Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación Android con tablón de anuncios ordenados enfocado a la impresión 3D

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ruslan Kyrch

Tutor: David de Andrés Martínez

[2016/2017]

Aplicación Android con tablón de anuncios ordenados enfocado a la impresión 3D

Resumen

Diseño e implementación de aplicación Android destinada para poner en contacto propietarios de impresoras 3D con usuarios que precisen de estas para sus impresiones. Los dueños de las impresoras podrán crear anuncios ofreciendo información de su impresora, detalles técnicos, imágenes de sus diseños impresos e información de contacto. Los usuarios de la aplicación que deseen dar forma a sus diseños 3D podrán navegar en el tablón principal donde estarán los anuncios ordenados por cercanía.

Este proyecto surge ante la escasez de "copisterías 3D", se vio una oportunidad de negocio después de realizar un análisis de mercado a partir de aplicaciones de ámbito similar, Milanuncios, Wallapop, Imprimalia 3D etc.

Palabras clave: Android, aplicación, impresora 3D, diseño 3D.

Abstract

Design and implementation of an Android application designed to connect owners of 3D printers with users who need these for their impressions.

Printer owners will be able to create announcements by offering information about their printer, technical details, images of their printed designs and contact information. Users of the application who wish to shape their 3D designs will be able to navigate on the main board where the ads will be sorted by proximity.

This project arises in the absence of "3D printing", saw a business opportunity after conducting a market analysis from applications of similar scope, Milanuncios, Wallapop, Imprimalia 3D etc.

Keywords : Android, application, 3D printer, 3D design.

Aplicación Android con tablón de anuncios ordenados enfocado a la impresión 3D

Tabla de contenido

Índice de Figuras

Aplicación Android con tablón de anuncios ordenados enfocado a la impresión 3D

1. Introducción

En los últimos años, la impresión 3D va adquiriendo cada vez más importancia en nuestras vidas y la vemos más presente a nuestro alrededor. Los enormes avances tecnologías en cuanto a hardware, software y materiales marcan el camino a un futuro prometedor a esta tecnología [1].

Aunque la enorme presencia actual de la impresión tridimensional podría hacernos pensar en que es un invento reciente, realmente no es así, nació en 1984, año en que Chuck Hull creó la primera impresora tridimensional, poco después fundó 3D Systems y desarrolló la primera impresora tridimensional comercial, basada en la estereolitografía (SLA). Desde aquel entonces y a través de múltiples nuevas técnicas de impresión que fueron surgiendo llegamos al 2005, donde dos momentos fueron los encargados de que la impresión 3D sea, hoy en día, tan conocida. Por un lado, la aparición de la impresora Spectrum Z510 de Z Corporation, se trata de la primera impresora 3D de alta resolución y a todo color del mercado.

La aparición de esta impresora dio pie a ver este tipo de dispositivos no solo como hardware para el diseño de prototipos, sino también de piezas finales personalizadas, que no solo tenían cabida en un ambiente de fabrica o empresarial, sino también para profesionales y consumidores finales. Con ellos comenzó el uso de la impresión con fines artísticos, lúdicos y caseros, por los que servicios de impresión 3D empezaron a desarrollarse aún más debido al aumento de demanda.

Este aumento en la demanda y el querer tener más control sobre la creación de piezas, añadido al auge de la filosofía de software libre, y recientemente también de hardware libre, llevaron a un punto de inflexión, el nacimiento del **proyecto RepRap.** El objetivo principal de este proyecto dirigido por Adrian Bowyer era crear una impresora 3D *open source* que pudiera autoreplicarse, es decir, el poder producir con esta impresora piezas para construir una impresora igual o incluso de mejores prestaciones. La importancia de este proyecto para explicar el enorme desarrollo actual es grandísima, ya que no sólo sentó las bases de la gran mayoría de impresoras 3D que existen hoy en día, sino que también obligó a la industria a llevar a cabo una reducción en costes y precios.

Justamente debido a esta popularidad de las impresoras 3D en la actualidad, vi una oportunidad de negocio que no estaba explotada. Una aplicación que permita a todos aquellos curiosos por esta tecnología experimentar y dar forma a sus proyectos, pero que no tengan la oportunidad de realizar un desembolso económico tan grande solo por hobby o que quieran darle un uso meramente puntual, esta les permitirá contactar con propietarios de impresoras más cercanos que quieran compartir sus impresoras para sacarles más rentabilidad o simplemente darlas en alquiler.

Para el desarrollo de esta idea se decidí empezar por el análisis de la viabilidad, donde compararía la aplicación a sus competidores más directos (estado del arte), una vez enumeradas las funcionalidades en las que se asentaría la aplicación para destacar frente a la competencia más directa pasé a plantear el tipo de monetización que se aplicaría para obtener rentabilidad de esta (modelo de negocio / monetización). Investigue en tecnologías actuales que podría aplicar en la aplicación para agilizar el desarrollo y conseguir las funcionalidades que buscaba, como

MongoDB, Firebase, y distintas librerías Android (especificación), teniendo claro estos puntos pase a realizar un primer diseño de la arquitectura global y empezar a programar.

2. Estado del Arte

En este apartado analizaré la principal competencia de la aplicación, sus funcionalidades, sus pros y sus contras, en que podemos inspirarnos y donde tendríamos que aprender de sus errores.

Wallapop

Wallapop (Figura 1) es una aplicación de compraventa de artículos usados o de segunda mano. En los últimos años se ha posicionado como el portal más reconocido de compraventa de este tipo de artículos, superando incluso a su predecesor Milanuncios.

Figura 1: Logotipo Wallapop

¡Su funcionamiento es muy simple, basta con hacer una foto al producto que queremos poner a la venta, escribir un título, añadir una descripción, especificar el precio y listo! Nuestro producto ya está a la venta en el portal (Figura 2).

Desde el punto de vista del comprador solo tenemos que escribir lo que queremos en el filtro de búsqueda y nos aparecerán numerosos anuncios en el tablón ordenados por proximidad o precio (Figura 4), donde podremos ir navegando entre ellos y contactar con el propietario del artículo que más nos interese, vía chat, directamente por la aplicación o a través de su información de contacto (Figura 3)

La principal fortaleza de Wallapop frente a su competencia del sector es la inexistencia de publicidad dentro de la aplicación o de comisiones a la hora de efectuar una venta de un artículo.

Figura 2: Formulario creación de anuncio. Wallapop

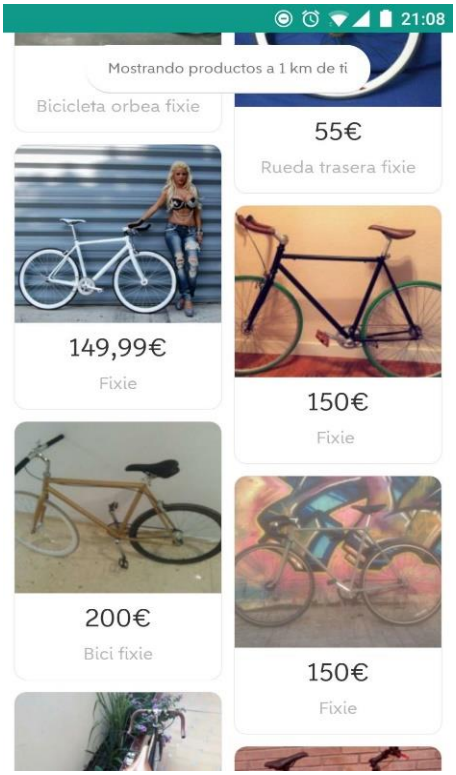

Figura 4: Tablon de anuncios.Wallapop

Figura 3: Anuncio ampliado

Llegados a este punto la pregunta es, ¿Cómo gana dinero Wallapop? Hasta hace un año no recibía ningún tipo de ingreso de la aplicación, pese a recibir millones de euros de financiación de inversores. Recientemente la compañía ha decidido implementar una plataforma de pagos propia llamada "Wallapay", que presenta una escala de comisiones dependiendo de la cantidad de dinero que se transfiera, e incluir publicidad en los anuncios.

Milanuncios

Milanuncios (Figura 5) es un portal de anuncios clasificados. Permite a los usuarios publicar anuncios de forma anónima y gratuita sin necesidad de ningún registro. También tiene un buscador para poder filtrar lo que buscas entre millones de anuncios, desde una guitarra de segunda mano hasta un inmueble, estos anuncios están divididos en categorías, motor, empleo, juegos, servicios…

Figura 5: Logotipo Milanuncios

Para publicar tu anuncio en la web tienes que seleccionar la categoría en la que quieres situarlo, añadir un título, imagen, descripción y un contacto (Figura 7), con esto ya tendríamos el anuncio publicado en el tablón de la web (Figura 6).

Figura 7: Formulario creación de anuncio

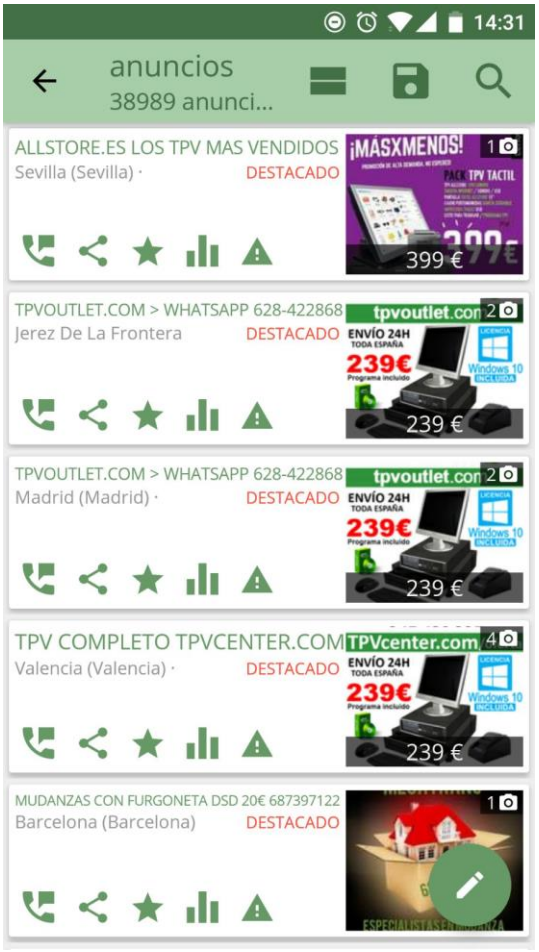

Figura 6: Tablón de anuncios. Milanuncios

El modelo de negocio de Milanuncios se basa en cobrar las renovaciones de los anuncios. Como he comentado antes, crear un anuncio es gratuito, sin embargo, al entrar en el tablón de anuncios principal o al hacer alguna búsqueda los primeros anuncios que aparecen son los últimos en haber sido creados o renovados, si quieres que tu anuncio sea visto por más gente puedes contratar la "auto-renovación", de forma que tu anuncio vuelva a ponerse el primero cada cierto tiempo.

Es decir, Milanuncios gana dinero cuando el anuncio **no** es efectivo, si un anuncio fuera efectivo a la primera, entonces no ganarían dinero porque nadie contrataría la auto-renovación.

Imprimalia 3D

Imprimalia 3D (Figura 8) es el primer portal en español y el más grande dedicado a la impresión 3D, fabricación aditiva y otras tecnologías asociadas.

Figura 8: Logotipo Imprimalia 3D

Imprimalia3D consta de una portada, con las noticias de actualidad más destacadas de la impresión 3D, avances tecnológicos, aplicaciones médicas, aeroespaciales, arquitectónicas etc (Figura 9). También tiene un apartado de "Comprador" donde puedes tanto comprar como comparar los precios de un amplio catálogo de impresoras 3D y componentes. Un blog donde encontrar una gran variedad de artículos de opinión, tutoriales para ayudarte en tus impresiones en casa y curiosidades.

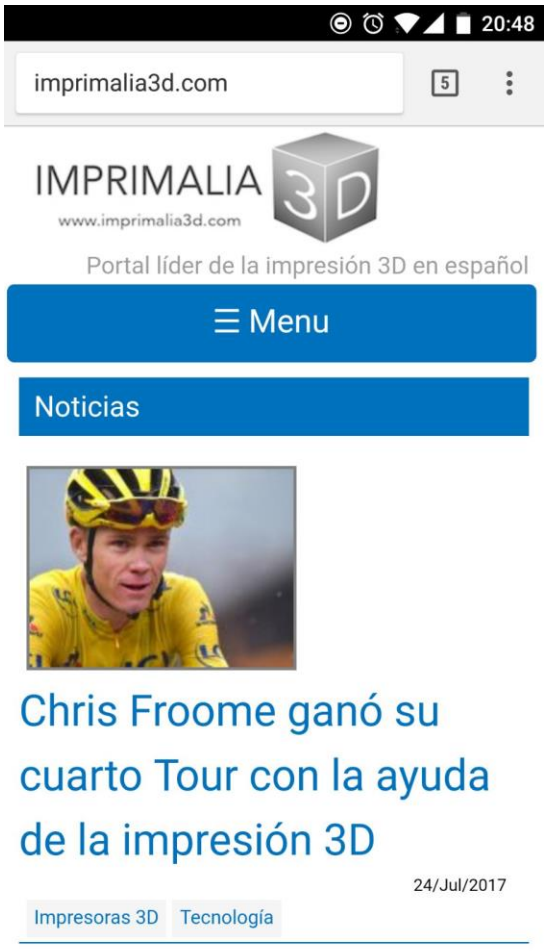

Figura 9: Página principal. Imprimalia 3D

Imprimalia se lucra a partir de los anuncios de empresas relacionadas al sector en su página web. En un banner lateral hay un sector dedicado a este fin (Figura 10). A parte de esto también recibe comisiones a través de los clicks en estos banners.

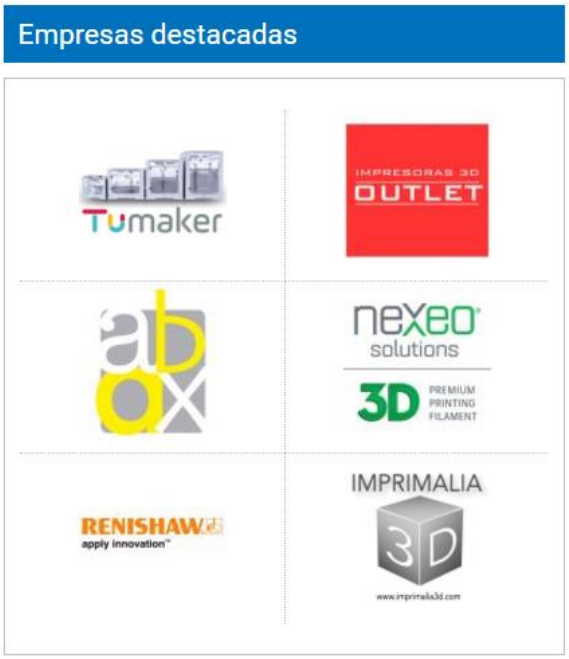

Figura 10: Banner de anuncios de empresas. Imprimalia 3D

A simple vista este último portal relacionado con la impresión 3D poco parece tener en común con la aplicación que venimos a desarrollar en este trabajo, sin embargo, aparte de crear una aplicación de compra venta de artículos de impresión, una herramienta para sacarle rentabilidad a tu impresora 3D a través del alquiler de esta o la impresión por encargo o un lugar donde darle forma a tus diseños, también pretende convertirse en una comunidad donde puedes compartir tus diseños, discutir temas de interés o conversar con personas con la misma afición.

A modo de resumen vamos a enumerar las diferentes características que comparten estas aplicaciones y portales y contrastarlas con las que queremos aplicar en nuestra aplicación.

Nombre	**Adquirir impresoras o complementos**	**Solicitar la impresión de un diseño**	**Ver diseños de otros usuarios**	**Abrir una conversación con otro usuario**
Imprimalia	Si	Si	No	no
Milanuncios	Si	Si	Si	Si
Wallapop	Si	Si	Si	Si
Shar3D	Si	Si	Si	Si

3. Metodologías utilizadas

En el desarrollo de esta aplicación se quiere conseguir un cierto de funcionalidades básicas las cuales se considerarán como la base de la aplicación y a partir de las cuales se podrá ampliar su funcionamiento, añadir nuevas funcionalidades o escalar el sistema para un uso más masivo. Como consecuencia, el objetivo principal de la programación de estas funciones es conseguir un código robusto, probado y escalable.

Teniendo en cuenta las premisas a las que me enfrento y las directrices que me he planteado, he decidido llevar a cabo este proyecto con una metodología de desarrollo ágil, ya que no tenía del todo claro todos los requisitos funcionales que deberían de estar presentes en la aplicación final y gracias a esta herramienta son más manejables y fáciles de implementar nuevas funcionalidades en fases más tardías o cambios en requisitos ya consensuados.

En toda metodología ágil el cliente es uno de los principales componentes. En este proyecto el profesor David de Andrés Martínez y un servidor compartiremos el rol de cliente, donde discutiremos los requisitos a implementar, las funcionalidades para revisar etc.

Metodologías ágiles del desarrollo del software

[2] Las metodologías agiles (Figura 11) son métodos de desarrollo de software en los que las necesidades y soluciones evolucionan a través de una colaboración estrecha entre el equipo de desarrollo y el cliente final. Se caracterizan por enfatizar la comunicación frente a la documentación, por el desarrollo evolutivo y su flexibilidad.

Estas metodologías surgen a principios del 2001 en respuesta a los modelos de proceso clásicos ya existentes. La aparición de procesos ágiles se debe a estas carencias presentes en los desarrollos precedentes:

1. Difícil predecir los requisitos que serán modificados en un futuro y los que persistirán en el software, así como las prioridades del cliente.
2. El diseño del software y su desarrollo están intercalados. Por ello se realizarán de forma incremental y conjunta, probando el diseño a medida que se crea.
3. El análisis, diseño y la implementación no son predecibles desde el punto de vista de la planificación.

Existe una denominada Alianza Ágil que define los siguientes 12 principios para toda metodología ágil:

1. **Satisfacer al cliente** con entregas tempranas y continuas de software valioso.
2. **Los requisitos cambiantes son bienvenidos**, incluso en fases tardías del desarrollo.
3. **Interacciones constantes**. Entregar con frecuencia software que funcione -de dos semanas a dos meses-.
4. **Trabajo colaborativo**. El cliente y el desarrollador deben de trabajar de forma conjunta en el proyecto.
5. **Motivación del equipo**. Construcción del proyecto en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
6. **Contacto directo con los clientes**. La forma más eficiente y efectiva de transmitir información hacia y dentro del equipo es la conversación cara a cara.
7. **Medida de progres**. El software que funciona es la principal medida de progreso.
8. **Desarrollo sostenido**. Los procesos ágiles promueven el desarrollo sostenido. Los participantes deben ser capaces de mantener un paso constante de manera indefinida.
9. **Búsqueda de la excelencia.** La atención continua a la excelencia técnica enaltece la agilidad.

10. **La simplicidad,** como arte de maximizar la cantidad de trabajo que se hace es esencia.
11. **Autorregulación.** Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.
12. **Revisión permanente.** En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajustar su conducta en consecuencia.

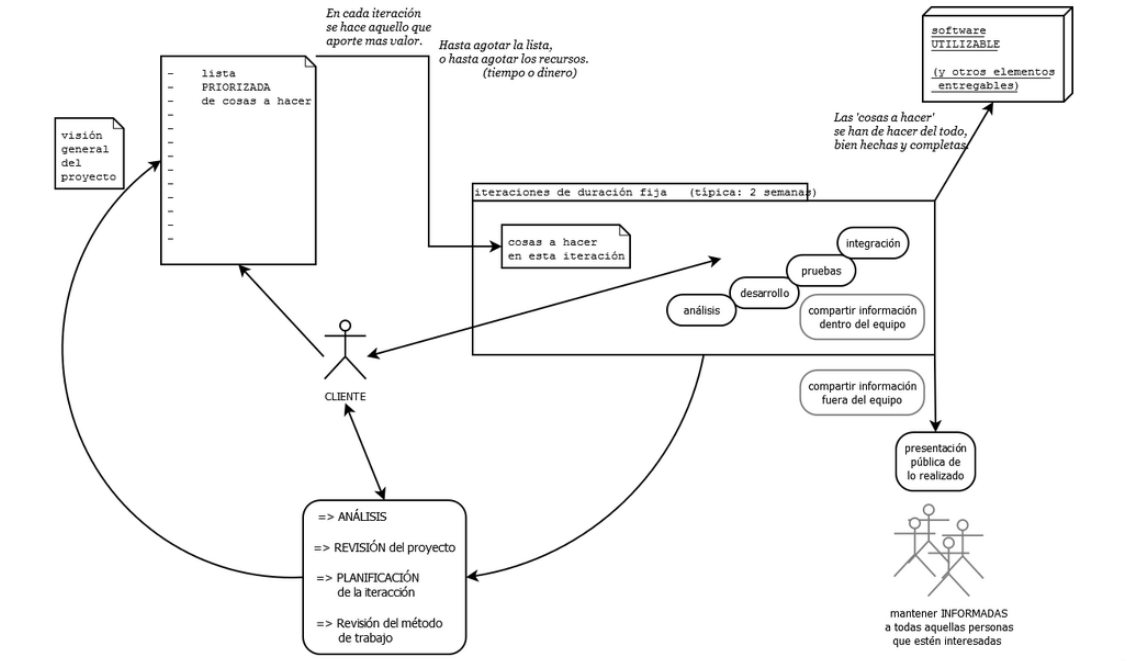

Figura 11: Esquema general de una metodología ágil para desarrollo de software

Existen diversos tipos de procesos agiles (XP, SCRUM, DSDM, FDD, ASD, Crystal…) donde todos comparten los principios del desarrollo de software ágil, sin embargo, nosotros nos vamos a decantar por el Scrumban.

Scrumban

Scrum (Figura 12) es un proceso ágil que nos permite centrarnos en ofrecer el más alto valor de negocio en el menor tiempo posible. Nos permite rápidamente, y en repetidas ocasiones, inspeccionar software real de trabajo (cada dos semanas o un mes). Los equipos se auto-reorganizan a fin de determinar la mejor manera de entregar las funcionalidades de más alta prioridad. Después de cada *sprint*, es posible ver un software real funcionando y decidir si liberarlo o seguir mejorándolo en el siguiente *sprint* [3].

Scrum se caracteriza por constar con equipos, los cuales cuentan con roles, artefactos y reuniones de forma que estas tres partes interaccionan continuamente entre ellas. Está basado en el conocimiento dado por la experiencia, por lo que todo lo que plantea debe ser tomado como un método para ir aprendiendo mientras se ejecuta, siempre buscando ver que es lo que funciona y lo que no para sacar el máximo rendimiento.

Scrum se compone de una lista de Requerimientos llamada *product backlog*, de esta lista se selecciona un número acotado de requerimientos, también llamados *historias de usuario*, este listado de requerimientos conforma el *sprint backlog*, el cual se usa durante un *sprint*. Un *sprint* es una iteración del ciclo, en ella se realiza una serie de reuniones y se desarrolla el software

según los requerimientos seleccionados. Este proceso se repite hasta que el cliente está satisfecho con el software creado, ya que cada Sprint entrega un incremento de software.

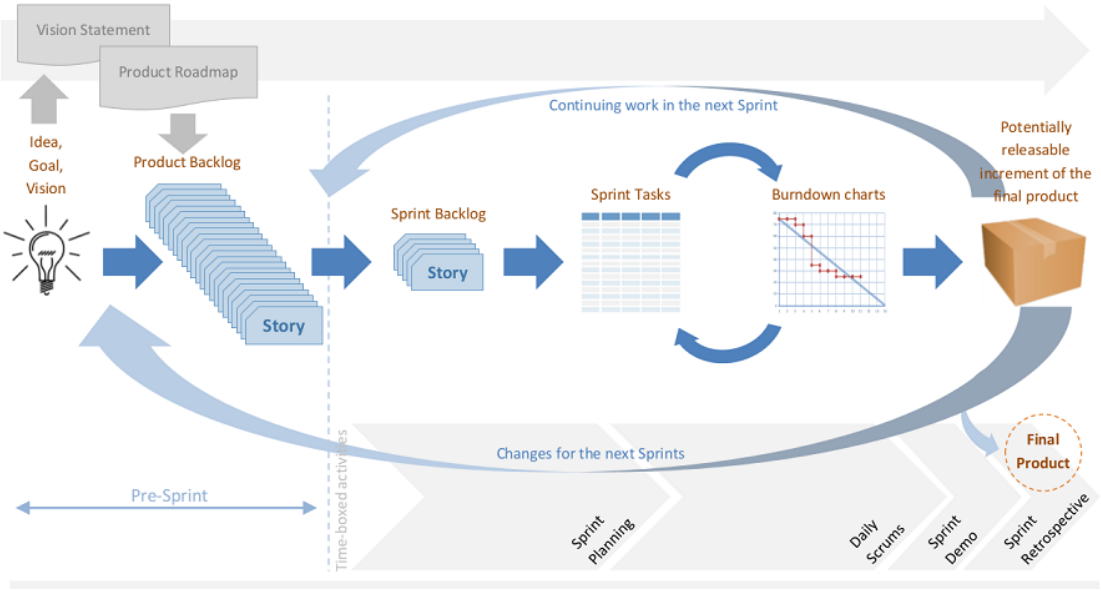

Figura 12: Ciclo Scrum

Kanban, por su parte, es un sistema de trabajo que proporciona un mejor flujo de trabajo al dividir un proceso productivo en varias fases perfectamente delimitadas (Figura 13).

Este método se rige por: lograr un producto de calidad, al obligar a cada fase del proyecto a finalizar su tarea correctamente, y acabar la saturación o cuello de botella que puede darse en una fase del proyecto en las que prima la rapidez por encima de la calidad del producto [4].

Los principios básicos o reglas que permiten al Kanban conseguir su propósito son cuatro:

1. **Empieza con lo que haces ahora.** El método Kanban se inicia con las tareas actuales y se estimulan los cambios.
2. **Acepta el cambio.** Tienes que estar dispuesto a aceptar cambios evolutivos en las tareas. Si algo no funciona, cámbialo.
3. **Respeta el proceso en curso, los roles, las responsabilidades y los cargos.** No se trata de que todos hagan todo, sino que cada cuál sepa qué hacer en el momento adecuado**.**
4. **Liderazgo en todos los niveles:** Se debe de incentivar hechos de liderazgo en todos los niveles.

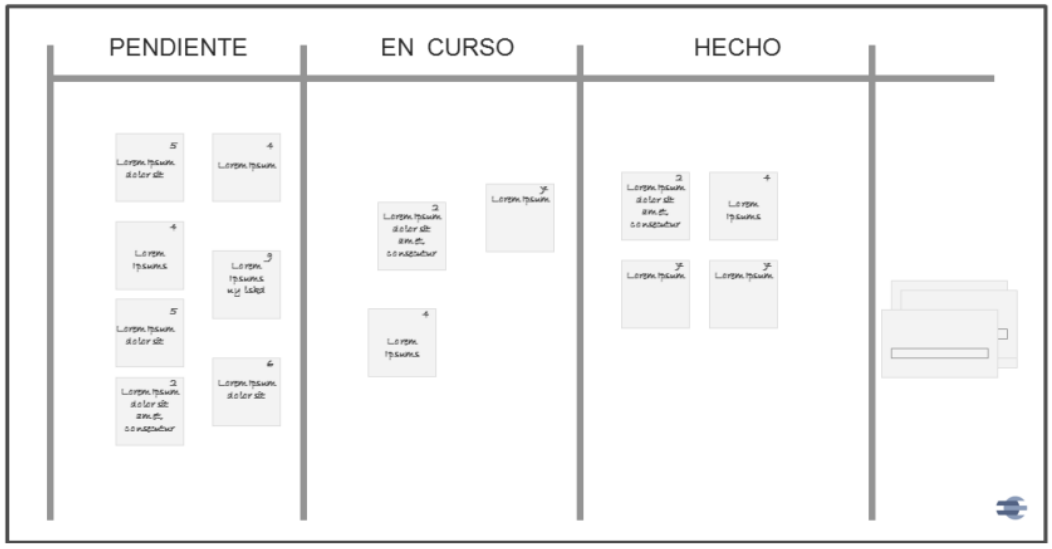

Figura 13: Tablero Kanban

Tanto Kanban como Scrum son metodologías ágiles. Scrumban combina las mejores características de ambos métodos, la naturaleza preceptiva de Scrum y la capacidad de mejora de Kanban, permitiendo a los equipos moverse hacia el desarrollo ágil y mejorar constantemente sus procesos.

Esta mezcla de metodologías nos permitirá tener reuniones diarias para dar soluciones a problemas que impactan sobre el desarrollo de un entregable, permite priorizar los entregables mediante reuniones con el líder usuario, permite presentar al usuario parte del producto cada sprint; así como también tener un control de cuantos *ítems* se encuentran en proceso, cuantos culminados y poder tener una organización visual de cómo estamos procesando los objetivos (Figura 14).

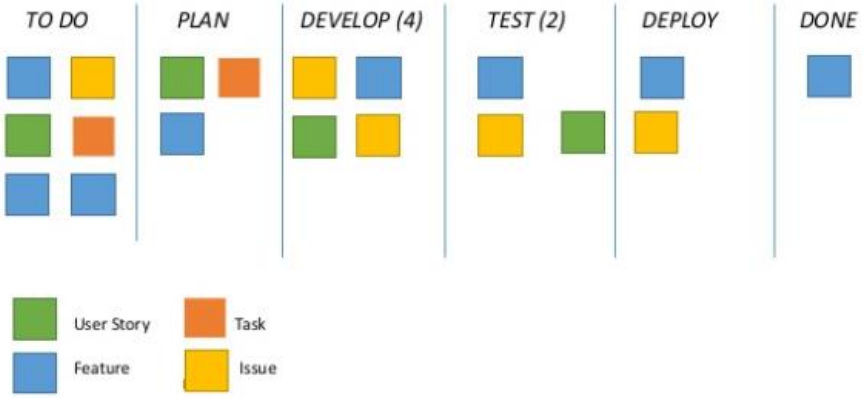

Figura 14: Tablero Scumban

4. Especificación de requisitos

A partir de las aplicaciones de la competencia y el valor añadido que le queremos dar para hacernos destacar, vamos a enumerar las funcionalidades que en primer momento queremos que estén presentes en nuestra aplicación final, algunas de ellas estarán sujetas a cambios,

eliminación o la incorporación de otras nuevas, debido a la opinión de los *product owner* a lo largo del desarrollo.

Representaremos estas funcionalidades a partir de casos de uso de un usuario común (Figura 15) ya que no tenemos otro tipo de roles en la aplicación, no he visto necesario añadir un rol de administrador ya que no se precisa de configuración externa por un usuario experto para el correcto funcionamiento de la aplicación o la gestión de esta. Sin embargo, no se descarta la posibilidad de añadirlo en un futuro ante la necesidad de controlar la monetización.

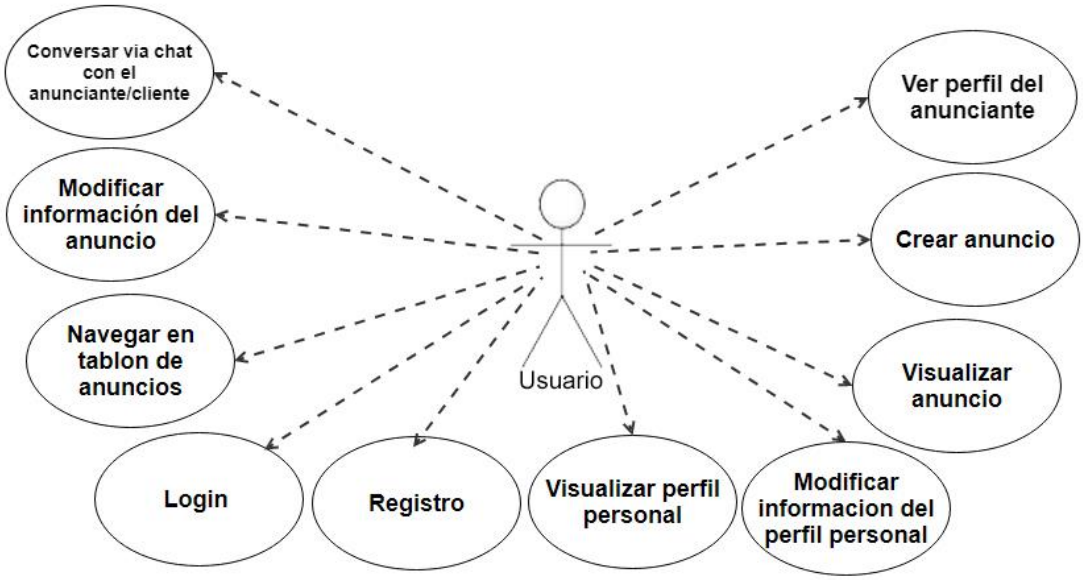

Figura 15: Diagrama de casos de uso.

- **Registro**: El usuario debe ser capa de registrarse en la aplicación a través de un formulario rellenando sus datos básicos, usuario, contraseña, correo electrónico y especificar desea crear el perfil como poseedor de una impresora 3D para posteriormente rellenar la información de esta y tener la posibilidad de crear anuncios o simplemente registrarse como un usuario básico buscando servicios.
- **Login**: El usuario será capaz de identificarse con sus datos de registro antes de acceder al uso de la aplicación.
- **Modificar información del perfil personal**: El usuario debe de poder modificar tanto su información personal como la imagen de perfil y su ubicación.
- **Visualizar perfil personal**: El usuario será capaz de ver toda su información personal localizada en un menú de fácil acceso de la aplicación.
- **Navegar en tablón de anuncios**: El tablón de anuncios deberá de ser la ventana principal de la aplicación, la cual aparece al acceder satisfactoriamente a través del login, en este, el usuario deberá poder navegar a través de múltiples anuncios del resto de usuarios que estén a una distancia máxima definida por el propio usuario. Los anuncios estarán organizados en una lista de banners donde se podrá consultar a golpe de vista la información básica de cada anuncio.
- **Visualizar anuncio**: Al seleccionar un anuncio, desde el tablón de anuncios, el usuario podrá ver en detalle toda la información de este, una imagen, título del anuncio, nombre del usuario, ubicación, descripción del anuncio, contacto y un acceso al perfil del usuario. En este se podrá ver más en detalle la información pública del usuario que ha creado el anuncio y la información de su impresora.

- **Crear anuncio**: El usuario podrá crear un anuncio propio desde el tablón principal, deberá de rellenar el formulario con la información necesaria y seguir los pasos indicados.
- **Modificar anuncio**: El usuario será capaz de modificar la información de su anuncio a posteriori de crearlo, simplemente accediendo a este.
- **Ver perfil del anunciante**: Como hemos adelantado antes, a través de los anuncios, el usuario será capaz de consultar la información pública del perfil de cada usuario, tanto los datos personales como la información de la impresora.
- **Conversar vía chat**: El usuario será capaz de ponerse en contacto con otros usuarios vía mensajería síncrona integrada en la propia aplicación.

Estas funcionalidades de la aplicación son las que deberemos de abordar en más detalle, agruparlas y darles forma en los siguientes apartados, definiendo tanto su lógica diseño y persistencia.

5. Diseño y modelado

Continuando con el apartado anterior vemos claras la organización de las funcionalidades en 4 grupos, la vista del login, perfil, tablón de anuncios y anuncio. A continuación, definiré estos grupos y los explicaré detalladamente.

Login

El login será un apartado imprescindible en nuestra aplicación. Necesitamos identificar a los usuarios que creen sus publicaciones tanto ofreciendo como buscando servicios. La ventana de login, por este motivo, será la primera a la cual un usuario accederá, tendrá la opción de registrarse a través de un formulario donde especificará su información básica, usuario, contraseña, correo electrónico y si es poseedor de una impresora 3D o simplemente introducir sus datos en la ventana de login para acceder a la aplicación.

El diseño de esta ventana será estándar y simple como muestran los primeros *mockups.*

Figura 16: Login mockup

Figura 17: Mockup de la ventana de registro

Después de rellenar sus datos personales y presionar siguiente (Figura 17), si el usuario especifica que es dueño de una impresora 3D, entonces, después del registro, pasará a una nueva plantilla donde rellenar los datos de su impresora, nombre, descripción y una imagen representativa (Figura 18).

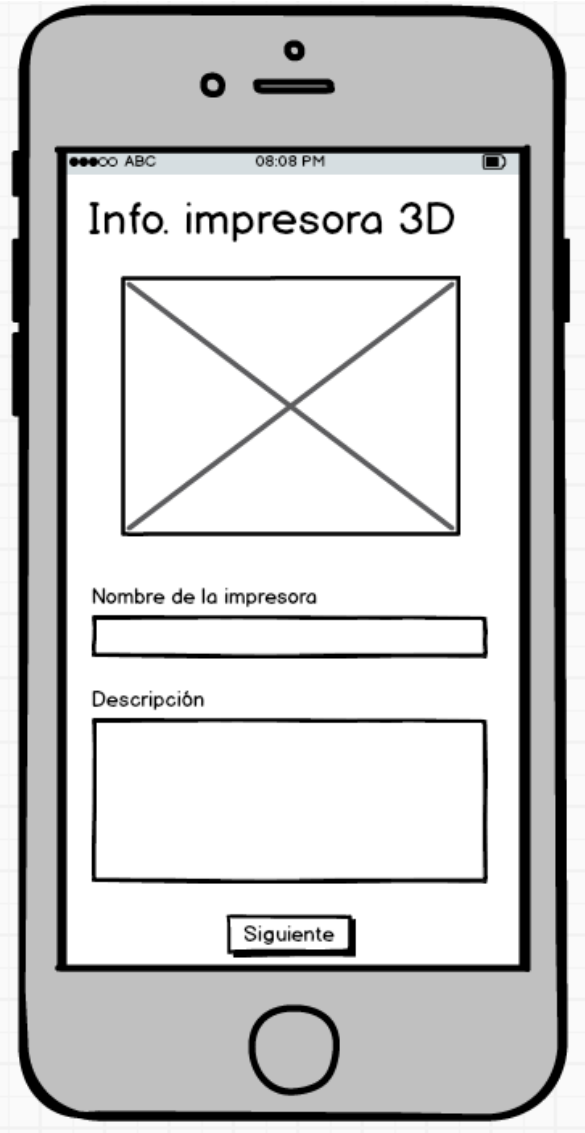

Figura 18: Mockup descripción impresora

Tanto si el usuario especifica que posee una impresora 3D como si no, también deberá definir una imagen de perfil (Figura 20) y especificar su ubicación en las ventanas posteriores (Figura 19).

Figura 20: Mockup seleccionar foto de perfil

Figura 19: Mockup definir ubicación

La jerarquía de estas ventanas sigue la siguiente estructura descrita en el diagrama de clases:

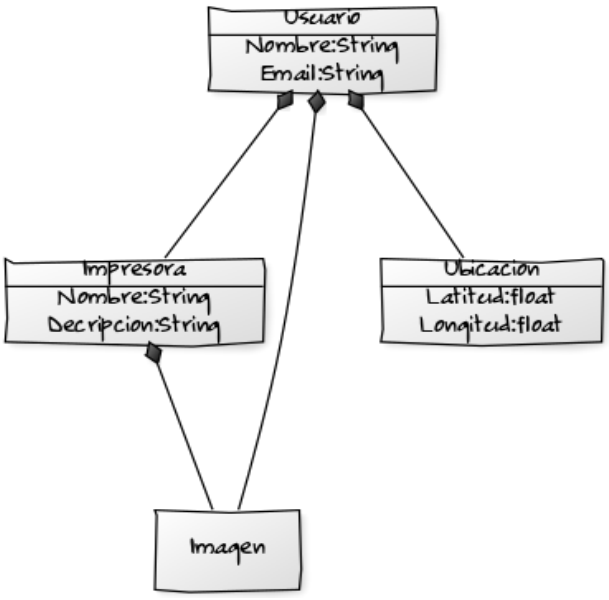

Figura 21: Diagrama de clases del objeto Usuario

Una vez definida esta información se dará por finalizado la creación del perfil. Mas adelante el usuario podrá modificar esta información accediendo a su perfil.

Perfil

En cuanto al perfil, el usuario será capaz de visualizar toda su información y modificarla, necesitaremos que en el perfil figure la ubicación aproximada del usuario, ya que una de las utilidades de la aplicación es que pueda priorizar anuncios dependiendo de la distancia a la que se encuentren de ti. También contaremos con información básica del usuario, donde figura el correo electrónico y nombre (Figura 22).

Figura 22: Mockup vista del perfil

Anuncios

Los anuncios son el apartado principal de nuestra aplicación. El desplegable del anuncio tendrá que mostrar toda la información necesaria para que el cliente se decida a ponerse en contacto con el vendedor para afianzar el trueque. Deberá de estar presente el nombre del anuncio, nombre del usuario que lo ha creado, la localidad de este, una breve descripción, y un contacto (Figura 23).

Figura 23: Mockup vista anuncio

La relación que existe entre los usuarios y los anuncios es simple, un usuario puede crear muchos anuncios y cada anuncio solo pertenece a un único usuario, como aparece en el siguiente diagrama.

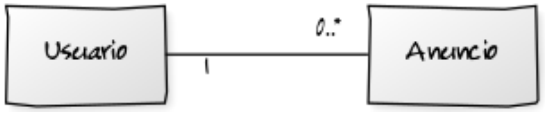

Figura 24: Diagrama de la relación Usuario-Anuncio

Tablón de anuncios

El tablón de anuncios será nuestro *main frame*, la vista principal que aparecerá después del login y desde la cual el usuario podrá navegar entre las distintas funcionalidades de la aplicación. En esta vista estará presentaran presentes unas pestañas de navegación en una barra en la parte superior de la vista, con las cuales podremos acceder tanto a los anuncios de los usuarios cercanos a nosotros como a lo que hemos creado nosotros. En el marco principal de la vista estarán ordenados los anuncios y con información básica en la vista previa (Figura 25). El usuario podrá acceder a toda la información del anuncio presionando encima de él.

Figura 25: Mockup tablón de anuncios

6. Tecnologías utilizadas

Tanto MongoDB como Firebase Cloud Messaging son dos tecnologías relativamente nuevas las cuales se ha decidido utilizar en este proyecto para darle un valor añadido y aprovechar las funcionalidades que nos ofrecen estas frente a tecnologías más clásicas.

MongoDB

MongoDB (Figura 26) es una base de datos *NoSQL*, es decir, no tiene tablas, registros ni *SQL*. Sin embargo, sigue siendo una buena opción para almacenar la información de nuestra aplicación.

Figura 26: Logo MongoDB

MongoDB es una base de datos orientada a documentos, es decir en lugar de guardar datos en registros, gurda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Una de las principales diferencias con una base de datos relacional, es que no es necesario seguir un esquema o un modelo. Los documentos de una misma colección pueden tener esquemas diferentes.

Figura 27: Ejemplo inserción de objeto

Podemos insertar un *documento* Usuario en nuestra colección de Usuarios de la siguiente manera (Figura 27), la novedad que MongoDB nos ofrece frente a las bases de datos

relacionales es que podremos insertar un nuevo Usuario tan solo con el nombre y un nuevo valor con diferente estructura, esto sería impensable en las bases de datos SQL (Figura 28).

Figura 28: Ejemplo insercion de objeto 2

MongoDB es muy útil en proyectos que presenten un alto grado de escalabilidad, la replicación y el *sharding*, opciones sencillas de configurar, nos ayudan a conseguir un sistema que escale horizontalmente sin demasiados problemas.

Algunas de las desventajas de MongoDB son que no existen transacciones, solo se garantizan operaciones atómicas a nivel de documento. Tampoco existen los JOINS, para consultar datos relacionados en dos o más colecciones, tenemos que hacer más de una consulta.

Firebase Cloud Messaging

Firebase Cloud Messaging (Figura 29) es una solución multiplataforma que te permite enviar, de forma gratuita y segura, mensajes y notificaciones.

Figura 29: Logo Firebase

El destino de dichas notificaciones son aplicaciones móviles instaladas tanto en Android como en iOS. Esto es posible gracias a la intervención del servicio de mensajería del servidor de Firebase.

Anteriormente este servicio de Google era conocido como Google Cloud Messaging o GCM, sin embargo, la plataforma actualizo las características de la arquitectura con esta nueva versión de Firebase Cloud Messaging. Algunas de las novedades que trajo este cambio fue el que ya no era necesario programar a mano el reintento de la suscripción o registro y una solución para las notificaciones donde ya se puede prescindir de un servidor externo con la consola web.

Las utilidades que ofrece las notificaciones *push* en Android son diversas y útiles:

- Comunicar la sincronización frente a un cambio
- Notificación del estado de un producto o servicio
- Envio de alertas informativas relacionadas a la aplicación.
- Implementar chats entre usuarios
- …

La arquitectura de Firebase Cloud Messaging es bastante sencilla, consta de tan solo tres componentes, la aplicación servidor, la aplicación cliente (Android) y el servidor de Firebase (Figura 30).

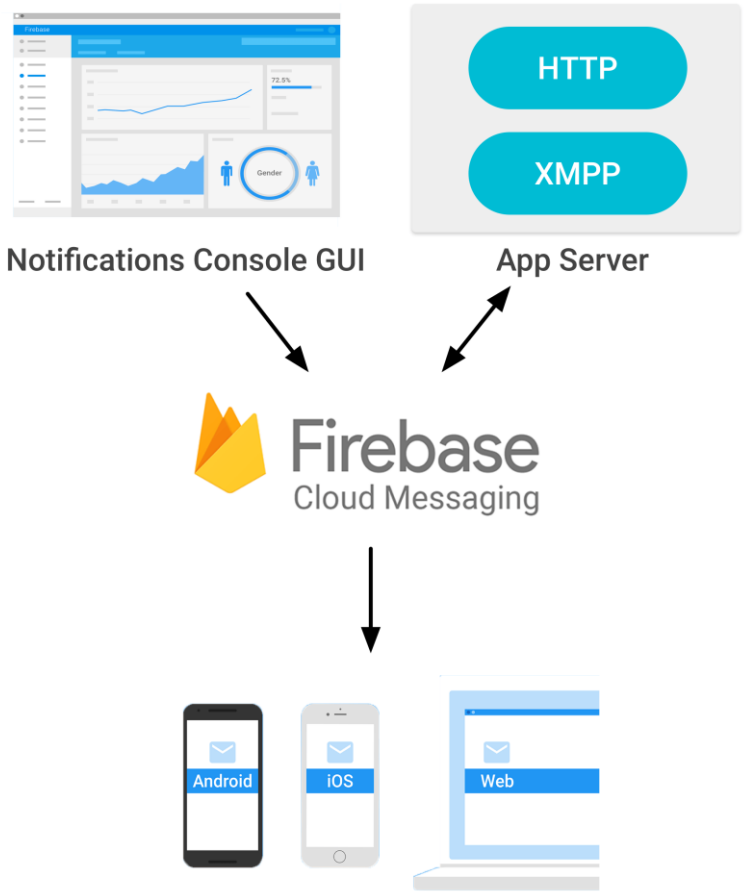

Figura 30: Arquitectura de Firebase Cloud Messaging

Como aparece en la figura, la aplicación servidor envía una petición al servidor Firebase, la plataforma notifica a todos sus dispositivos que estén registrados al proyecto.

Google Maps Android Api

Con la API de Google Maps Android (Figura 31) ya es posible agregar de forma sencilla y practica mapas de Google Maps a tu aplicación Android.

La API administra el acceso a los servidores, la descarga de datos, visualización de los mapas y la respuesta a gestos en los mapas de Google Maps.

Figura 31: Logo Google Maps API

La *API* de Google Maps se integró con los *Google Play Services* a finales del año 2012. Este cambio trajo consigo importantes mejoras. La utilización de mapas vectoriales y las mejoras en el sistema cache mejoraron considerablemente el rendimiento de las aplicaciones, esto repercutió directamente a una mayor velocidad de carga y menor consumo de datos.

Esto afectó también a forma en que los desarrolladores interactuaríamos con los mapas, dejando atrás las *activitis* como *MapActivity* y *MapView* para empezar a utilizar el *MapFragment*, con las correspondientes ventajas que conlleva el uso de los *fragments*.

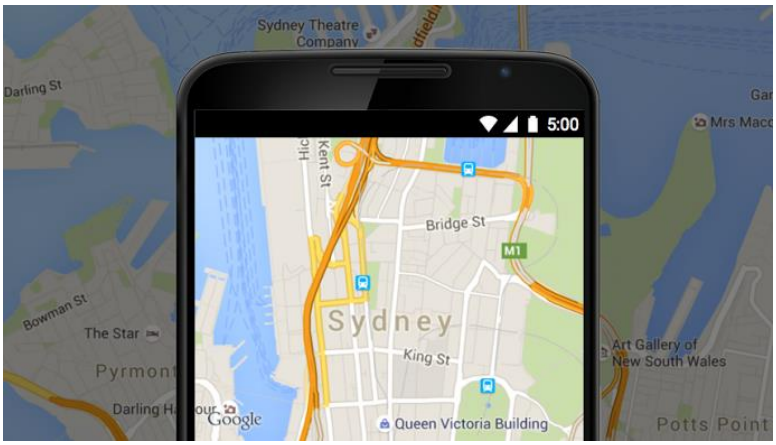

Figura 32: Vista de mapa básico

Trello

Trello (Figura 33) es un gestor de proyectos online. Permite aclarar rutinas de trabajo, priorizar, generar avisos de citas, hacer comentarios en tareas, indicar el porcentaje de progreso de una tarea, etc.

Figura 33: Logo Trello

Trello permite crear distintos paneles (llamados tableros), podemos crear tantos como nos sea necesario. En ellos, se pueden crear distintas columnas (o listas) que pueden ser los diferentes flujos de trabajo, separación de equipos o lo que nos convenga mejor a nuestro proyecto. En estas columnas podemos crear distintas tarjetas, donde podemos especificar las tareas que tenemos pendientes, o las tareas concretas propiamente dichas (Figura 34). En estas tarjetas es posible designar un responsable o encargado de esta, comentar sobre ella, crear *check lists*, crear fechas límite, etc.

Es cómodo de usar, puedes arrastrar cualquiera de las tareas a otra columna, catalogar las tareas por colores o comentar la terea asignada a un compañero. Es visual, con un golpe de vista puedes ver el estado global del proyecto.

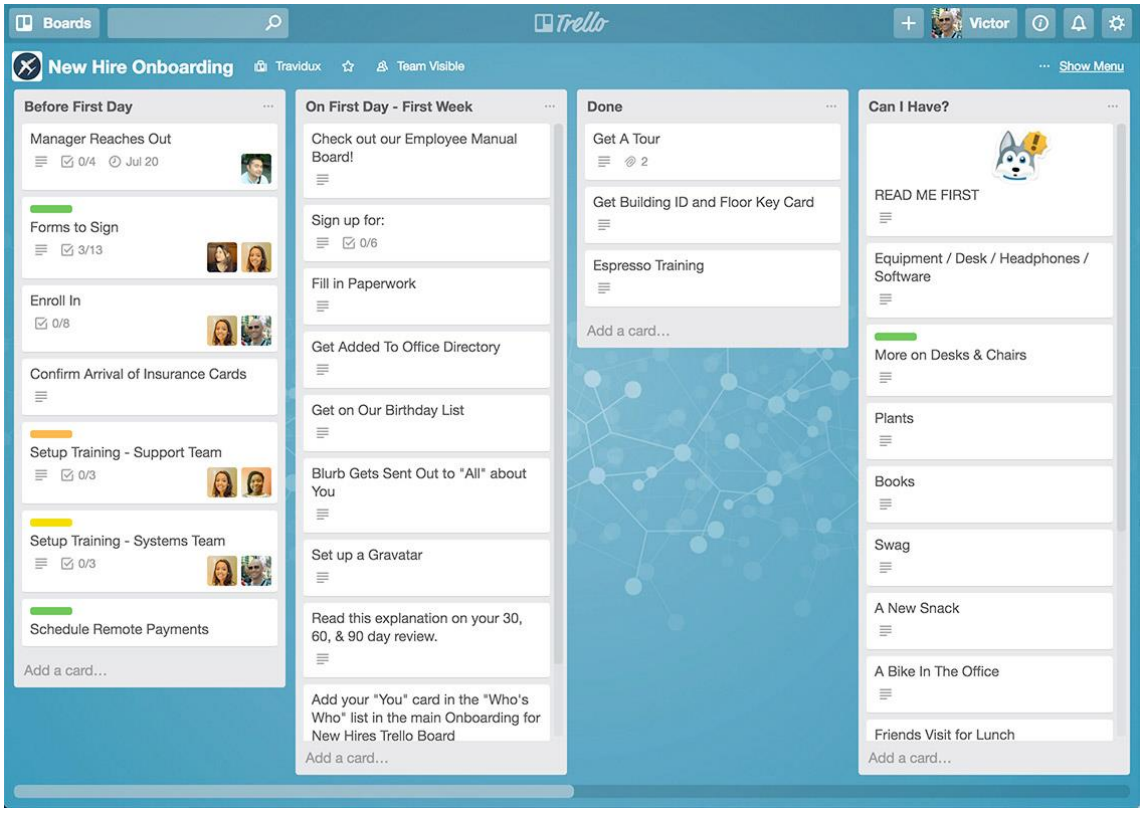

Figura 34: Vista principal de la herramienta Trello.

Si este tipo de plataformas está dirigido, entre otras, a gente que trabaja de forma remota, también cabe contemplar que muchos de ellos lo hagan en movilidad. Por ello, Trello es completamente multidispositivo.

7. Organización del trabajo

En este apartado panificaremos todo el transcurso del proyecto, destinaremos tiempo al aprendizaje de las nuevas tecnologías, a la elicitación de requisitos, desarrollo de casos de uso, diseño de la base de datos, diseño de la capa de presentación, elaboración de la lógica de la aplicación, etc.

Todo este trabajo será dividido en tres *sprints* de tres semanas. Al principio de cada *sprint* habrá una planificación de este dónde se seleccionarán las UTs (unidades de trabajo) que formarán el backlog de este sprint y las cuales deberán completarse en el transcurso de este. Al finalizar el sprint se hará una revisión del sprint y si alguna de las UTs no se ha podido realizar, esta pasará al backlog del *sprint* siguiente.

Sprint 1

Este primer sprint estará destinado a la investigación de las tecnologías que utilizaré en el proyecto, análisis de requisitos, especificación y diseño de la arquitectura. Se describirán detalladamente los cimientos de la aplicación y el camino por el cual se seguirá el desarrollo de la aplicación.

El backlog de este sprint estará compuesto por:

- **Investigación sobre MongoDB**. Leer información sobre la base de datos MongoDB y sopesar las ventajas y desventajas que podría conllevar aplicarla en el proyecto en comparación a bases de datos de tipo entidad-relación.
- **Elaborar un *Storyboard* y un mapa de características**. Hacer un primer esbozo del flujo de la aplicación, estimar aproximadamente el número de vistas necesario, primeros requisitos y un mapa de características.
- **Enumerar y especificar todos los requisitos.** Utilizar si es necesario metodologías de elicitación de requisitos, por ejemplo, los diagramas de contexto, prototipos, la observación o el análisis de documentos. Especificar con el máximo detalle posible estos para evitar el retrabajo o las ambigüedades. Ordenar por prioridad los requisitos una vez consensuados con métodos como el *MoSCoW* (Must-Should-Could-Won't).
- **Elaborar un diagrama de clases.**
- **Elaborar un diagrama de casos de uso.**
- **Investigar a la competencia**. Revisar funcionalidades de las aplicaciones de la competencia como son Wallapop, Milanuncios e Imprimalia. Elaborar una tabla con las características que comparten y aquellas que les dan un valor añadido y las destacan frente al resto. Sacar conclusiones al respeto y mejorar si es necesario la especificación de requisitos.
- **Elaborar un modelo de base de datos básico**. Después de elegir un sistema de base de datos, crear un modelo básico necesario para empezar a trabajar.
- **Diseño y modelado de las vistas**. Elaborar esquemas o *mockups* de las principales vistas de la aplicación como son el *login*, registro, tablón principal, anuncio, menú, descripción impresora y perfil. No es necesario entrar en mucho detalle a la hora de elaborarlos, pero procurar que cumplan toda la funcionalidad básica descrita.

Todas estas UTs están incluidas y descritas en la herramienta de trabajo Trello. En ella se llevará un seguimiento de estas y se anotaran los avances (Figura 35).

Figura 35: Backlog del sprint 1. Trello

Sprint 2

En este segundo sprint comenzaremos a programar nuestra aplicación, empezaremos por lo más básico, crearemos los constructores para nuestras clases, definiremos y crearemos un api para la comunicación con nuestra base de datos y la lógica de la aplicación.

Una vez tengamos esta base comenzaremos a diseñar las vistas y a implementarlas. Crearemos la vista para el login y la lógica que lo acompaña, la vista para la creación de un usuario a través del login (registro) y su correspondiente lógica, la vista para la creación de una impresora asociada al usuario y la lógica y por último tanto la vista del tablón principal de anuncios como su manejo.

Hecho este breve resumen, se presentan las UTs de este segundo *sprint*:

- **Definir y crear la clase constructor para los anuncios.** Clase con los atributos necesarios para su creación como son el usuario, descripción, zona, contacto, precio y descripción de la impresora. También deben de estar los *setters* y *getters* correspondientes.

- **Definir y crear la clase constructor para los usuarios.** Clase con los atributos de nombre y descripción, con sus correspondientes *setters* y *getters*.
- **Crear una clase *Api* para la comunicación con la base de datos.** Esta clase tendrá la configuración de la base de datos (APIKEY y URL) acompañado de los métodos REST necesarios para poder interactuar con esta.
- **Definir y crear la clase constructor para las impresoras.** Clase con los atributos necesarios para crear un usuario: usuario, contraseña, email e impresora. También estarán implementados los *setters* y *getters* correspondientes.
- **Crear la vista para el login.** Una ventana simple con el logo de Shar3D, fields para introducir el usuario y contraseña y dos botones para acceder o registrarse.
- **Crear la vista para la creación del usuario.** Vista con los campos necesarios para la creación de un usuario, un *checkbox* para especificar si es dueño de una impresora y un botón para aplicar el registro.
- **Implementar la lógica para el login.** Se deberá pedir un usuario y contraseña para identificar a los usuarios. Tras comprobar que el usuario existe en la base de datos y la contraseña coincide con la introducida se le redirigirá a la vista principal.
- **Implementar de creación de impresora.** Vista con los campos necesarios para la creación de una impresora.
- **Implementar la lógica de creación de una impresora.** Esta vista será complementaria a la creación del usuario en el registro. Si el usuario marca la opción de poseer una impresora entonces pasará a la vista de la creación de esta, una vez rellenado el formulario podrá finalizar el registro.
- **Creación de la vista del tablón de anuncios.** Esta vista será una lista de anuncios, implementar complementariamente un *adapter* para las tarjetas de los anuncios con un diseño personalizado donde estará especificado la impresora del usuario, el nombre del usuario y la zona especificada en el anuncio. Este tablón también deberá de tener un menú superior de pestañas donde se podrá navegar entre los anuncios de los usuarios y los propios.

Estas UTs del segundo sprint han sido añadidas en la herramienta Trello, como se puede ver en la Figura 36, para mantener una organización y seguimiento de estas.

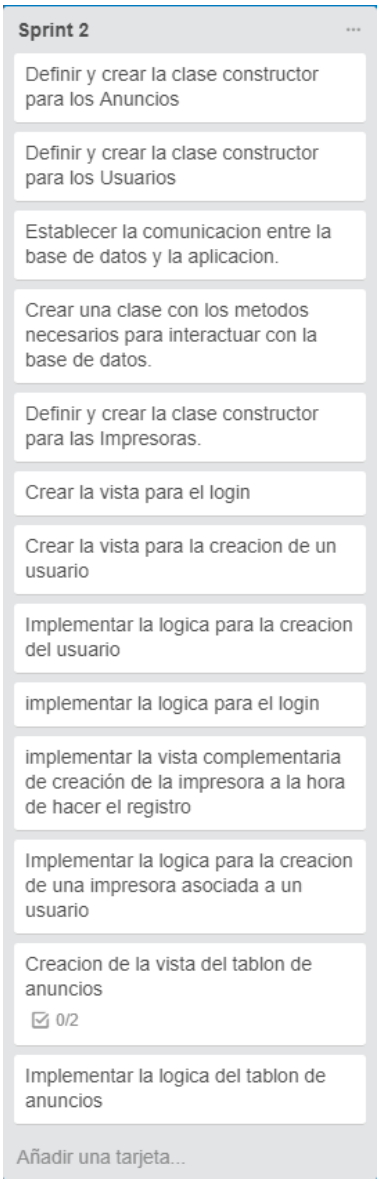

Figura 36: Backlog del sprint 2. Trello

Estas por tanto serán las UTs para este sprint, donde después de finalizar esta segunda iteración tendremos una primera versión de nuestra aplicación. Conforme a los principios del desarrollo ágil seriamos ya capaces de mostrarle al cliente un avance con valor para este.

Sprint 3

En este tercer *sprint* deberemos de seguir implementando la funcionalidad básica de la aplicación como es la creación de anuncios y la visualización, modificación y eliminación de estos. Un menú lateral para navegar entre el perfil y el tablón de anuncios y finalmente la visualización del perfil personal y modificación de este.

Acabado con la funcionalidad básica, pasaremos a implementar funcionalidades que le añadirán valor a la aplicación, como son, la selección de una imagen, tanto al crear el perfil como al crear

la impresora, la selección de la ubicación en la creación del perfil y la creación de un chat para interacción de los usuarios en la aplicación.

El backlog de este tercer sprint será, por tanto:

- **Crear la vista de creación de un anuncio.** La vista será un formulario con los campos necesarios para la creación de un anuncio, descripción, localización, contacto y precio.
- **Crear acceso a la creación de un anuncio.** El acceso al formulario de creación de un anuncio será un *FloatingActionButton* en el tablón de anuncios.
- **Crear la vista del perfil.** En esta vista estará la información básica del usuario, su nombre y su dirección de correo electrónico. También habrá un botón de cerrar sesión.
- **Implementar la lógica de creación de un anuncio.** Al pulsar en el botón de crear anuncio, se redirigirá al formulario de creación, después de rellenarlo con los datos necesarios y aplicar los cambios el anuncio aparecerá en la pestaña de "Mis anuncios". Si otro usuario entra en la aplicación verá el anuncio en la pestaña de "General". Tener en cuenta que la opción de crear anuncio solo estará disponible para los usuarios dueños de una impresora.
- **Crear la vista del anuncio ampliado.** En esta vista estará toda la información del anuncio, el nombre de la impresora, nombre del usuario, zona, descripción y email. También habrá un botón para acceder al perfil del usuario.
- **Crear menú lateral de navegación.** Este menú deberá de implementar con un *NavigationView*. Se podrá acceder tanto al tablón de anuncios como al perfil de usuario. Deberá añadirse al cargar la vista principal.
- **Implementar la lógica de acceso, visualización y modificación del perfil.** El perfil debe de ser accedido desde el menú de navegación por el propio usuario y desde el anuncio creado por este, por los demás usuarios. Una vez en el perfil, si el que lo visualiza es el dueño, tendrá la opción de modificarlo.
- **Implementar la lógica de subir imágenes desde el dispositivo.** Investigar la librería que mejor se adapte al proyecto y aplicarla para conseguir una funcionalidad sencilla de subir imágenes para el usuario. Esta opción de subir imágenes tendrá que integrarse en el formulario de creación del perfil y de creación de la impresora.
- **Implementar la funcionalidad de definir la ubicación.** Investigar sobre las librerías que mejor se adapten al proyecto y aplicarla para conseguir la funcionalidad de definir una ubicación asociada a cada usuario
- **Añadir la lógica de ordenar los anuncios por distancia.** En los anuncios deberá aparecer la distancia a la ubicación que definió el usuario que creo el anuncio. Los anuncios estarán ordenados de más cerca a más lejos por esta distancia.
- **Crear la vista para añadir la ubicación.** Esta vista deberá estar presente tanto en el registro del usuario como posteriormente poder modificarla en el perfil.
- **Añadir la ubicación en el mapa en la vista del perfil.** Crear un cuadro donde irá embebido el mapa con la localización exacta definida por el usuario.
- **Crear un chat interno.** Investigar sobre que librería se adaptaría mejor al proyecto e implementar un chat en la aplicación. Las conversaciones comenzarán al presionar el botón de "Abrir chat" en un anuncio, se creará una sala donde el propio usuario y el dueño del anuncio podrán intercambiar mensajes. Deberán de implementarse notificaciones al recibir un nuevo mensaje, donde aparecerá el usuario remitente y el anuncio desde el que se realiza. Debe de implementarse una vista donde aparecerá una lista con los usuarios que te han enviado al menos un mensaje en algún momento.

En este tercer sprint es donde recae una mayor cantidad de trabajo y funcionalidades a implementar es posible que alguna de estas se quede fuera de plazo de entrega y no sea posible implementarlo a tiempo antes de finalizar el sprint, sin embargo, quedara constancia en el backlog y se podría discutir la necesidad de estas funcionalidades y negociar incluso un cuarto sprint de un periodo más corto donde poder terminarlas.

Todas estas unidades de trabajo están introducidas en el gestor de tareas, como aparece en la Figura 37.

Figura 37: Backlog del sprint 3

8. Implementación

En este punto voy a explicar tan solo los apartados que considero más interesantes o en aquello en los que he tenido que invertir más tiempo en investigar cómo implementarlos. Comentare los cambios de decisiones que tome respecto a la especificación inicial y como ha repercutido esto en el resultado final.

La estructura que emplearé para para hablar de esto será una división de las capas de la aplicación, presentación, lógica de negocio y persistencia. Donde destacare lo más importante en cada una de ellas.

Presentación

En esta capa he intentado seguir de forma estricta la especificación inicial. La mayor parte de las vistas implementadas aparecen en el diagrama de flujo de la Figura 40.

Una de las decisiones estéticas de la aplicación que tome, que no estaba descrita en la especificación del aplicativo, fue implementar un redondeado a las imágenes. Este redondeo se aplicaría tanto en las imágenes del perfil de usuario como en la impresora.

Para implantar esto decidí usar la librería ***Glide***. Es fácil de usar y bastante popular en el desarrollo de aplicaciones, sobre todo, por la optimización en la fluidez al hacer *scroll* sobre listas de imágenes. Consume menos memoria en comparación con otras librerías como Picasso. Además, ofrece la posibilidad de trabajar con gifs y la transformación de imágenes.

Glide

El código para la implementación es el siguiente:

```
1. public class CircleTransform extends BitmapTransformation {
2.     public CircleTransform(Context context) {
3.         super(context);
4.     }
5.
6.     @Override
7.     protected Bitmap transform(BitmapPool pool, Bitmap
   toTransform, int outWidth, int outHeight) {
8.         return circleCrop(pool, toTransform);
9.     }
10.
11.          private static Bitmap circleCrop(BitmapPool pool,
   Bitmap source) {
12.              if (source == null) return null;
13.
14.              int size = Math.min(source.getWidth(),
   source.getHeight());
15.              int x = (source.getWidth() - size) / 2;
16.              int y = (source.getHeight() - size) / 2;
17.
18.              Bitmap squared = Bitmap.createBitmap(source, x, y,
   size, size);
19.
```

```
20.              Bitmap result = pool.get(size, size,
   Bitmap.Config.ARGB_8888);
21.              if (result == null) {
22.                  result = Bitmap.createBitmap(size, size,
   Bitmap.Config.ARGB_8888);
23.              }
24.
25.              Canvas canvas = new Canvas(result);
26.              Paint paint = new Paint();
27.              paint.setShader(new BitmapShader(squared,
   BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP));
28.              paint.setAntiAlias(true);
29.              float r = size / 2f;
30.              canvas.drawCircle(r, r, r, paint);
31.              return result;
32.          }
```

También será necesario añadir las dependencias al proyecto.

```
1. dependencies {
2.   compile 'com.github.bumptech.glide:glide:3.7.0'
3.   compile 'com.android.support:support-v4:19.1.0'
4. }
```

Después de esto solo será necesario llamar *Glide* para aplicar la transformación. Deberemos de añadir el contexto, la imagen a transformar y el *ImageView* donde lo queremos colocar.

```
1.  Glide
2.     .with(this)    // pass Context
3.     .load(url)     // pass the image url
4.     .into(myImageView); // the ImageView to which the image is
   to be loaded
```

También decidí que sería necesario definir una tarjeta para los anuncios personalizada que se ajuste a mis necesidades en vez de utilizar la fila por defecto de la vista *list*. Esto le daría más personalidad a la aplicación y podría estructurar de forma visual toda la información que necesitaría mostrar de forma genérica y posteriormente tan solo introducir estos valores en cada caso.

CardView

Para ello definí una vista para la fila, “row_list_cardview”, que tendría un aspecto como aparece en la Figura 38 y estaría compuesto por un *ImageView* de fondo, el nombre del anuncio, el Nick del usuario y la distancia o localización.

El código para implementarlo es sencillo, tan solo defino una clase con los valores del objeto, un método para inicializarlos y un método para rellenar la información de cada tarjeta con la información del anuncio.

```
1.     public class AnuncioViewHolder
2.             extends RecyclerView.ViewHolder {
3.         //atributos de cada uno de los items en la ui
```

```
4.          private TextView nombre;
5.          private TextView lugar;
6.          private ImageView imagenImp;
7.          private View itemView;
8.
9.          public AnuncioViewHolder(View _itemView) {
10.                 super(_itemView);
11.                 //inicializo punteros
12.                 itemView = _itemView;
13.
    nombre = (TextView) itemView.findViewById(R.id.nombreView);
14.
    lugar = (TextView) itemView.findViewById(R.id.lugView);
15.
    imagenImp = (ImageView) itemView.findViewById(R.id.imagenImpreso
    ra);
16.             }
17.
18.             //relleno item
19.             public void bindAnuncio(Anuncio e) {
20.                 SharedPreferencesHelper
    session = new SharedPreferencesHelper(ctx);
21.
      if (e.getImgPrinter() != null && !Utils.isNullOrEmpty(e.getImg
    Printer()))
22.
    Glide.with(ctx).load(Base64.decode(e.getImgPrinter(),
    Base64.DEFAULT))
23.                                 .crossFade()
24.                                 .thumbnail(0.5f)
25.
    .bitmapTransform(new CircleTransform(ctx))
26.
    .diskCacheStrategy(DiskCacheStrategy.ALL)
27.                                 .into(imagenImp);
28.
29.
30.
    nombre.setText(e.getPrintername() + "\n" + e.getUsername());
31.
      if (e.getLatitude() != 0.0 && e.getLongitude() != 0.0 && sessi
    on.getLatitude() != 0.0 && session.getLongitude() != 0.0) {
32.
    lugar.setText(Utils.calculateDistance(e.getLatitude(),
    e.getLongitude(), session.getLatitude(),
    session.getLongitude()));
33.                 } else
34.                     lugar.setText(e.getZona());
35.             }
```

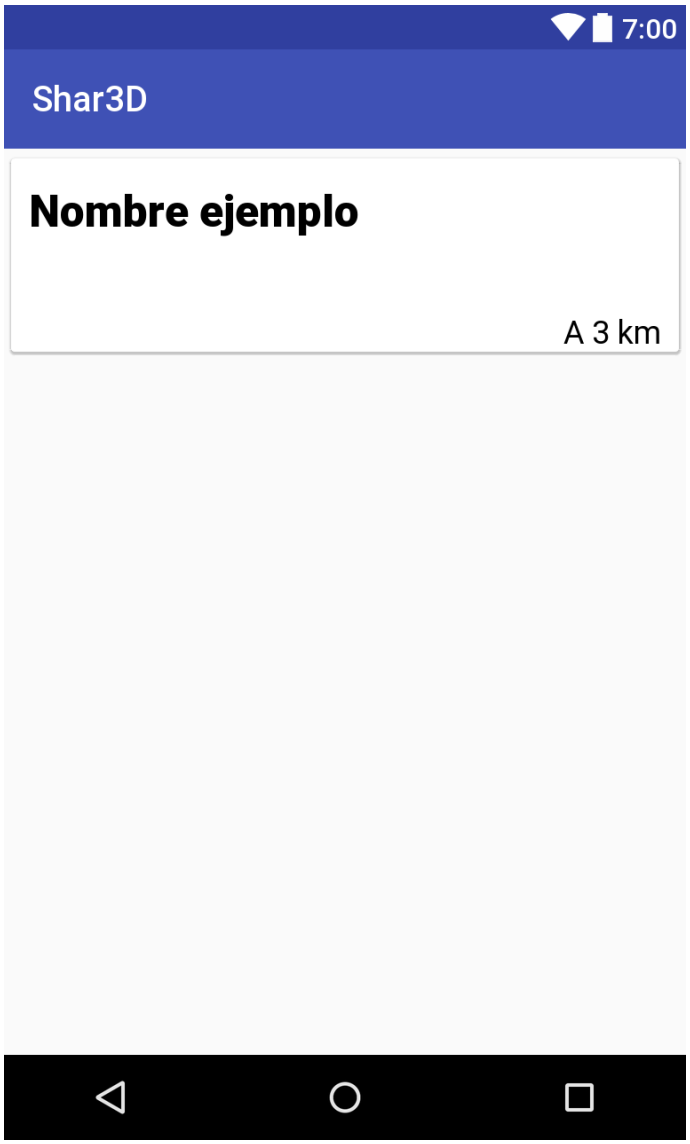

Figura 38: row_list_cardview

Por último, en cuento a las decisiones de diseño de la aplicación quizás destacar la implementación de un menú lateral de navegación con el componente NavigationView en vez de simplemente utilizar el menú de los tres puntos desplegando las opciones con el componente básico *Menu* de Android.

Esta implementación pretende estandarizar un poco más el diseño de la aplicación siguiendo las guías de diseño de *Material Design*. También ofrece al usuario una experiencia de navegación más cómoda e intuitiva. A parte, nos permite escalar las funcionalidades de forma más sencilla.

He seguido un diseño estándar para esta vista. En la parte superior del marco aparecerá la imagen del usuario y su información básica, el nombre y correo electrónico, en el marco inferior aparecerán todas las opciones de navegación de la aplicación, como aparece en la Figura 39.

En nuestro caso tanto pulsando en la imagen de perfil del marco superior como en la opción "Perfil" del menú, el usuario podrá acceder a la vista de su perfil. En el caso en el que el usuario acceda a la aplicación sin estar identificado la opción del perfil servirá para redirigirle a la ventana del *login.*

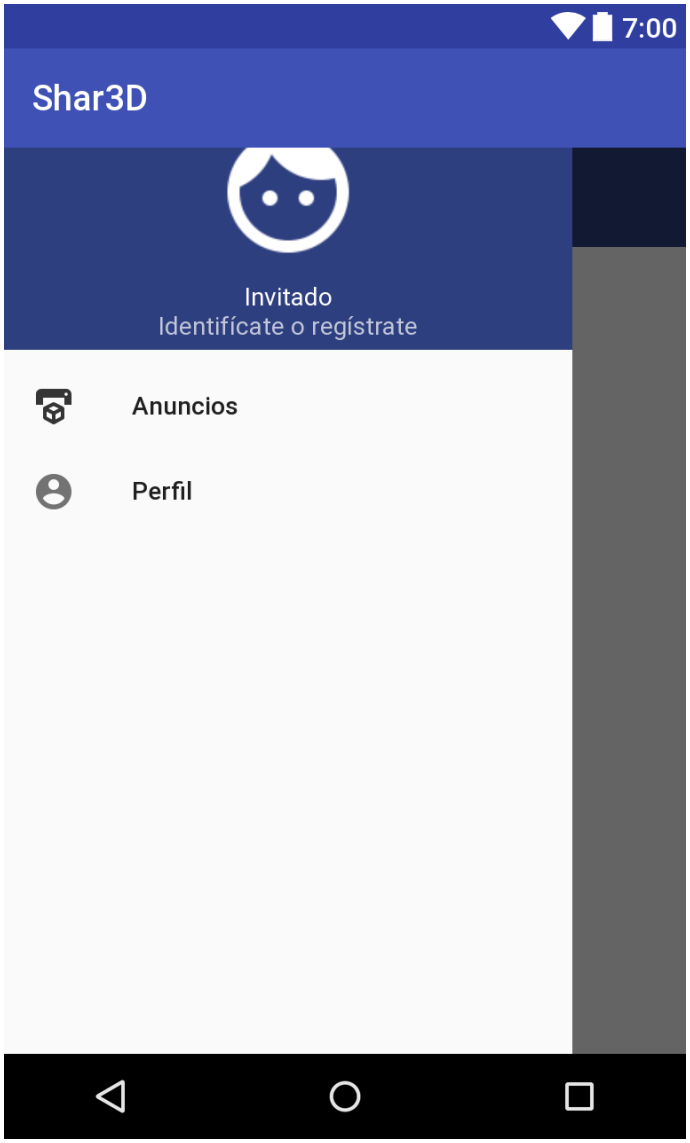

Figura 39: NavigationView

Voy a obviar el código porque no aporta ningún valor añadido, la creación del menú es genérica, en la actividad principal se instancia la vista y se rellena con los datos del usuario.

Figura 40: Diagrama de flujo

Persistencia

En cuanto a la persistencia me decidí a usar *MongoDB*. Puesto que nunca había trabajado con bases de datos *NoSQL*, pese que esta era una buena oportunidad para aprender a usarlas y darle un valor añadido al proyecto.

mLab

Para el despliegue de la base de datos he utilizado *mLab*. Es un servicio *cloud,* que ofrece 500Mb para tus proyectos y utilidades para realizar de forma más cómoda y simple la comunicación entre la base de datos y tu aplicación.

Como aparece en la Figura 41, desde el panel de administración de la web podremos configurar todas las bases de datos, crear nuevas, añadir colecciones, índices, definir restricciones, etc.

He encontrado bastante útil la opción que ofrece esta página web de almacenar tus backups de las bases de datos, incluso con la sesión gratuita.

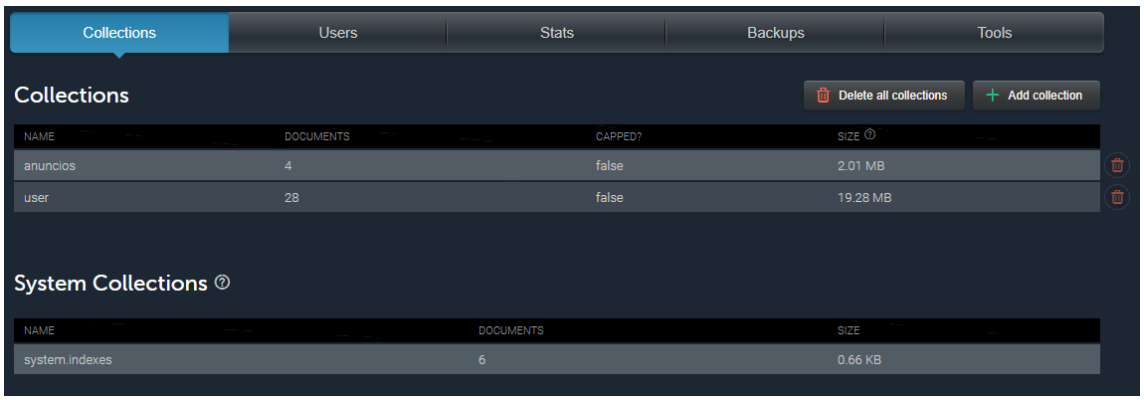

Figura 41: Panel de control. mLab

Para las peticiones Http he utilizado la librería *Volley*. Esta destaca por su sencillez, ya que no es necesario crear tantas estructuras intermediarias para establecer una comunicación como con otras librerías, por ejemplo, *retrofit*.

El modelo de datos tratándose de una base de datos NoSQL, es sencillo, en nuestro caso solo hay dos colecciones, anuncios y usuarios.

Para interactuar con la base de datos tan solo tendremos que hacer una llamada de petición de servicio (*RequestService*). Para explicar esto utilizare el ejemplo de creación de un usuario.

Una vez el usuario introduce todos los datos en el formulario de registro y pulsa el botón de “Crear”, después de comprobar que los datos introducidos son correctos…

```
1.     if (!Utils.checkEmaill(email)) {
2.                     Toast.makeText(parent, "Introduce una
   direccion de correo valida", Toast.LENGTH_SHORT).show();
```

```
3.
     } else if (password.length() == 0 || password.isEmpty()) {
4.                     Toast.makeText(parent, "Introduce una
   contraseña valida", Toast.LENGTH_SHORT).show();
5.
     } else if (username.isEmpty() || username.length() == 0) {
6.                     Toast.makeText(parent, "Introduce un usuario
   valido", Toast.LENGTH_SHORT).show();
7.                 } else {
```

Notificaremos al usuario que vamos a proceder a crear el usuario y lo ponemos en espera.

```
1. pd = new ProgressDialog(parent);
2. pd.setCancelable(false);
3. pd.setCanceledOnTouchOutside(false);
4. pd.setMessage("Espera por favor");
5. pd.show();
```

Pasaremos a crear el objeto “Usuario” y lo convertiremos en Json.

```
1. User user = new User();
2. user.setUsername(username);
3. user.setPassword(password);
4. user.setEmail(email);
5.
6. GsonBuilder gb = new GsonBuilder();
7. gb.serializeNulls();
8. Gson gson = gb.create();
9. final String jsonUser = gson.toJson(user);
```

Una vez tenemos el *Json* con el usuario listos haremos la llamada con *JsonObjectRequest* para guardar el usuario en la base de datos.

```
1. try {
2.      JsonObjectRequest
   req = new JsonObjectRequest(Request.Method.POST,
   Utils.getSignUpURL(), new JSONObject(jsonUser), new Response.Lis
   tener<JSONObject>() {
3.         @Override
4.         public void onResponse(JSONObject response) {
5.            Log.d(TAG, response.toString());
6.            Bundle args = new Bundle();
7.            args.putString("user", jsonUser);
8.            if(pd.isShowing())
9.               pd.dismiss();
10.
11.                  //IR A LA SIGUIENTE PANTALLA DEPENDIENDO DEL
   CHECKBOX.
12.                  Fragment nextFrag;
13.                  if (isPrinter.isChecked())
14.                     nextFrag = new FragmentPrinterInfo();
15.                  else
16.                     nextFrag = new FragmentProfilePicture();
17.
18.                  nextFrag.setArguments(args);
19.
    getActivity().getSupportFragmentManager().beginTransaction()
```

```
20.                     .replace(R.id.content_frame, nextFrag)
21.                     .commit();
22.             }
23.         }, new Response.ErrorListener() {
24.             @Override
25.             public void onErrorResponse(VolleyError error) {
26.                 NetworkResponse
   response = error.networkResponse;
27.                 if (response != null) {
28.                     String json = new String(response.data);
29.                     String msg = "";
30.                     try {
31.
   msg = new JSONObject(json).getString("message");
32.                     } catch (JSONException e) {
33.                         e.printStackTrace();
34.                     }
35.
36.                     Log.d(TAG, msg);
37.                     if (msg != null) {
38.                         msg = msg.substring(msg.indexOf("dup
   key"));
39.
40.                         if (msg.contains(user.getUsername())) {
41.                             Snackbar.make(topView, "El nombre
   usuario ya existe.", Snackbar.LENGTH_LONG).show();
42.
    } else if (msg.contains(user.getEmail())) {
43.                                 Snackbar.make(topView, "El
   email ya existe.", Snackbar.LENGTH_LONG).show();
44.                             }
45.                         } else {
46.                             Snackbar.make(topView, "Ha habido un
   error con la red. Inténtalo más tarde.",
   Snackbar.LENGTH_LONG).show();
47.                         }
48.
49.                         if(pd.isShowing())
50.                             pd.dismiss();
51.
52.             }
53.             });
54.
   RequestService.getInstance(getContext()).addToQueue(req);
55.         }catch (Exception a) {
56.             a.printStackTrace();
57.         }
```

A la petición *JsonObjectRequest* le pasamos como paramento el método *REST*, la URL y el recurso *Json*. Esta llamada puede fallar por varios motivos, debido a esto la hacemos en un *try-catch*. Uno de estos motivos puede ser un fallo en la conexión de internet al realizar la llamada, este fallo lo controlamos y mostraremos un mensaje informando del error como aparece en la línea 46. También puede ocurrir que el nombre de usuario o el correo electrónico que estamos creando ya este en uso, informaremos al usuario en ese caso con el mensaje que aparece en las líneas 41 y 43. Si ninguno de estos casos se presenta podremos guardar de forma satisfactoria el usuario en nuestra base de datos.

Las URLs que necesitamos para interactuar con la base de datos las definiremos en una clase auxiliar llamada "Utils". Un ejemplo de la url que hemos utilizado antes esta descrita a continuación.

```
1.      public static String getSignUpURL() {
2.
      return "https://api.mlab.com/api/1/databases/sdm/collections/u
    ser?apiKey=" + APIKEY;
3.      }
```

El paramento "APIKEY" es la clave que obtenemos al crear nuestra base de datos en *mLab* y permite identificar nuestra base de datos. Esta variable la definiremos como estática en la clase "Utils".

```
public static final String APIKEY = "78uYSF3hn2cshyPy8tECye7XcN*****";
```

Con esto ya tendríamos listo la comunicación con la base de datos. La forma en la que creamos los anuncios y los consultamos se puede extrapolar fácilmente del método de crear un usuario explicado anteriormente.

Lógica de negocio

En cuanto a la lógica en la aplicación es bastante simple, se controlan el flujo entre las distintas ventanas de la aplicación, la creación tanto de usuarios como de anuncios a través de formularios, peticiones a la base de datos etc. Sin embargo, em gustaría destacar la implementación de los mapas a través de la API de Google Maps y la aplicación para el uso de imágenes.

Google Maps Android API

En la aplicación vamos a hacer uso de los mapas de Google Maps para facilitar al usuario especificar su ubicación, poder ver la ubicación de los usuarios en sus perfiles y organizar los anuncios por proximidad en el tablón de anuncios.

Para empezar a trabajar con la *API* será necesario añadir el complemento de Google Play Services desde el panel de *Android SDK*, posteriormente será necesario añadir tanto las dependencias al archivo "*build.gradle*" como la APIKEY y el nombre en forma de metadatos al archivo "*AndroidManifest*" como aparece a continuación.

```
1. <meta-data
2.     android:name="com.google.android.geo.API_KEY"
3.     android:value="AIzaSyCSqC201wYrI89N-yYX9ISh89DT4ws****"/>
```

En nuestro caso la vista en la que vamos a implementar la funcionalidad de los mapas será tanto al definir la ubicación en la creación del usuario como en la modificación de esta en el perfil. En ambos casos utilizaremos la vista que aparece en la Figura 42.

Figura 42: Vista para definir la ubicación.

Desde esta ventana tenemos varias funcionalidades. Al presionar en "Seleccionar ubicación" se abrirá un mapa donde podremos navegar y definir la localización que queramos. Al seleccionar la distancia a la que estamos dispuesto desplazarnos mediante la *seekBar*, se utilizara esta distancia para filtrar los anuncios que aparezcan en nuestro tablón de anuncios. Al seleccionar "No mostrar radio" la distancia se establecerá como 0 y no se tendrá en cuenta para filtrar los anuncios.

Al presionar en "Seleccionar ubicación" crearemos un *intent* a partir de un *PlacePiker* como aparece en la línea 12. Este bloque deberemos de ejecutarlo en un try-catch para manejar los posibles errores que puedan ocurrir, como por ejemplo, que el usuario no haya dado permisos a la aplicación a acceder a su ubicación.

```
1. mapButton.setOnClickListener(new View.OnClickListener() {
2.     @Override
3.     public void onClick(View v) {
4.         try {
```

```
5.           if (ContextCompat.checkSelfPermission(parent,
6.
     Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PE
   RMISSION_GRANTED) {
7.
8.
   ActivityCompat.requestPermissions(parent,new String[]{Manifest.p
   ermission.ACCESS_FINE_LOCATION}, 3);
9.           } else {
10.                          PlacePicker.IntentBuilder builder = new
11.                          PlacePicker.IntentBuilder();
12.                          Intent intent = builder.build(parent);
13.
    startActivityForResult(intent,PLACEPICKER_INTENT);
14.                          Log.d("START PLACEPICKER", "Abriendo
   mapas...");
15.
16.                  }
17.                } catch (GooglePlayServicesRepairableException
   e) {
18.
   GooglePlayServicesUtil.getErrorDialog(e.getConnectionStatusCode(
   ), getActivity(), 0);
19.                } catch (GooglePlayServicesNotAvailableException
   e) {
20.                           Toast.makeText(getActivity(), "Google
   Play Services is not available.",Toast.LENGTH_LONG).show();
21.                }
22.        }
23.      });
```

Al seleccionar nuestra ubicación en el mapa y presionar en guardar (Figura 43), recogeremos las coordenadas de latitud y longitud para actualizar la ubicación del usuario mediante una llamada *PUT* a la base de datos.

```
1. JsonObjectRequest
   req = new JsonObjectRequest(Request.Method.PUT,
   Utils.getUpdateProfileURL(username), new JSONObject(Utils.getUpd
   ateLocationTemplate(latitude, longitude, radius))
```

Figura 43: Confirmación de la ubicación.

Evidentemente también tendremos que manejar los posibles errores que puedan ocurrir al hacer esta llamada, como un error en la red, aunque no mostremos el código aquí para evitar repetir funcionalidades explicadas anteriormente.

He querido destacar el apartado del manejo de las imágenes en la aplicación porque he tenido que dedicarle algo más tiempo a investigar cómo podía reducir el peso de las imágenes al darme cuenta que los tiempos de carga en las vistas donde había varias imágenes, como es el caso del tablón de anuncios, tardaba demasiado en cargar y aparecer la información.

Después de barajar varias librerías, entre ellas *Picasso*, decidí utilizar *Glide*. Dado que los motivos de elegir *Glide* y los pasos a seguir para poder comenzar a usarlo en el proyecto ya los he explicado en el apartado de Presentación, voy a pasar a explicar directamente el uso de la librería *Glide* en el manejo de las imágenes tanto de perfil como de las impresoras.

Glide

En nuestra aplicación gestionamos imágenes desde varios sitios, en concreto, podemos cambiar la nuestra imagen de perfil tanto al modificar el perfil como al crearlo y también podemos cambiar la imagen de la impresora de la misma forma, tanto desde el perfil del usuario en el apartado de información de la impresora como al crear la impresora al hacer el registro del usuario. Todas estas vistas aparecen en la Figura 44.

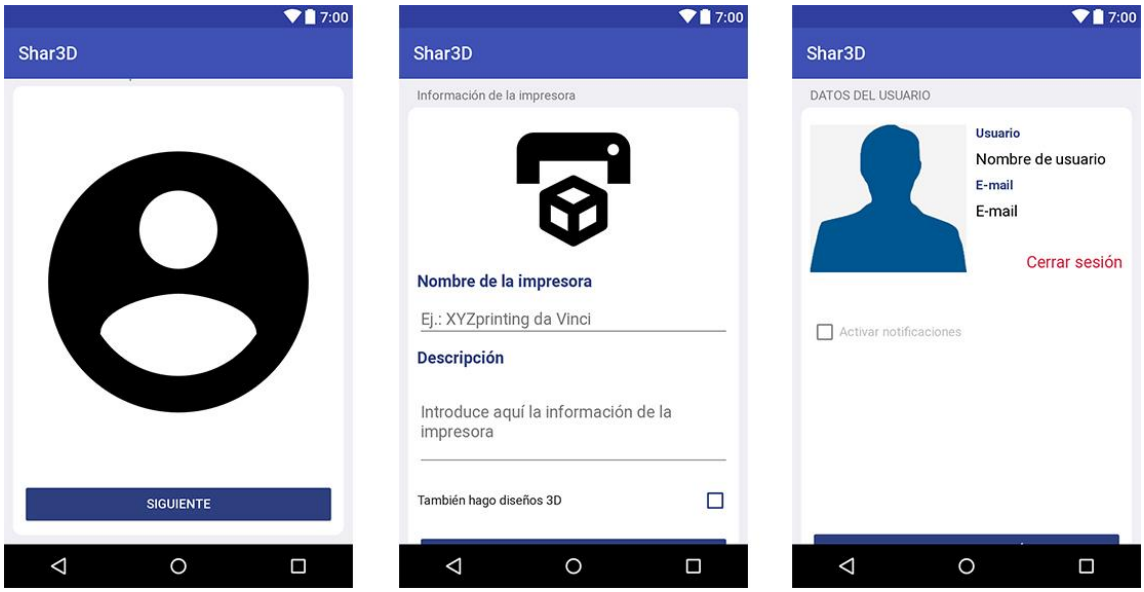

Figura 44: Vistas que interactúan con imágenes.

Todos estos *imageView* tienen la misma llamada al ser pulsados, en sus correspondientes Fragments.

```
1. image.setOnClickListener(new View.OnClickListener() {
2.     @Override
3.     public void onClick(View v) {
4.           imageUri = Utils.createFileUri(parent);
5.           Intent chooser = Utils.getMediaChooser(imageUri);
6.           startActivityForResult(chooser, 1);
7.      }
8. });
```

El método *getMediaChooser* maneja la elección del usuario sobre la importación de la imagen desde su galería o hacerla desde la cámara del dispositivo.

```
1.     public static Intent getMediaChooser(Uri uri) {
2.         Intent intent = new Intent();
3.         intent.setType("image/*");
4.         intent.setAction(Intent.ACTION_GET_CONTENT);
5.
6.         Intent
   camera = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
7.         camera.putExtra(MediaStore.EXTRA_OUTPUT, uri);
8.
```

```
9.          List<Intent> extraIntents = new ArrayList<Intent>();
10.               extraIntents.add(camera);
11.               Intent
   chooser = Intent.createChooser(intent, "Selecciona una
   aplicación");
12.
13.               chooser.putExtra(Intent.EXTRA_INITIAL_INTENTS,
   extraIntents.toArray(new Parcelable[extraIntents.size()]));
14.               return chooser;
15.           }
```

Al seleccionar la imagen la actividad anterior deja de estar en pausa y se ejecuta el método onActivityResult.

```
1.      @Override
2.
      public void onActivityResult(int requestCode, int resultCode,
   Intent data) {
3.          super.onActivityResult(requestCode, resultCode, data);
4.
      if (requestCode == MEDIA_CODE && resultCode == RESULT_OK) {
5.              Log.d(TAG, "Imagen cambiada!!!!!");
6.              if (null != data && data.getDataString() != null) {
7.                  Log.d(TAG, "GALERIA");
8.                  String uri = data.getDataString();
9.                  Glide.with(this).load(uri)
10.                             .crossFade()
11.
   .diskCacheStrategy(DiskCacheStrategy.ALL)
12.                             .into(profilePicture);
13.                 } else {
14.                     Log.d(TAG, "CAMARA");
15.                     Utils.compressBitmap(imageUri, 10, parent);
16.                     Glide.with(this).load(imageUri)
17.                             .crossFade()
18.
   .diskCacheStrategy(DiskCacheStrategy.ALL)
19.                             .into(profilePicture);
20.                 }
21.             }
22.         }
```

Si la imagen guardada es efectuada desde la cámara entonces llamamos al método *compressBitMap*, donde comprimiremos la imagen a un 10% de su calidad original.

```
1.      public static Uri compressBitmap(Uri
   imageUri, int quality, Context context) {
2.          Bitmap bitmap = null;
3.          FileOutputStream os = null;
4.
5.          try {
6.
   bitmap = MediaStore.Images.Media.getBitmap(context.getContentRes
   olver(), imageUri);
7.
   os = new FileOutputStream(imageUri.getEncodedPath());
8.          } catch (FileNotFoundException e) {
9.              e.printStackTrace();
```

```
10.                 } catch (IOException e) {
11.                     e.printStackTrace();
12.                 }
13.
     if (bitmap != null) bitmap.compress(Bitmap.CompressFormat.JPEG
   , quality, os);
14.                 return imageUri;
15.             }
```

Con esto conseguiremos tanto implementar una gestión de imágenes en nuestra aplicación como hacer esta eficiente en los tiempos de carga.

---

9. Conclusiones

Conclusiones técnicas

Pese a seguir un *planning* de trabajo, donde dividí toda la implementación del proyecto en plazos de tres semanas, no me ha sido posible implementar la funcionalidad del chat con Firebase por cuestiones de tiempo. Dedique un tiempo que no estaba previsto a la optimización del tiempo de carga al mostrar imágenes. Debido a esto cuando comencé a implementar la lógica del chat vi que necesitaría bastante más tiempo del que me quedaba para conseguir la funcionalidad que necesitaba.

No he podido tampoco dedicarle el tiempo que me gustaría al *testing* de la aplicación. He ido realizando pruebas mientras programaba la aplicación para comprobar que el funcionamiento era el que quería, pero no he realizado test de usabilidad, rendimiento en distintos dispositivos o con diferentes conexiones de red, mantenibilidad, etc.

Pese a todo creo que he conseguido implementar de forma adecuada toda la funcionalidad básica que necesitaba la aplicación, dado que las funciones que me plantee desde la especificación de requisitos están presentes en el aplicativo.

Trabajo futuro

El primer paso claro seria terminar las funcionalidades que quedaron pendientes en el *backlog* del tercer *sprint*, tanto la implementación de un chat interno en la aplicación como la realización de pruebas más concienzudas. Una vez esto estuviera acabado sería interesante subir una versión Alpha de la aplicación a Google Market, tanto para escuchar la opinión de los usuarios sobre la aplicación como para tener un *feedback* de nuevas funcionalidades que el público ve necesarios en la aplicación o corrección y ajuste de las ya presentes.

Los pasos siguientes seria tanto mantener la aplicación como implementar nuevas funcionalidades y planear una vía de monetización.

Conclusiones personales

Al finalizar este proyecto he de confesar que he aprendido bastante. Desconocía prácticamente el uso de bases de datos no relaciones y he aprendido a trabajar con *APIs*. He descubierto la inmensa cantidad de librerías que hay para Android y muchas muy interesantes para conseguir una aplicación completa y optimizada. He puesto en práctica los conocimientos de la rama Ingeniería del Software para plantear, organizar y llevar a cabo un proyecto software.

Pese a no haber podido acabar la aplicación con todas las funcionalidades que me hubiera gustado estoy bastante satisfecho con el proyecto, creo que ha quedado una aplicación usable y específica, como se ha planteado desde el principio.

En resumen, he adquirido un conocimiento que sin ponerme frente a un proyecto como este solo, no hubiera podido adquirir.

Glosario

Open source: termino con el que se conoce al software distribuido y desarrollado libremente.
Sprint: es el nombre con el que se denomina a los marcos de desarrollo ágiles.
Backlog: Lista de tareas planificadas para completar en la iteración siguiente.
UT: Unidad de trabajo. Tarea única.
Product owner: uno de los futuros usuarios del sistema o alguien que sabe lo que quieren los usuarios del sistema.
Mockup: Diseño o maqueta de un dispositivo utilizado para la demostración, evaluación del diseño, etc.
Main frame: vista principal del sistema.
Sharding: técnica que consiste en particionar los datos de una base de datos horizontalmente agrupándolos de tal modo que tenga sentido y permita direccionamiento más rápido.
Activity: componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar acciones.
Fragment: representa un comportamiento o parte de ella de la interfaz de usuario en una activity.
Setter: Método para definir una propiedad en una clase.
Getter: Método para obtener una propiedad de una clase.
Adapter: Objeto que hace la función de puente entre la vista y los datos de esa vista.
Material design: es un concepto y unas pautas enfocadas al diseño utilizado en Android.
JSON: acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.
REST: Transferencia de Estado Representacional.
Feedback: Capacidad de un emisor para recoger reacciones de los receptores y modificar su mensaje de acuerdo con lo recogido.

Bibliografía

http://agilismoatwork.blogspot.com.es

https://developers.google.com

Referencias

[1] Jaime Domenech, “Impresión 3D: llega el futuro de los sistemas de producción”, Articulo, [Online], http://www.silicon.es/impresion-tridimensional-llega-el-futuro-de-los-sistemas-de-produccion-49043 , 2013.

[2] Wikiversity, “Metodologías ágiles de desarrollo software”, [Online], https://es.wikiversity.org/wiki/Metodolog%C3%ADas_%C3%A1giles_de_desarrollo_software , 2015.

[3] ricardoroldan, “SCRUM Desarrollo ágil”, Articulo, [Online], https://es.slideshare.net/ricardoroldan/11870-2323976 , 2009.

[4] josemlopez “Mejora tu trabajo en equipo con el método Kanban”, Articulo, [Online], https://hipertextual.com/archivo/2013/11/que-es-kanban/ , 2013.