres plantillas que más vería el cliente, por lo que es interesante dejarlas bien presentadas sin que nada quede descuadrado, y por la otra, por la mayor complejidad de la plantilla, en el caso de la galería.

Para ello se volvieron a crear manualmente, suponiendo una mayor dificultad solamente el caso de la galería, ya que las otras actividades, al ser menús, solamente fue necesario cambiar la orientación de la plantilla para que los botones quedasen bien encuadrados.

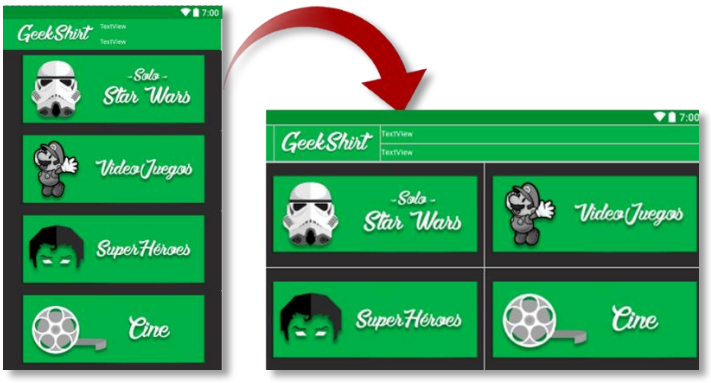


34.Ejemplo de un mal encuadre Automático de Android

35. Corrección manual del encuadre

El entorno Android Studio resulta muy cómodo para realizar este tipo de ajustes, ya que automáticamente configura la aplicación para que use la plantilla que se ha creado para usar alternativamente con otra orientación. **En caso de no haber creado ninguna, Android decidirá como ajustar la pantalla**. En muchos de los casos, si la plantilla es sencilla, no suele ser necesario nada más que ese ajuste automático, pero si tiene una complejidad mayor, o se quiere dar una gran personalización estética, es conveniente crear una plantilla alternativa para cada actividad (incluso para cada densidad de pantalla, necesitando invertir mucho más tiempo). Como se ha mencionado anteriormente, en este proyecto particular se han decidido adaptar las tres actividades más a la vista del cliente, ya que se ahorra tiempo y el ajuste automático es más que suficiente para el resto.

36.Ajuste manual horizontal de la actividad principal



37.Ajuste manual horizontal del selector de categoría

38.Ajuste manual horizontal de la galería

3.3.3 CLASES

Las clases Java en Android, se utilizan para todo, convierten en objetos las tablas de la base de datos, aplican la parte lógica y se responsabilizan del comportamiento de las actividades. También se encargan de hacer trabajos especiales, como comunicarse con la base de datos. Por ello se han separado para este proyecto en tres categorías para poder explicar correctamente su funcionamiento: **Clases de base de datos, de actividades, y especiales.**

3.3.3.1 CLASES DE BASE DE DATOS

Estas clases se encargan de convertir las tablas de la base de datos en objetos manipulables a través de la aplicación. La conversión es relativamente sencilla para todas y cada una de las clases, ya que se han implementado los mismos atributos en la clase que los que existen en la base de datos, y añadiendo funciones **getters** y **setters,** las cuales permiten recuperar datos cuando se instancia un objeto desde otra clase. Las clases resultantes y sus atributos son los siguientes:

```
public class Cliente {

    private int id;
    private String nombre;
    private String direccion;
    private String poblacion;
    private String codigoPostal;
    private String telefono;
    private String nif;
    private int recargo;
    private int formaPago;
    private String comentarios;
    private String cuentaCorriente;
    private int diasPago;
```

```
public class Orden {

    private int id_pedido;
    private int id_producto;
    private int cantidad;
    private String medida;
    private String nombreProducto; //Campo sólo para la lista
    private String nombreGama; //Campo sólo para la lista
```

```
public class Gama {

    private int id;
    private String desc;
```

```
public class Precio {

    private int idProducto;
    private String medida;
    private double precio;
```

```
public class Comercial {

    private int id;
    private String nombre;
    private String mail;
    private String telefono;
    private int descuento;
```

```
public class Producto {

    private int id;
    private String desc;
    private int id_gama;
```

```
public class Pedido {

    private int id;
    private String fecha;
    private int id_comercial;
    private int id_cliente;
```

39. Representación en clases JAVA de las tablas de la base de datos

Como se puede apreciar en la imagen, excepto la tabla **Orden**, que se le han agregado dos campos más para mejorar la representación gráfica en la aplicación, son un reflejo exacto de las tablas existentes en la base de datos. **El cometido de estas clases es el de la manipulación de los datos como objetos en el programa.**

3.3.3.2 CLASES DE ACTIVIDADES

Como se ha explicado anteriormente, las actividades son el resultado de una combinación entre una clase **JAVA** y una plantilla **XML**, siendo las plantillas las que aportan la parte gráfica y las clases la parte lógica y su comportamiento.

Antes de citar las clases asociadas a cada plantilla y su funcionamiento, es importante saber cómo interactúan estas dos partes. En primer lugar, se ha de diseñar la plantilla XML, creando los diferentes objetos que se han podido ver hasta ahora, como **etiquetas, botones, listas, campos de edición de texto, etc.** A éstos se les ha de dotar de un nombre identificativo, por lo que no puede haber dos objetos con el mismo nombre en una plantilla.

Una vez creados y posicionados en la plantilla ¿Cómo se logra que funcionen?

Muy sencillo, en la clase se han de crear los objetos de la plantilla que se deseen manipular en tiempo de ejecución y durante el método **onCreate** (en la creación de actividad) estos objetos se inicializan relacionándolos con el elemento de la plantilla, por lo que el nuevo objeto instanciado en la clase, tendrá las propiedades y atributos adecuados para dicho objeto.

Un ejemplo práctico: La barra superior de la aplicación. Ésta contiene dos elementos del tipo **TextView**, es decir, una etiqueta que muestra un texto. Por defecto, ésta puede tener el texto que previamente se le asigne, pero si se quiere lograr un comportamiento dinámico, es necesario instanciar ambas etiquetas en la clase, y una vez allí poder modificar su contenido, leerlo, mostrarlo, esconderlo, o tantas acciones como funciones disponibles tenga dicho elemento.

10. Equivalencia del código XML con su representación gráfica

Como se ve en la miniatura 40, las dos etiquetas de la cabecera, entre muchos otros atributos que definen sus características visuales, se encuentra en primer lugar el **id** que da nombre a cada una de las etiquetas, llamándose la superior **textViewComercial** y **textViewPedido** la inferior. Aunque se les puede dar otros usos como dar mensajes informativos, se les ha dado esos nombres porque su principal función es mostrar el comercial y pedido activo.

Para conseguir que se comporten de manera dinámica en la clase que deseada, es necesario:

1. Que la plantilla esté relacionada con la clase, de modo que la clase pueda reconocer los elementos de la plantilla. Esta acción la realiza Android Studio por defecto cuando se crea una actividad, pero es necesario, en el método **onCreate** de la actividad añadir la siguiente función:
   `setContentView(R.layout.nombre_de_la_plantilla);` Siendo el nombre de la plantilla el relacionado con la clase, que en conjunto forman la actividad. De este modo se le está indicando a la clase que ha de cargar este contenido visual y los componentes que hay en su interior.
2. Haber definido los objetos que se encargarán del comportamiento dinámico de los objetos de la plantilla. En este caso la declaración e instanciación quedaría de la siguiente manera:
   ```
   cabeceraComercial = (TextView)
   findViewById(R.id.textViewComercial);
   ```

```
cabeceraPedido = (TextView) findViewById(R.id.textViewPedido);
```

En este punto, la clase y la plantilla quedan relacionados, y se podrá modificar en tiempo de ejecución la variable `cabeceraComercial` o `cabeceraPedido` permitiendo obtener su texto, modificarlo, cambiar la visibilidad y un amplio repertorio de métodos que facilitan estas acciones.

Pasado este punto, se procede a realizar una consulta a la base de datos, la cual devolverá un objeto de la clase **Pedido**, y otro de la clase **Comercial**. Se obtendrá el nombre de ambos y mediante el método **setText,** se consigue que cada vez que se crea la actividad, cargue los nombres adecuados; el del comercial y el del pedido activo. Al modificarse cualquiera de ellos en la base de datos, se modificará la representación gráfica de la etiqueta.

41. Ejemplo de etiquetas generadas dinámicamente

Otro claro ejemplo del control que ejerce una clase sobre la plantilla se puede ver en el caso de la galería. Además de la barra superior anteriormente citada, su comportamiento varía bastante depende de las acciones que se utilicen. Durante su creación, se realiza:

- Enlazamiento de todos los componentes de la plantilla gráfica con sus respectivos objetos dentro de la clase para su posterior manipulación
- Inicialización de la barra superior
- Carga de los artículos de la categoría elegida y sus precios

En este punto, se configura el comportamiento de los siguientes elementos:

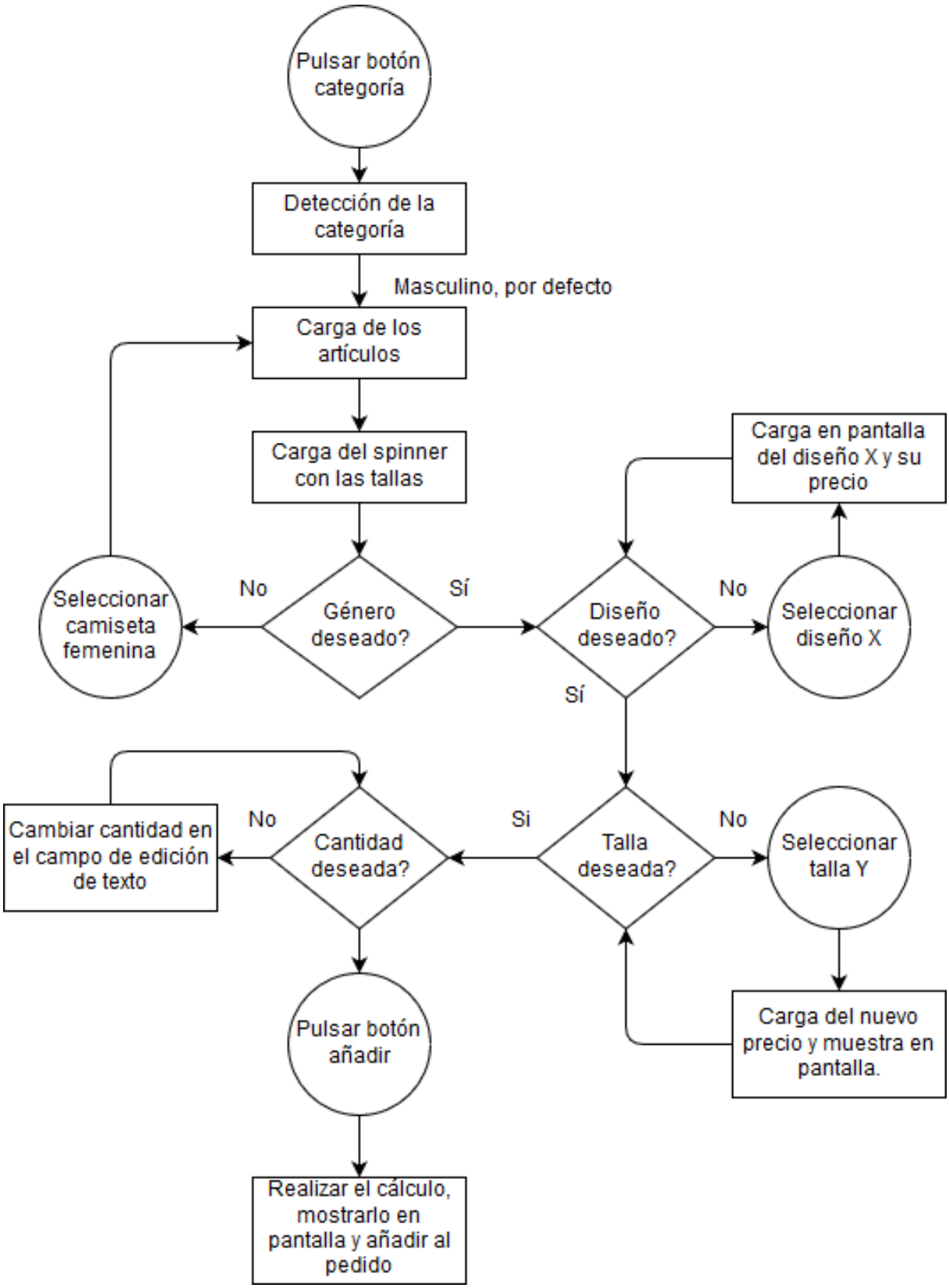
- Radio botones tipo de camiseta: Pese a ser elementos de control pasivo (se marcan y se comprueban posteriormente que están marcados con otra acción), se les puede implementar métodos de control activo, como por ejemplo cuando se hace click en ellos, se marcan, se desmarcan, o se pasa el dedo por encima. En este caso, se ha implementado el método **onClickListener**, cuya finalidad es realizar acciones cuando se pulsa para activarlo. Si se pulsa el femenino, cargará las fotos del catálogo femeninas, el spinner de tallas adecuado y sus precios. Lo mismo sucederá con la variante masculina cuando se pulse el botón apropiado.
- Spinner de talla: El spinner tendrá un contenido u otro dependiendo del tipo de camiseta que esté cargado. Éste también tiene implementado un método de control activo para su pulsación, el cual detectará el elemento pulsado, consultará su precio en la base de datos y lo mostrará en pantalla.

- Deslizante de modelos: Cargará los modelos de la categoría seleccionada. Cuando se pulse uno de ellos, se consultará si el tipo de camiseta activo es femenino o masculino, y cargará la miniatura correspondiente.
- Botón añadir: Consultará el tipo de camiseta activo y el campo que indica la cantidad de artículos que se desea añadir al pedido, posteriormente comprobará la talla que hay activa en el spinner y realizará el cálculo del precio para mostrarlo en un pequeño mensaje flotante .Por último, añadirá la selección a la base de datos y al pedido.

La finalidad de todo esto es dejar las acciones lo más naturales y en el menor número de pasos posibles. Todo el trabajo que pueda hacer una clase, será trabajo que el usuario final no tendrá que hacer.

Una tarea tan sencilla para el usuario cómo:

Internamente, se realizan una serie mucho mayor de acciones, que darán una mejor experiencia de uso.



Como existe una clase por actividad, la aplicación al completo suma el siguiente listado de clases destinadas a manipular actividades:

- ArchivoPedidos

- EditorCliente
- EditorComercial
- EditorPedido
- Galeria
- MainActivity
- MenuAjustes
- SelectorClase
- SelectorCliente

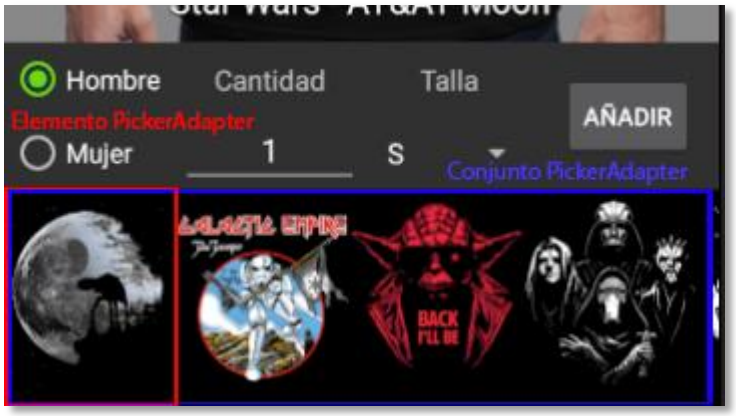
3.3.3.3 CLASES ESPECIALES

Las clases que se explican continuación, son clases que se han desarrollado explícitamente para una tarea concreta. Algunas de ellas van asociadas a una plantilla, pero no

conforman una actividad como tal, sino que tienen la responsabilidad de que el resto de actividades funcionen.

De este grupo, las siguientes tendrían una función e irían asignadas a una correspondiente plantilla (aunque es una plantilla menor, ideada para completar una función en concreto de una actividad

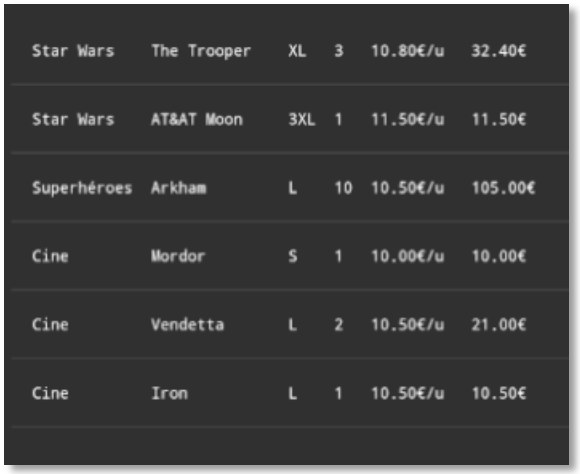
- **PickerAdapter y PickerAdapterH**: Ambas son la misma clase, salvo por que están ideadas cada una para una orientación diferente en la pantalla. Esta clase se encarga de poblar el deslizador en el que el usuario selecciona el diseño elegido y el sentido en el que se moverá. Parte de una plantilla con una única imagen por elemento (aunque se le pueden añadir más componentes, al final se decidió sólo dejar la imagen, ya que el texto se mostraba dos veces en pantalla y daba un efecto demasiado redundante)



42. Vista del conjunto de la clase PickerAdapter

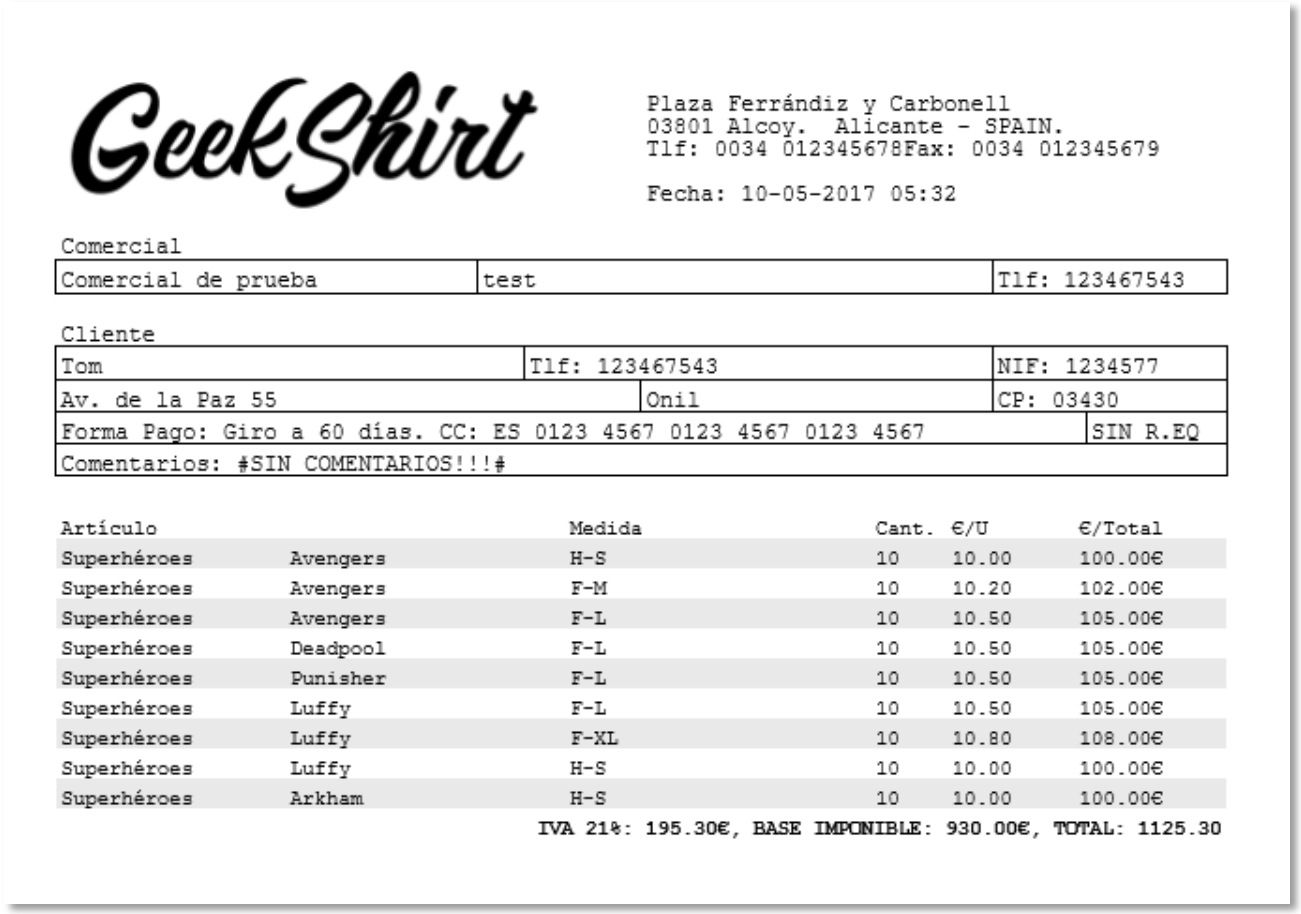
- **MonoSpaceAdapter**: Se implementó para la que la vista de los pedidos saliese monoespaciada y más legible (del mismo modo que **PickerAdapter**, se utiliza como

elemento para poblar una lista). Se compone de la clase y una pequeña plantilla en la que sólo existe una etiqueta de texto con una fuente monoespaciada y debidamente formateada.



43**. Ejemplo de lista poblada por MonoSpaceAdapter**

- **PDFCreator**: Esta clase tiene una gran importancia, ya que es la que se encarga de generar la hoja de pedido en PDF al finalizar el mismo, además de generar el reporte de pedidos y el reporte de clientes. Trabaja con la librería externa iTextPDF y tiene una sencilla configuración. En este caso particular, en primer lugar, se detecta qué tipo de función se va a realizar (pedido o reporte), y posteriormente se formatea el documento según la opción elegida y se procede al volcado de datos. Por último, se guarda el documento PDF en la memoria del dispositivo e y se inicia una nueva actividad, en este caso será el lector de PDF establecido por defecto en el sistema Android, mostrando el documento que acaba de generar, desde el cual se podrá visualizar para comprobar que está correctamente generado y enviarlo/compartirlo mediante las aplicaciones de comunicación vía e-mail.



GeekShirt

Plaza Ferrándiz y Carbonell
03801 Alcoy. Alicante - SPAIN.
Tlf: 0034 012345678Fax: 0034 012345679

Fecha: 10-05-2017 05:32

Comercial

Comercial de prueba	test	Tlf: 123467543

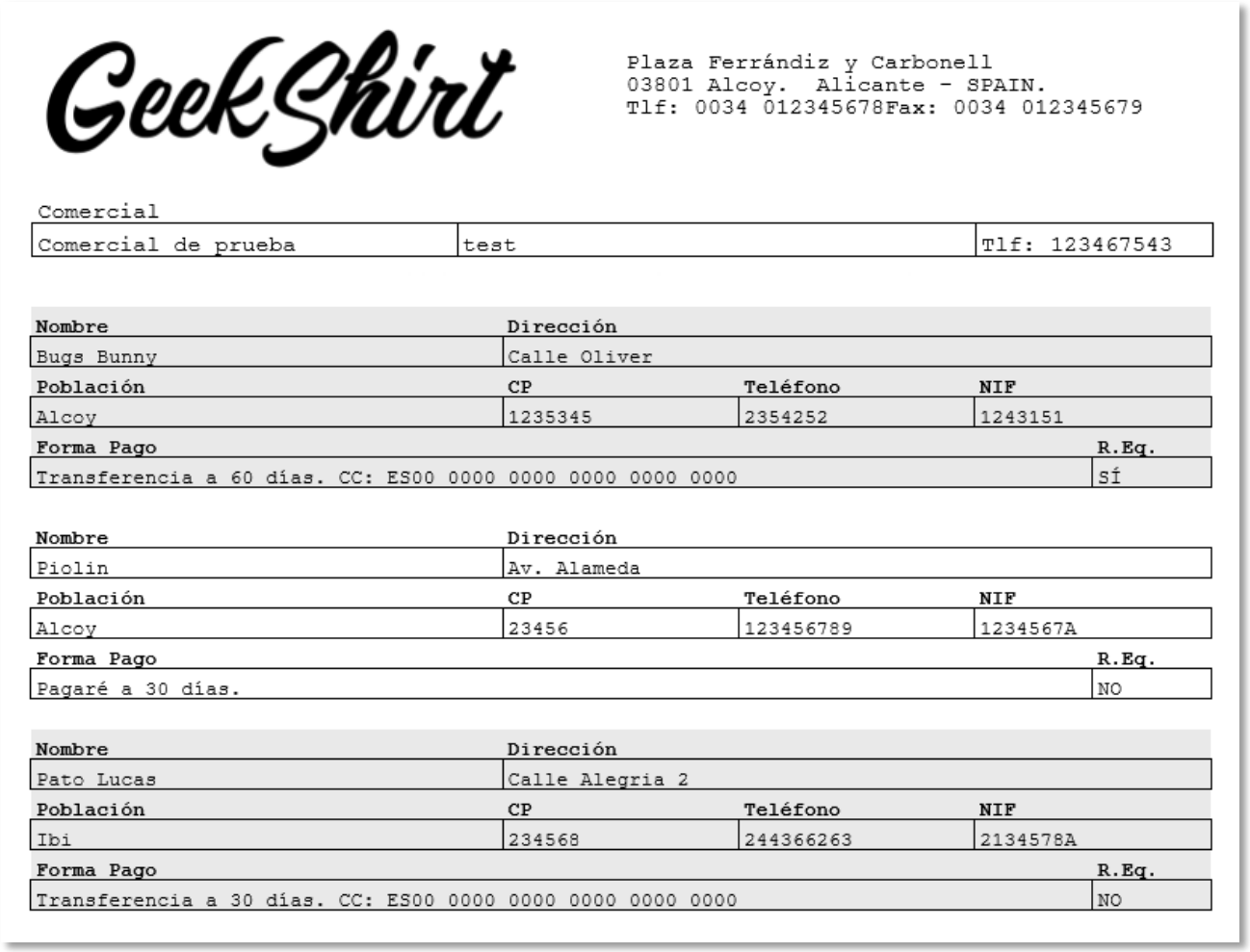
Cliente

Tom	Tlf: 123467543	NIF: 1234577
Av. de la Paz 55	Onil	CP: 03430
Forma Pago: Giro a 60 días. CC: ES 0123 4567 0123 4567 0123 4567		SIN R.EQ
Comentarios: #SIN COMENTARIOS!!!#		

Artículo		Medida	Cant.	€/U	€/Total
Superhéroes	Avengers	H-S	10	10.00	100.00€
Superhéroes	Avengers	F-M	10	10.20	102.00€
Superhéroes	Avengers	F-L	10	10.50	105.00€
Superhéroes	Deadpool	F-L	10	10.50	105.00€
Superhéroes	Punisher	F-L	10	10.50	105.00€
Superhéroes	Luffy	F-L	10	10.50	105.00€
Superhéroes	Luffy	F-XL	10	10.80	108.00€
Superhéroes	Luffy	H-S	10	10.00	100.00€
Superhéroes	Arkham	H-S	10	10.00	100.00€

IVA 21%: 195.30€, BASE IMPONIBLE: 930.00€, TOTAL: 1125.30

44.Ejemplo de PDF de pedido generado con iTextPDF



GeekShirt

Plaza Ferrándiz y Carbonell
03801 Alcoy. Alicante - SPAIN.
Tlf: 0034 012345678Fax: 0034 012345679

Comercial

Comercial de prueba	test	Tlf: 123467543

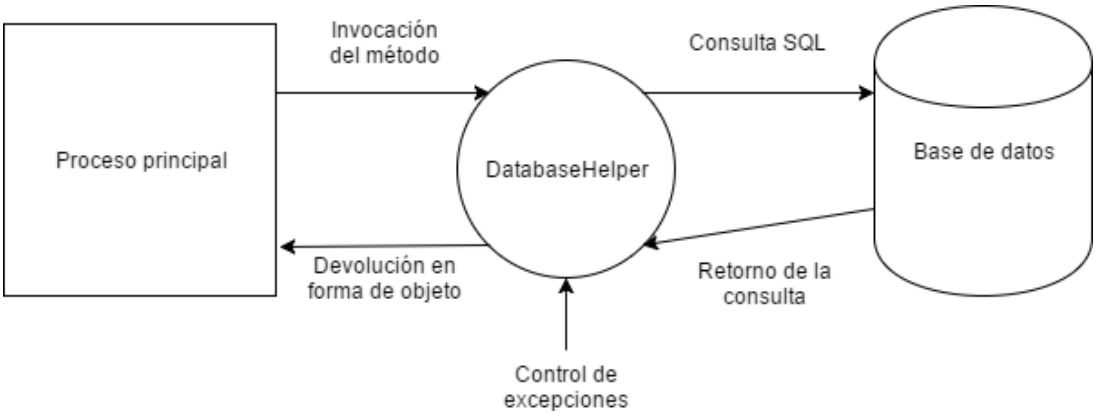
Nombre	Dirección		
Bugs Bunny	Calle Oliver		
Población	**CP**	**Teléfono**	**NIF**
Alcoy	1235345	2354252	1243151
Forma Pago			**R.Eq.**
Transferencia a 60 días. CC: ES00 0000 0000 0000 0000 0000			SÍ

Nombre	Dirección		
Piolin	Av. Alameda		
Población	**CP**	**Teléfono**	**NIF**
Alcoy	23456	123456789	1234567A
Forma Pago			**R.Eq.**
Pagaré a 30 días.			NO

Nombre	Dirección		
Pato Lucas	Calle Alegria 2		
Población	**CP**	**Teléfono**	**NIF**
Ibi	234568	244366263	2134578A
Forma Pago			**R.Eq.**
Transferencia a 30 días. CC: ES00 0000 0000 0000 0000 0000			NO

45.Ejemplo de reporte de clientes generado por la clase PDFCreator

- **DatabaseHelper**: Esta clase es la que se encarga de comunicar la aplicación con la base de datos. Cada instancia de esta clase, genera una conexión y permite hacer las peticiones que se han programado exclusivamente para cada función de la aplicación. Desde consultar el pedido activo, la información del comercial, devolver una lista de artículos de una categoría determinada o consultar los pedidos archivados entre otros. Una vez finalizadas las consultas la base de datos se cierra.



46.Diagrama de funcionamiento de la clase DatabaseHelper

La clase gestiona todo el proceso, desde la invocación del método hasta la devolución del objeto solicitado y el control de la excepción si lo hubiese. Como ejemplo, el método **devolverPedidosArchivados** el cual devuelve una **lista** con todos los pedidos archivados:

```
//Devolver TODOS los PEDIDOS archivados (!=0)
public ArrayList<Pedido> devolverPedidosArchivados(){
    SQLiteDatabase db = this.getReadableDatabase();                    Conexión con la BBDD
    ArrayList<Pedido>  alp = new ArrayList<>();                        Inicialización de la lista (vacía)
    String selectQuery = "SELECT * FROM " + TABLA_PEDIDO + " WHERE " + KEY_ID + " > 1";   Formación de la sentencia SQL
    Cursor c = db.rawQuery(selectQuery, null);                         Cursor que recorre los datos obtenidos
    if (c.moveToNext()) {                                              Si existen filas con datos
        do {
            Pedido pedidoTemp = new Pedido(                            Creamos un objeto Pedido con los datos
                    c.getInt(c.getColumnIndex(KEY_ID)),
                    c.getString(c.getColumnIndex(KEY_FECHA_PEDID)),
                    c.getInt(c.getColumnIndex(KEY_ID_COMERCIAL)),
                    c.getInt(c.getColumnIndex(KEY_ID_CLIENTE))
            );
            alp.add(pedidoTemp);                                       Añadimos el objeto a la lista
        } while (c.moveToNext());                                      Repetimos la operación si existen más filas
    }
    db.close();                                                        Cerramos la conexión
    return alp;                                                        Devolvemos la lista
}
```

11.Explicación del funcionamiento del método devolverPedidosArchivados de la clase DatabaseHelper

Como éste, existen otros 24 métodos que se encargan de todas las operaciones. A continuación, se citan con una descripción de su funcionamiento:

- obtenerGamaPorID(int id)
  ENTRADA: Número entero
  SALIDA: Objeto Gama
  FUNCIÓN: Devuelve la gama que tiene el identificador “id”
- obtenerProductosPorGama(int id)
  ENTRADA: Número entero
  SALIDA: Lista de objetos Producto
  FUNCIÓN: Devuelve una lista con todos los productos pertenecientes a la gama “id”
- obtenerProductoPorID(int id)
  ENTRADA: Número entero
  SALIDA: Objeto ProductoFUNCIÓN:
  Devuelve el producto con identificador “id”
- obtenerListadoClientes()
  ENTRADA: NADA
  SALIDA: Lista de objetos Cliente
  FUNCIÓN: Devuelve una lista con todos los clientes existentes en la base de datos
- obtenerClientePorID(int id)
  ENTRADA: Número entero
  SALIDA: Objeto Cliente
  FUNCIÓN: Devuelve un objeto Cliente con los datos del cliente con el identificador “id”
- crearNuevoCliente(Cliente cliente)
  ENTRADA: Objeto Cliente
  SALIDA: Número Entero
  FUNCIÓN: Inserta un nuevo cliente en la base de datos, devuelve el número de identificación que se le ha asignado automáticamente
- actualizarCliente(Cliente cliente)
  ENTRADA: Objeto Cliente
  SALIDA: Número Entero
  FUNCIÓN: Actualiza los datos del cliente relacionado con el objeto Cliente introducido, devuelve el número de identificación
- obtenerClientePorNombre(String nombre)
  ENTRADA: Cadena de texto
  SALIDA: Objeto Cliente
  FUNCIÓN: Devuelve el un objeto Cliente con los datos del primer cliente cuyo

nombre coincida con la cadena de texto introducida, es una función hecha adrede para mejorar el funcionamiento de una actividad usada en casos muy particulares ya que los datos devueltos podrían no tener consistencia (si hubiesen dos clientes con el mismo nombre, por ejemplo)

- crearNuevoComercial(Comercial comercial)
  ENTRADA: Objeto Comercial
  SALIDA: Número Entero
  FUNCIÓN: Establece los datos del comercial asignado a la aplicación y devuelve el número de identificación.
- actualizarComercial(Comercial comercial)
  ENTRADA: Objeto Comercial
  SALIDA: Número Entero
  FUNCIÓN: Actualiza los datos del comercial actual, devuelve el número de identificación
- actualizarDescuento(int descuento)
  ENTRADA: Número Entero
  SALIDA: Número Entero
  FUNCIÓN: Actualiza el valor descuento del comercial activo. Devuelve el nuevo valor de descuento a modo de verificación
- checkComercial()
  ENTRADA: NADA
  SALIDA: Objeto Comercial
  FUNCIÓN: Devuelve el comercial actual como objeto Comercial
- crearNuevoPedido(Pedido pedido)
  ENTRADA: Objeto Pedido
  SALIDA: Número Entero
  FUNCIÓN: Crea un nuevo pedido en la base de datos, devuelve el número de identificación
- modificarPedido(Pedido pedido)
  ENTRADA: Objeto Pedido
  SALIDA: Número Entero
  FUNCIÓN: Actualiza el pedido de la base de datos con los valores del pedido introducido
- devolverPedidoActual()
  ENTRADA: NADA
  SALIDA: Objeto Pedido
  FUNCIÓN: Devuelve el pedido que el programa considera como activo (id = 0)

- devolverPedidoPorID(int id)
  ENTRADA: Número entero
  SALIDA: Objeto Pedido
  FUNCIÓN: Devuelve el pedido con identificador “id” en la base de datos
- borrarPedido(Pedido pedido)
  ENTRADA: Objeto Pedido
  SALIDA: NADA
  FUNCIÓN: Borra el pedido relacionado con el objeto introducido como parámetro
- archivarPedido(Pedido pedido)
  ENTRADA: Objeto Pedido
  SALIDA: Número Entero
  FUNCIÓN: Archiva el pedido activo (id = 0), asignándole un nuevo identificador, el cual devuelve.
- devolverPedidosArchivados()
  ENTRADA: NADA
  SALIDA: Lista de objetos Pedido
  FUNCIÓN: Devuelve todos los pedidos marcados como archivados
- anyadirOrden(Orden orden)
  ENTRADA: Objeto Orden
  SALIDA: NADA
  FUNCIÓN: Añade una orden de producto y cantidad al pedido, sumando estos si se repiten.
- modificarCantidadOrden(Orden orden)
  ENTRADA: Objeto Orden
  SALIDA: NADA
  FUNCIÓN: Modifica la cantidad de un producto en una orden.
- borrarOrden(Orden orden)
  ENTRADA: Objeto Orden
  SALIDA: NADAFUNCIÓN: Borra la orden introducida, eliminándola del pedido
- devolverOrdenesPedido(Pedido pedido)
  ENTRADA: Objeto Pedido
  SALIDA: Lista de Ordenes
  FUNCIÓN: Devuelve una lista con todas las órdenes relacionadas con el objeto pedido introducido
- archivarOrdenes(int id_pedido)
  ENTRADA: Número Entero
  SALIDA: Nada

FUNCIÓN: Método auxiliar para la función de archivar un pedido. Archiva las órdenes del pedido con “id” introducido

- devolverPrecioProducto(Producto producto, String medida)
  ENTRADA: Objeto Producto, Cadena de texto
  SALIDA: Número decimal
  FUNCIÓN: Devuelve el precio del producto introducido y la variación “medida” introducida.

Mediante este conjunto de métodos, se logra tener un control de la base de datos sin dar lugar a situaciones inesperadas, ya que se tienen la totalidad de los datos archivados y gestionados adecuadamente. Quedan pues explicadas las clases, actividades y diferentes funciones que conforman la totalidad de la aplicación.

4.Coste de proyecto

	TAREAS DEL PROYECTO	HORAS MANO OBRA	COSTO MANO OBRA (€)
DISEÑO DEL PROYECTO	Desarrollar especificaciones funcionales	24.0	288.00 €
	Reuniones para implementar el diseño	21.0	252.00 €
	Análisis del caso y toma de decisiones	12.0	144.00 €
	Diseño de la base de datos	10.0	120.00 €
	Subtotal	**67.0**	**804.00 €**
DESARROLLO DEL PROYECTO	Implementar base de datos	18.0	216.00 €
	Puesta a punto del equipo y entorno	5.0	60.00 €
	Desarrollo de software	145.0	1,740.00 €
	Evaluación y pruebas	42.0	504.00 €
	Formación	22.0	264.00 €
	Subtotal	**232.0**	**2,784.00 €**
ENTREGA DEL PROYECTO	Instalar la aplicación en dispositivos	20.0	240.00 €
	Explicar funcionamiento a los comerciales	30.0	360.00 €
	Realizar prueba de aceptación	15.0	180.00 €
	Realizar revisión posterior al proyecto	24.0	288.00 €
	Subtotal	**89.0**	**1,068.00 €**
ADMINISTRACIÓN DEL PROYECTO	Reuniones/informes de progreso con el cliente	12.0	144.00 €
	Administración de la configuración	10.0	120.00 €
	Control de calidad	28.0	336.00 €
	Subtotal	**50.0**	**600.00 €**
Total		**438.0**	**5,256.00 €**

5.Conclusiones y propuestas de trabajo futuro

5.1 DISCUSIÓN PERSONAL

Tras completar la aplicación y ver la funcionalidad la misma quedé muy satisfecho pese al reto que ha supuesto. Para alcanzar los diferentes puntos he tenido que llevar a cabo una formación continua durante el desarrollo de la aplicación para evitar quedar estancado el mínimo tiempo posible, además de que personalmente es el método con el que más conocimientos y práctica adquiero al mismo tiempo. Este modo de formación también me ha ayudado a poder comprender otros puntos de vista, ya que no todo el mundo programa de la misma manera y una misma acción se puede explicar de diferentes formas, por lo que el resultado suele ser una selección de las (que considero) mejores ideas y en muy pocas ocasiones me he quedado atascado en un problema y he podido avanzar.

Haber estudiado mayormente JAVA durante el largo recorrido del grado ha sido de vital ayuda para poder usar la herramienta de Android Studio. Aunque se pueda con otros lenguajes, me he sentido realmente cómodo y no he tenido problemas con el entorno de desarrollo. El resto del abanico que engloban las asignaturas de la programación y sus paradigmas han sido también causantes de que el código tenga una estructura aceptable, con código reutilizable y una buena gestión de los errores que puedan aparecer.

El proyecto ha surgido de una experiencia que he tenido durante las prácticas de empresa el cual me dio la idea de hacer este tipo de aplicación. Las mismas prácticas han sido una experiencia muy afortunada al haberme brindado esta oportunidad y ampliar mis límites con JAVA y Android, ya que el proyecto actual es perfectamente ampliable y modificable de cara a crear variaciones o ampliaciones del mismo.

5.2 CONCLUSIONES

- **Objetivo principal** El objetivo principal se ha logrado, ya que la aplicación resultante del proyecto es completamente funcional.

- **Utilidad** La aplicación cumple con el cometido para el que fue diseñada: Mostrar un catálogo interactivo de unos productos determinados, y permitir a un agente comercial realizar pedidos desde el mismo catálogo, ahorrando tiempo, costes y facilitando su labor no teniendo que portar consigo los catálogos físicos impresos en papel y los demás medios para la realización del pedido

- **Robustez** También se ha conseguido cumplir con las exigencias de la tolerancia a fallos y la alta disponibilidad, portando la base de datos de manera local, haciendo la aplicación utilizable sin cualquier tipo de conexión a internet.

- **Facilidad de uso** Se ha conseguido que la aplicación sea sencilla de utilizar con una curva de aprendizaje muy corta (realizar un pedido de prueba suele ser más que suficiente para alguien que nunca ha usado la aplicación), y se han reducido en la medida de lo posible los pasos a realizar.

- **Portabilidad** El catálogo es fácilmente personalizable por cualquier empresa que quiera utilizarlo, tan fácil como introducir los logotipos, aplicar un color diferente si se desea a la aplicación, y dar de alta todos los productos con sus variaciones y características. Quitando estos datos propios de la empresa, el resto de la aplicación es completamente independiente de estos datos, por lo que puede emplear en cualquier empresa y artículo

5.3 PROPUESTAS DE TRABAJO FUTURO

Gracias a la portabilidad anterior mencionada, el proyecto queda abierto como un punto de inicio para cualquier idea que pueda crecer de él. Como ideas a realizar próximamente, se encuentra la de agregar la opción de sincronizar la base de datos con una remota y, gracias a esta función, poder controlar temas como el stock de los artículos e incluso actualizar los mismos, aplicar ofertas y promociones, etc. de manera remota.

También está en el aire la posibilidad de migrar la aplicación a iOS, para poder usarla en dispositivos Apple, aunque requiere una inversión grande tanto en equipos como en la formación pertinente para realizar el proyecto en esta plataforma.

Por último, mientras la aplicación esté activa, está implícito que habrá que ir adaptándola a las diferentes versiones de Android que aparezcan en el mercado y se tenga la necesidad de utilizar esta aplicación en ella.

5.4 AGRADECIMIENTOS

En primer lugar, agradecer a mis padres el haberme dado la oportunidad y las facilidades para poder cursar este grado, ya que sin ellos ni su esfuerzo nada de esto habría sido posible.

A mis compañeros, que también han hecho de profesores en horarios especiales durante los cuatros años que ha durado este viaje, y algunos han sido culpables de que haya sabido defenderme con Android y haber podido empezar este proyecto.

También a mis profesores, por haber conseguido que un técnico superior en hostelería haya despertado interés por la programación y el hardware, y haya hecho de ello a día de hoy su profesión.

Finalmente, pero no por ello menos importante, a Óscar, por haber aceptado guiarme durante este trabajo de final de grado, ayudándome con ello a finalizar el grado y aportándome una gran ayuda para el desarrollo del proyecto y la documentación.

BIBLIOGRAFÍA

1. https://developer.android.com/
2. https://docs.oracle.com
3. https://stackoverflow.com/
4. https://sqlite.org/docs.html
5. Apuntes de programación JAVA (1º)
6. Apuntes de interfaces (2º)
7. https://www.w3schools.com/sql/
8. https://helpx.adobe.com/
9. https://github.com/explore

ÍNDICE DE MINIATURAS