Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

TEXTAT: DI ALGO SOBRE ESTE LUGAR

Alumno: Andriy Leutskyy Polishchuk

Tutor: Arturo Montejo Ráez
Dpto: Departamento de Informática

Septiembre, 2018

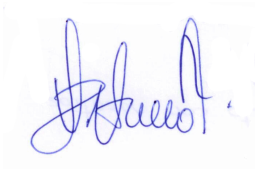
Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Arturo Montejo Ráez , tutor del Proyecto Fin de Carrera titulado: TextAt: di algo sobre este lugar, que presenta Andriy Leutskyy Polishchuk, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2018

El alumno: Los tutores:

Andriy Leutskyy Polishchuk



Firmado digitalmente por MONTEJO RAEZ ARTURO - 26026531F
Fecha: 2018.09.05 09:10:49 +02'00'

Arturo Montejo Ráez

Índice

Indice de Figuras

Indice de tablas

Agradecimientos

El final de este T.F.G. significa que por fin he acabado el Grado de Ingeniería Informática, esto no habría sido posible sin el apoyo de las siguientes personas:

Los profesores de la Universidad de Jaén que no sólo me han ayudado a entender conceptos sino que también me han exigido tener los conocimientos necesarios para realizar un proyecto de este tipo y estar preparado para entrar en el mercado laboral.

Mi tutor, Arturo Montejo Ráez, al que me gustaría agradecer su labor como docente y su facilidad para transformar mi estrés y agobio en motivación para continuar adelante con el trabajo.

Mis compañeros y amigos de la universidad que han estado conmigo durante estos 4 maravillosos años y con los que he compartido tantas horas realizando prácticas de madrugada porque no llegabamos al plazo de entrega. Algunos de ellos, al igual que estoy a punto de hacer yo, ya han acabado la carrera pero a los que aún no, les emplazo a dar un último empujón y demostrar los grandes ingenieros que son.

A mi amigo José Díaz, al que también considero uno de mis mentores en cuanto a Informática. Siempre nos quedarán las interminables clases de Álgebra y Matemáticas Discretas con Paint.

Por último y más importante: a mis padres. Que desde que yo era pequeño sabían que me gustaba la informática y que “esto era lo mío” me compraron mi primer ordenador y me han apoyado incondicionalmente no sólo los 4 años de carrera sino toda mi vida.

1. Introducción

1.1. Introducción

Hoy en día todo el mundo dispone de un smartphone y la mayoría de personas tienen conexión permanente de datos, el uso de las redes sociales no para de crecer de forma exponencial y las personas siempre quieren compartir información. Esto provoca cada año la aparición de nuevas aplicaciones que buscan ser lo nuevo, lo que se ponga de moda ya sea por incluir funcionalidades nunca vistas anteriormente o por tener detrás una gran compañía.

La geolocalización se define como el proceso de obtener la ubicación geográfica de un dispositivo basada en un sistema de coordenada. La más conocida es GPS (*Global Positioning System*). En la actualidad las aplicaciones usan esta capacidad con una gran variedad de objetivos, desde guiarte a un lugar elegido hasta para mostrar publicidad de empresas cercanas.

Aplicaciones como Google Places o Foursquare permiten una geolocalización social limitada. Esto se debe a que sí que permiten dejar un comentario o puntuación sobre un lugar en el que has estado, pero están condicionadas a hacer hincapié en enriquecer su propia base de datos con opiniones de sus usuarios más que en ofrecer una aplicación orientada al usuario en sí.

La diferencia entre las aplicaciones ya existentes en el mercado y este TFG es que, en las primeras, los usuarios están limitados a dejar ya sea una valoración o descripción de un lugar ya existente en estas plataformas. En TextAt no existirán lugares donde dejar una anotación, será posible dejar una anotación estando en cualquier parte del mundo.

1.2. Motivación

En 2018 se cumplen 10 años de la aparición de las tiendas de aplicaciones

de Android y iOS. Ya no se puede decir que sea un mercado en vías de desarrollo, según fuentes oficiales ya hay más de 2 millones de aplicaciones disponibles en cada una de ellas. Si hablamos de dinero, una de las estadísticas más impresionantes es que en más de 40 países a lo largo del mundo se gastan más de 100 millones de dólares al año en la industria de las aplicaciones móviles.

Una de las consultoras especializadas en el desarrollo y monetización de aplicaciones, *Appanie* [1], predijo que para este año los ingresos aumentarán aún más, alcanzando un total de 110 mil millones entre las principales tiendas, destacando como una de las principales fuentes de ingreso las suscripciones.

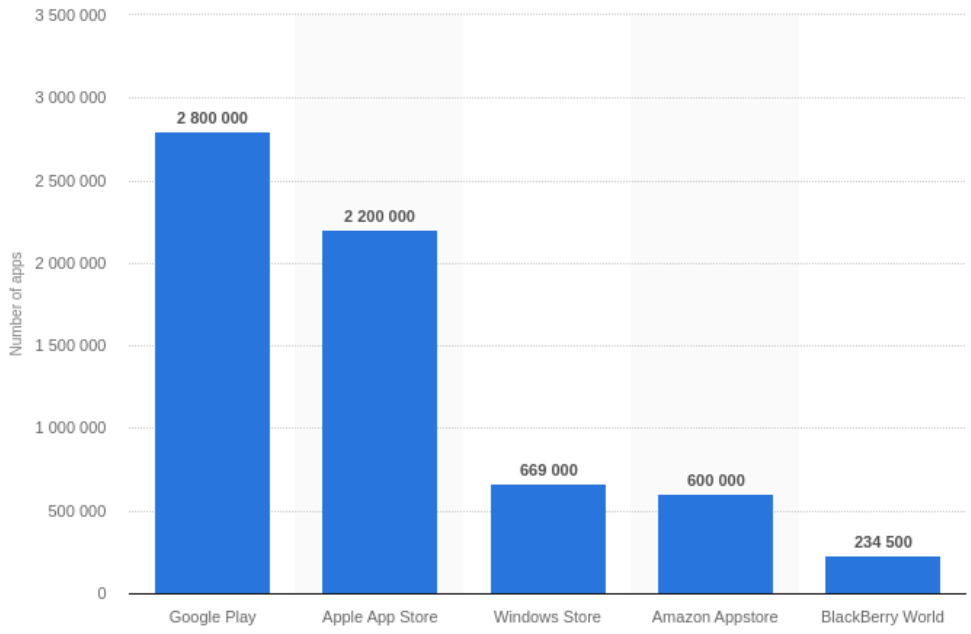


Figura 1: Número de aplicaciones disponibles en Marzo de 2017 según Statista

Stack Overflow, conocido por ser la mayor plataforma de preguntas y respuestas para la comunidad de desarrolladores informáticos, realiza cada año una encuesta a sus usuarios. Si nos fijamos en los resultados que se han obtenido este año y años anteriores, podemos ver como cada vez hay más desarrolladores especializados en aplicaciones móviles, alcanzando un 20.4% en 2018 comparado

con el 8.4% que había en 2016.

El crecimiento del número de aplicaciones y del número de desarrolladores Android no puede pasar desapercibido. Es por eso que escogí este TFG: para poder adentrarme en Android con un primer proyecto interesante.

1.3. Objetivos del proyecto

Los objetivos de este T.F.G. son:

- Diseñar e implementar una aplicación Android, **TextAt**, basada en el uso de la geolocalización que permita asociar comentarios a lugares específicos, mantenerlos privados o publicarlos y etiquetar estas anotaciones.

- Analizar el mercado de aplicaciones y extraer qué características tienen en común estos servicios que los hacen tan populares así como posibles carencias de los mismos a remediar para encontrar un nicho de usuarios que no estén satisfechos con las soluciones existentes.

- Profundizar en los dos factores más importantes tras el desarrollo y publicación de una aplicación: escalabilidad y monetización.

1.4. Metodología

Según la propia definición del TFG se debe seguir una metodología ágil, con revisiones entregables en cada iteración. Para esto la herramienta elegida ha sido Github: la mayor plataforma online de desarrollo colaborativo aprovechando así tanto su funcionalidad principal, que son los repositorios de código, como la parte de *Proyectos* que permite planificar de manera sencilla el desarrollo de un software.

Cada una de las iteraciones tiene como objetivo la implementación de una funcionalidad básica hasta alcanzar un *MVP* (proyecto mínimo viable).

1.5. Estructura de la memoria

Este documento se encuentra estructurado en tres grandes bloques:

- **Estudio inicial**: Imprescindible para el desarrollo de cualquier tipo de software. En esta parte, además de buscar soluciones con características similares al proyecto se intenta prever quiénes son los potenciales usuarios de la aplicación y cuáles son las características y funcionalidades que necesitan.

- **Desarrollo de la aplicación**: Es la mayor parte de esta memoria, relata desde el punto inicial donde el proyecto son sólo un par de ideas hasta cada uno de los diferentes estados por los que se pasa durante su desarrollo. Provee al lector de la información necesaria para entender cómo se ha desarrollado la aplicación.

- **Planificación de futuro**: El ciclo del desarrollo de una aplicación software, y más concretamente de una aplicación móvil con características sociales, no termina al publicarla en una de las tiendas disponibles. Esta parte de la memoria habla de posibles mejoras y funcionalidades futuras, de cómo poder soportar un crecimiento del número de usuarios y de cómo aprovechar estos usuarios para generar beneficios.

2. Análisis de mercado

2.1. Introducción

Dentro del mercado de aplicaciones basadas en geolocalización podemos distinguir entre aplicaciones en las que ésta es un factor clave y la aplicación no puede funcionar sin ella y otras que sólo lo usan para una parte de la aplicación siendo incluso opcional. Las más populares son las siguientes:

- **TripAdvisor**: Plataforma para la búsqueda y contratación de servicios relacionados con viajes: desde vuelos a hoteles. En TripAdvisor la geolocalización está centrada a la parte de los diferentes negocios que ofrece contratar donde los usuarios pueden dejar reseñas enriqueciendo y facilitando así la experiencia de usuario a la hora de planificar un viaje.

Figura 2: Logo Tripadvisor

- **Google Places**: Places es el servicio de Google para que empresarios de todo el mundo puedan mostrar su negocio en la web y atraer así a nuevos clientes. Integrado en Google Maps, es un factor clave para mejorar el algoritmo de búsqueda de Google ordenando establecimientos según las opiniones y valoraciones que dejen los usuarios, aunque no restringe que los usuarios deban estar en la localización en cuestión para poder realizar una reseña.

Figura 3: Logo Google Places

Cabe destacar que la propia aplicación de Google Maps manda notificaciones

a los usuarios al estar localizados en negocios incluidos en Places para que estos dejen su opinión.

- **FourSquare**: Red social creada en 2009 cuya idea principal es que los usuarios puedan hacer un *check-in* en lugares específicos, es decir, que una persona pueda decir “yo he estado aquí”. Al igual que las anteriores, una de sus características es la valoración y la recomendación de lugares, pero difiere de estas ofreciendo características de red social tales como la posibilidad de mencionar a otros usuarios en el momento de hacer check-in o la opción de ocultar contenido para que sólo los usuarios que tu desees vean dónde has estado.

Figura 4: Logo Foursquare

- **Twitter**: Servicio de microblogging fundado en 2006 que utiliza la geolocalización para establecer desde dónde se ha mandado un tuit. A diferencia de los anteriores, en Twitter la geolocalización es opcional. Una de las características más interesantes de las que dispone, no presente desde el inicio de esta plataforma, es la creación de hilos de tuits lo cual combinado con geolocalización puede mostrar una cronología de dónde ha estado un usuario durante un periodo de tiempo.

Figura 5: Logo Twitter

Teniendo en cuenta que cada una de ellas posee millones de usuarios, se puede afirmar sin duda que el mercado de la valoración de lugares utilizando geolocalización está muy saturado. Haría falta una idea revolucionaria y una gran inversión inicial para poder hacer frente a los gigantes ya existentes.

FourSquare presenta las características más interesantes aunque no estén aprovechadas del todo. Las listas de localizaciones es una herramienta muy útil para, por ejemplo, compartir recomendaciones de lugares a otras personas que está

limitada a los lugares presentes en la plataforma. Estas listas de lugares no tienen una cronología definida, lo cual podría dar lugar a usos interesantes como un recorrido guiado por una ciudad.

2.2. Propuesta

TextAt propone al usuario una aplicación de geolocalización social pura. Al entrar en la aplicación, los usuarios verán un mapa centrado en su ubicación actual en el que aparecen anotaciones públicas creadas por otros usuarios.

El usuario puede ver los detalles de estas anotaciones, las cuales tienen diferentes metadatos sobre el lugar, pueden tener menciones a otros usuarios o anotaciones y pueden ser etiquetadas con *hashtags*. Además, puede añadir anotaciones propias, incluyendo imágenes relacionadas con la ubicación y dejando una valoración que puede ser positiva, negativa o neutra.

Estas anotaciones pueden ser públicas (visibles para todo el mundo), privadas (visibles sólo para el propio creador de la anotación) o visibles sólo para usuarios cercanos.

2.3. Diferencias con el mercado

Lo que propone TextAt es poner en el foco al usuario. El usuario puede dejar una anotación en cualquier parte del mundo sin tener que estar presente ya en la plataforma. Por tanto, TextAt se aleja de soluciones limitadas a la valoración de negocios o recomendación de lugares, va un paso más allá. En este proyecto no existe diferenciación de los lugares, lo que sí existen son puntos en forma de coordenadas.

En TextAt, con el fin de recopilar opiniones honestas y descripciones precisas, es importante que sólo se usen ubicaciones visitadas por el usuario, por tanto sólo se permitirá añadir una anotación en la localización exacta donde se encuentre localizado el usuario.

2.4. Ejemplos de uso

Con sólo la idea básica de TextAt, podemos ver que abarca la mayoría de los servicios que ofrecen otras aplicaciones disponibles en el mercado, ya que al igual que estas, permite dejar una valoración sobre un lugar.

Otros ejemplos de posibles usos de TextAt son:

- **Guía turística:** Combinando el uso de de las etiquetas y de las referencias a otras anotaciones podemos simular un recorrido con un orden definido.

- **Yincana**: El término yincana se refiere a los tipos de juegos en los que se realizan pruebas en diferentes localizaciones. Además de aprovechar las referencias para pasar a cada una de las etapas de la prueba, si se aprovechan las opciones de visibilidad por proximidad, permite descubrir progresivamente la ubicación de cada etapa.

- **Geocaching**: Búsqueda de objetos o “tesoros” mediante la geolocalización. Si un usuario crea una anotación describiendo la localización del tesoro a encontrar visible sólamente a usuarios cercanos a la misma, se creará un servicio de geocaching local.

- **Integración con juegos basados en geolocalización**: Existen multitud de juegos basados en geolocalización que pueden aprovechar una aplicación de este tipo. Los usuarios de juegos como Pokemon GO pueden beneficiarse de esta aplicación para establecer los lugares a los que tienen que ir para completar misiones y mencionar a amigos para completarlas juntos.

3. Planificación del proyecto

A la hora de planificar un proyecto basado en una metodología ágil, es decir, incremental y basada en iteraciones, es imprescindible definir las tareas que serán completadas y realizar una estimación inicial de la duración de cada una de ellas. Estas tareas pasarán posteriormente a ser parte de los hitos incrementales con los que el proyecto quedará terminado.

3.1 Definición de tareas

1. **Estudio inicial sobre Android y Firebase**: Primer contacto con las tecnologías que conforman este proyecto. Esta tarea abarca desde el estudio de la documentación hasta la realización de pequeñas pruebas sobre las diferentes funcionalidades y posibilidades que ofrecen estas plataformas.

2. **Análisis del proyecto**: En este parte se hará un análisis de Ingeniería de Software en los que se diseñarán diagramas de casos de uso, se definirán los requisitos tanto funcionales como no funcionales del proyecto y se realizará un prototipo inicial de la aplicación en forma de *mock-up*.

3. **Diseño de la arquitectura del proyecto**: Definición de los modelos de datos, diseño de la comunicación entre el cliente (aplicación) y servidor para el almacenamiento y recuperación de los datos.

4. **Desarrollo e implementación**: Desarrollo de la aplicación e integración de los servicios usados de la plataforma Firebase.

5. **Pruebas y validación**: Tras la implementación de la aplicación se realizarán pruebas de integración de forma que se verifique que todo funciona correctamente.

6. **Redacción de la memoria**: Es imprescindible documentar el desarrollo de un

proyecto de este tipo, es por esto que esta tarea se realizará en paralelo al desarrollo de la aplicación con el objetivo de ser lo más detallada posible.

3.2 Estimación de tiempos

Tabla 1: Tabla de las tareas del proyecto

Tarea	Tiempo estimado
Estudio inicial sobre Android y Firebase	5 días
Análisis del proyecto	5 días
Diseño de la arquitectura del proyecto	10 días
Desarrollo e implementación	15 días
Pruebas y validación	2 días
Redacción de la memoria	15 días

3.3 Diagrama de Gantt

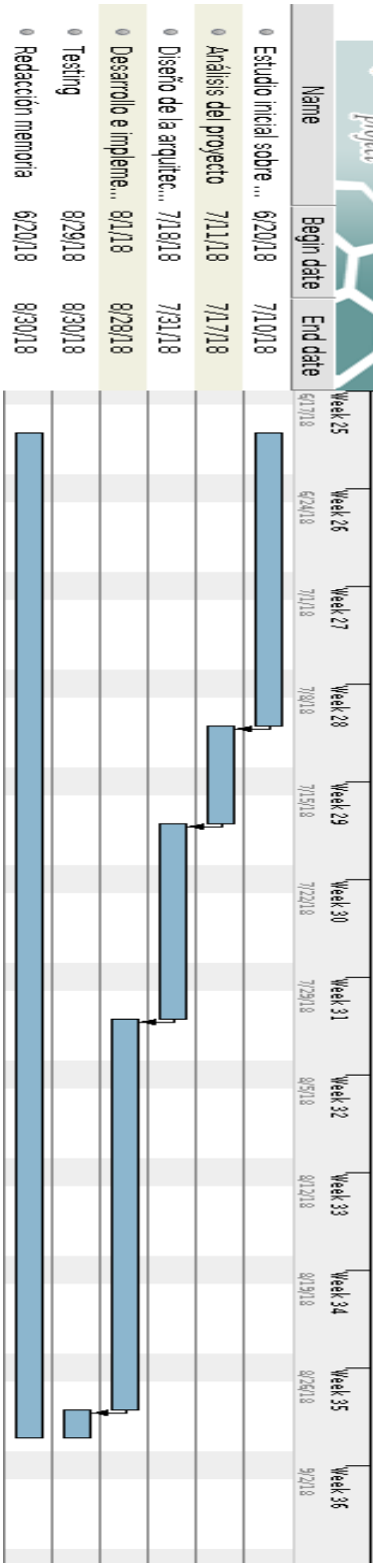


Figura 6: Diagrama de Gantt

3.4 Análisis de costes

En todo proyecto es necesario realizar una estimación de los costes tanto humanos como materiales. Además del trabajo a realizar, que ha sido definido en secciones anteriores, tenemos que establecer también el hardware, software y las horas de trabajo de las personas involucradas que serán necesarios para poder dar una estimación final del coste total del proyecto.

El software usado para desarrollar este proyecto será siempre gratuito, por tanto el coste de este apartado será nulo.

Algunas de las herramientas que ofrece Firebase como la autentificación son totalmente gratuitas, otras tienen planes de pago que que se activan tras superar los límites de los planes gratuitos. Dentro de éstas últimas están incluídas Firestore, usado para almacenar la información de la aplicación y Storage, usado para almacenar las imágenes de los usuarios.

Tabla 2: Planes ofrecidos por Firestore

Funcionalidad	**Límites del plan gratuito**	**Costes del plan de pago**
Almacenamiento	1 GB	0.18 $ / GB
Ancho de banda	10 GB / mes	Dependiente de la región
Escrituras de documentos	20000 operaciones/ día	0.18 $ / 100k operaciones
Lecturas de documentos	50000 operaciones/ día	0.06 $ / 100k operaciones
Borrado de documentos	20000 operaciones/ día	0.02 $ / 100k operaciones

Tabla 3: Planes ofrecidos por Storage

Funcionalidad	**Límites del plan gratuito**	**Costes del plan de pago**
Almacenamiento	5 GB	0.026 $ / GB
Ancho de banda	1 GB / día	0.12 $ / GB
Operaciones de carga	20000 operaciones / día	0.05 $ / 10k operaciones
Operaciones de descarga	50000 operaciones / día	0.004 $ / 10k operaciones

Por otra parte, en cuanto al hardware son necesarios dos dispositivos: un ordenador para desarrollar y un móvil con sistema operativo Android para poder probar la aplicación dado que el uso intensivo de GPS es incompatible con el emulador de Android Studio.

El ordenador usado será un Asus K55VM del año 2012 y el teléfono móvil un Lenovo Zuk Z2 del año 2016. Si la vida útil de un ordenador portátil es de unos 5 años y la de un teléfono móvil 2 años, podemos considerar que ambos equipos están totalmente amortizados.

El proyecto será realizado por una única persona que llevará acabo todos los roles: analista, programador, tester, etc.. Según el sitio web Indeed [8] un analista programador tiene un sueldo medio bruto anual de 26.852 € o lo que es lo mismo, 1.918 € al mes. Si la duración de este proyecto es aproximadamente 60 días estamos hablando de un coste humano total de 3.836 €.

4. Análisis del proyecto

En esta sección estarán incluidos los diagramas de casos de uso y el análisis de requisitos, estableciendo tanto los funcionales como los no funcionales.

4.1 Casos de uso

Los diagramas de casos de uso permiten definir en formato UML la interacción entre usuario y sistema: especifican la comunicación y el comportamiento que tiene el sistema ante determinadas acciones.

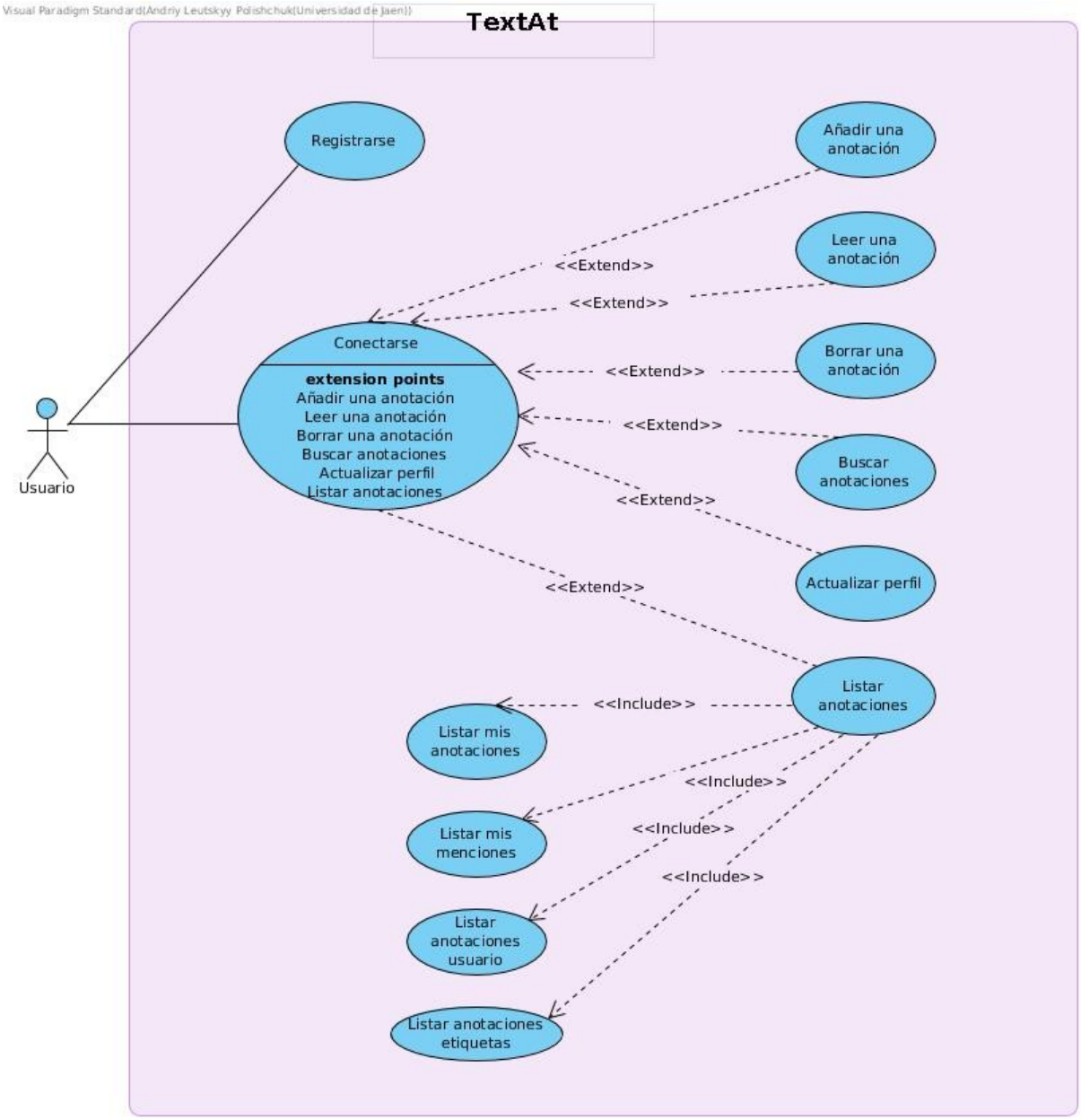


Figura 7: Diagrama casos de uso

4.2 Análisis de requisitos

4.2.1 Requisitos funcionales

Basándonos en los casos de uso presentados anteriormente, TextAt tendrá los siguientes requisitos funcionales:

- Los usuarios podrán crear anotaciones y referenciar en estas a otros usuarios, anotaciones o hashtags.
- Los usuarios podrán ver anotaciones.
- Los usuarios podrán eliminar sus propias anotaciones.
- Los usuarios podrán registrarse y conectarse a la aplicación mediante cuentas sociales ya existentes o mediante correo electrónico.
- Los usuarios podrán especificar y cambiar a posteriori su nombre de usuario y avatar.
- Los usuarios podrán realizar distintos listados anotaciones:
    - Las creadas por el usuario actual
    - Las públicas creadas por un determinado usuario
    - Las públicas vinculadas a un determinado *hashtag*
    - Aquellas en las que el usuario es mencionado
- Los usuarios podrán realizar búsquedas de anotaciones por contenido.

4.2.2 Requisitos no funcionales

TextAt tendrá los siguientes requisitos no funcionales:

- **Estética de la aplicación:** TextAt deberá tener una interfaz intuitiva que facilite el aprendizaje del uso de la aplicación.

- **Usabilidad:** TextAt deberá tener un tiempo de respuesta aceptable para cada una de las acciones que pueda realizar el usuario.

- **Portabilidad:** TextAt deberá funcionar correctamente en cualquier smartphone basado en un sistema operativo Android con API igual o superior

a 21.

- **Rendimiento**: TextAt deberá cuidar el consumo de recursos del sistema entre las que se encuentran CPU, RAM, uso de datos y el gasto de batería.

4.3 Prototipo

A la hora de diseñar una aplicación es importante hacer hincapié en el diseño de una interfaz familiar para los usuarios que permita a estos usar los conocimientos adquiridos con el uso de otras aplicaciones y le permita moverse por las diferentes partes de la aplicación sin problema.

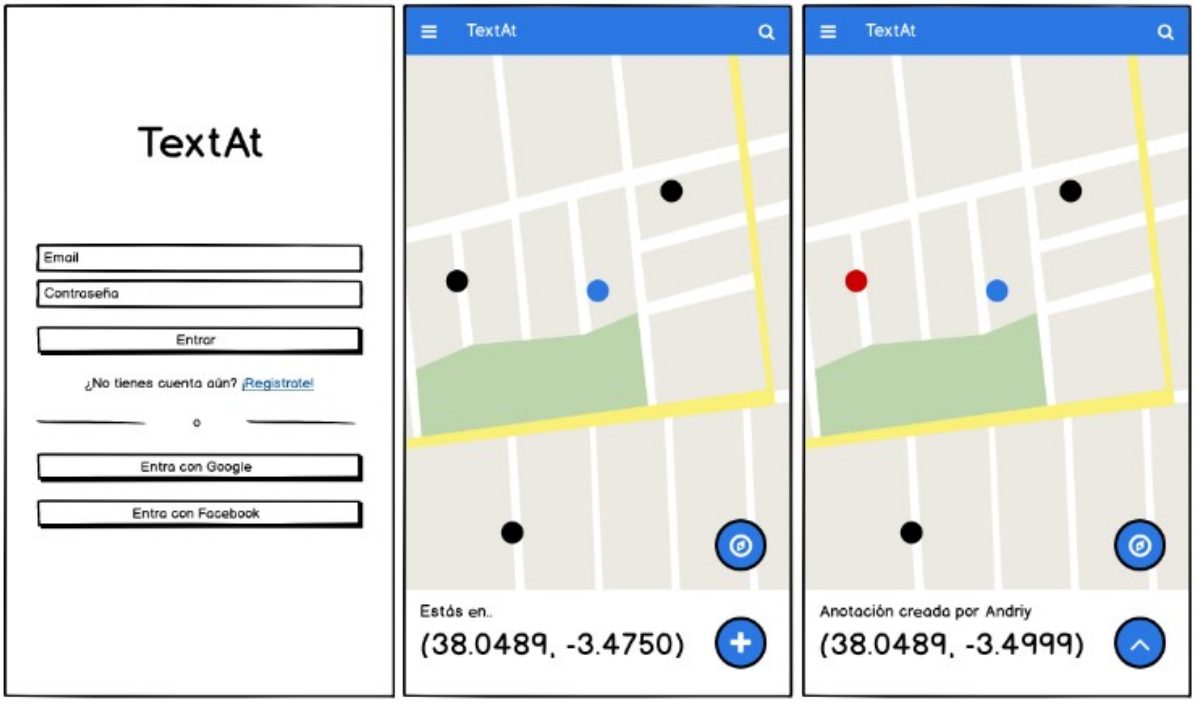


Figura 8: Login y Pantalla principal

En la Figura 8 se observa a la izquierda la pantalla de conexión del usuario en la que podemos ver cómo el usuario puede acceder a la aplicación tanto usando su e-mail como mediante cuentas en las principales redes sociales.

En las otras dos se muestra la pantalla principal, en la que el usuario podrá

ver las distintas anotaciones en el mapa. En el momento de inicio de la aplicación, la cámara se centrará en la ubicación actual del usuario. Si se hace click en una de las anotaciones existentes en el mapa nos mostrará un resumen de esa anotación y podremos acceder a los detalles de dicha anotación haciendo otro click en ella. En la barra inferior, a la izquierda, aparecen las coordenadas GPS correspondientes a la localización del usuario. A la derecha, se muestra un botón que permite al usuario agregar una nueva anotación.

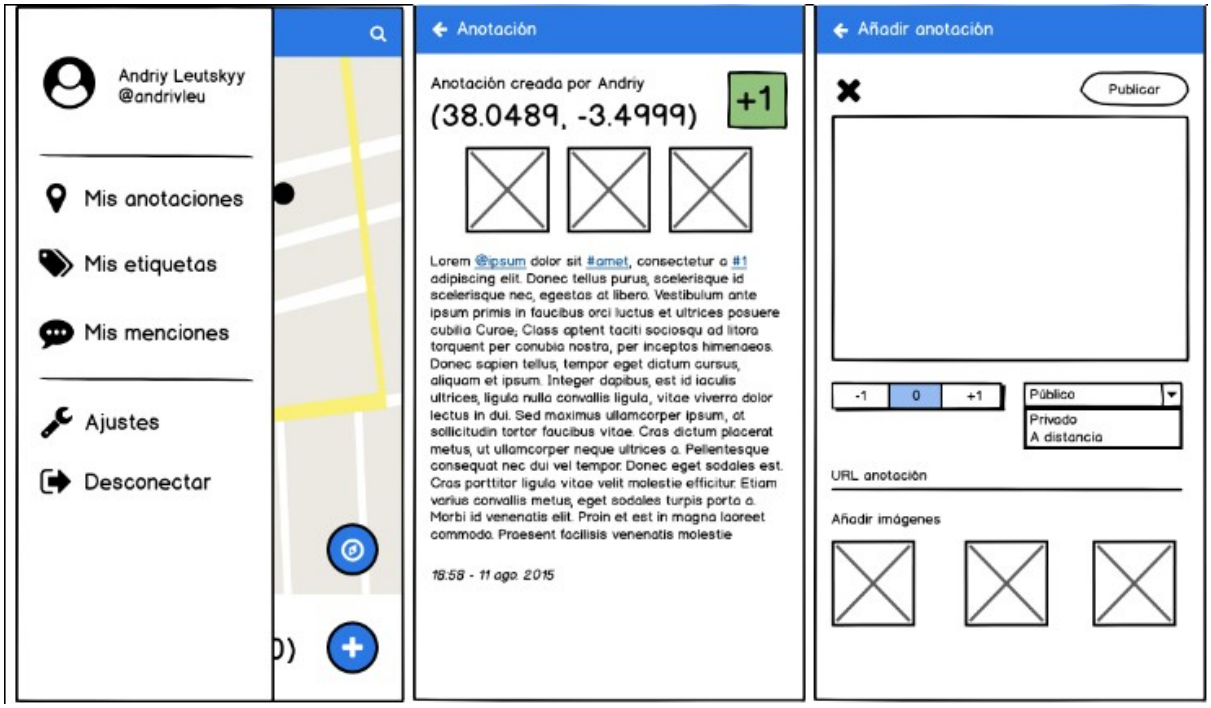


Figura 9: Sidebar, detalles de una anotación y vista para agregar una anotación.

En la primera imagen de la Figura 9 observamos el menú lateral que se abre al realizar un gesto arrastrando desde la izquierda de la pantalla hasta el centro. Dispone de varios apartados que permiten al usuario acceder a diferentes pantallas:

- Mis anotaciones: Lista todas las anotaciones del usuario.
- Mis etiquetas: Lista los hashtags usados por el usuario.
- Mis menciones: Lista las menciones del usuario.
- Ajustes: Permite al usuario cambiar de nombre de usuario y avatar.
- Desconectar: Permite al usuario salir de su cuenta.

La segunda imagen muestra una vista con todos los detalles disponibles para una anotación. Se puede incluir un máximo de tres imágenes por anotación y la descripción de éstas puede contener menciones a otros usuarios (que llevarán a una vista con las anotaciones públicas de dicho usuario), una referencia a otra anotación (que puede considerarse como un enlace para acceder a dicha mención) y un *#hashtag* para etiquetar la anotación.

En la última imagen podemos ver la interfaz que permite al usuario crear una nueva anotación. Si el usuario desea añadir una anotación que sólo sea visible por usuarios cercanos, deberá seleccionar esa opción en el menú desplegable que muestra las distintas opciones de privacidad. En ese momento se dará, en esta misma vista, la opción de elegir la distancia máxima a la que es visible la anotación, de entre los valores disponibles (50, 500 y 1000 metros).

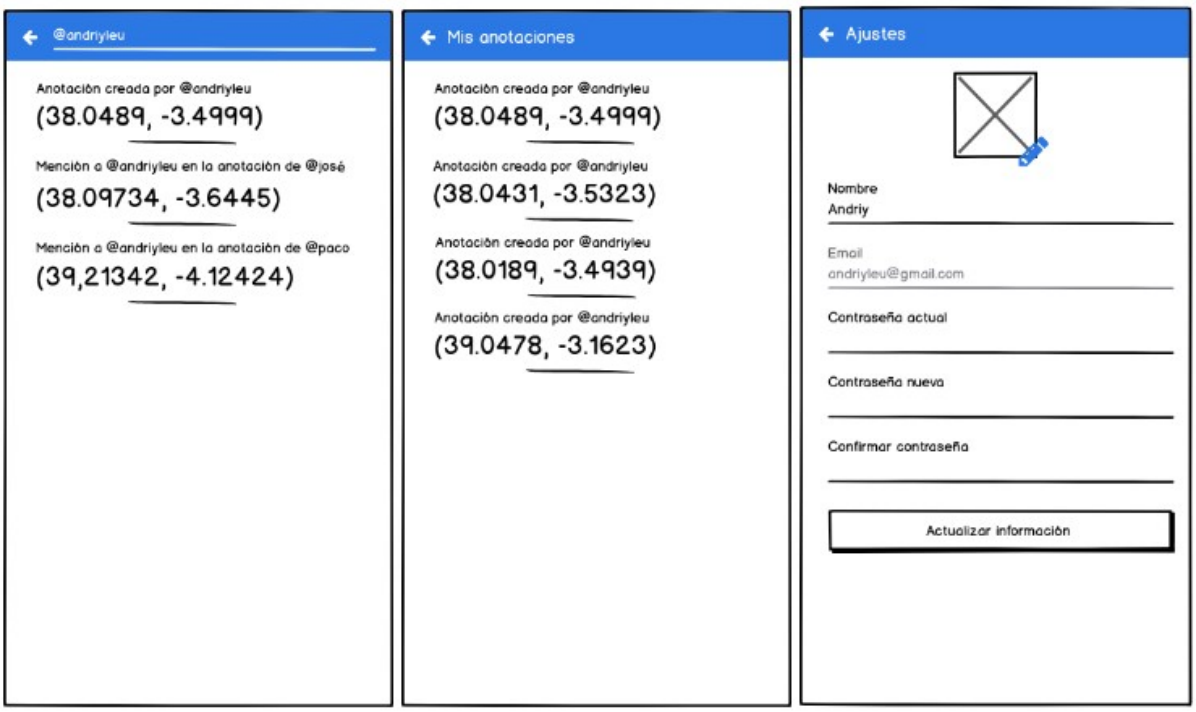


Figura 10: Listados de anotaciones y ajustes de usuario.

Las dos primeras imágenes mostradas en la Figura 10 parecen similares a primera vista, ya que se aprovecha la misma interfaz. Estas tres vistas son las que aparecen al elegir una de las tres primeras opciones que se muestran en el menú izquierdo de la primera imagen de la Figura 9. Las dos primeras listan un conjunto de anotaciones, mostrando los detalles básicos de cada una. Al hacer click en una anotación de la lista, se mostrará la interfaz que presenta todos sus detalles (segunda imagen en la Figura 9).

La vista de ajustes contiene las funciones básicas que un usuario necesita realizar en una aplicación de tipo social: actualizar su nombre de usuario, contraseña o avatar.

5. Diseño e implementación

Esta sección está dividida en varias partes empezando por una descripción más en detalle de las tecnologías usadas en este proyecto y si procede alguna característica que esté relacionada con ellas y sea relevante. A continuación se explica la arquitectura que siguen las aplicaciones Android y cómo se implementa en TextAt, especificando además los modelos de datos usados en la aplicación.

La parte más interesante de este apartado corresponde con los ejemplos de implementación en los que se han elegido diferentes trozos de código de funcionalidades que merece la pena destacar.

Por último se detalla las pruebas y validaciones que se han realizado para comprobar la funcionalidad del proyecto.

5.1 Tecnologías y librerías

5.1.1 Android

Android es un sistema operativo para dispositivos móviles mayoritariamente creado por Google en 2008 basado en Linux. Está escrito en Java, C y C++. La mayoría de las aplicaciones que se desarrollan para esta plataforma se escriben en Java, y se desarrollan utilizando el IDE Android Studio.

Figura 11: Logo Android

5.1.2 Git

Git es un software de control de versiones diseñado por Linus Torvalds en 2005. Es una de las herramientas más importantes en el arsenal de un programador cuando tiene un proyecto con infinidad de archivos y varias personas están trabajando en el mismo proyecto.

Figura 12: Logo Git

Para este proyecto se usará la plataforma web Github.com para tener una copia en todo momento en la nube.

5.1.3 Android Studio

Android Studio es el IDE (*Entorno de desarrollo integrado*) más usado para desarrollar aplicaciones de Android. Además de integrar Git, también permite incorporar al proyecto de una forma rápida las herramientas de Firebase desde el propio menú del IDE.

5.1.4 Firebase

Firebase, al igual que se ha dicho antes, es una plataforma BaaS que ofrece multitud de servicios. Dentro de las funciones de Firebase de las que hace uso TextAt podemos encontrar:

- **Firebase Auth**: Sistema que permite identificar a los usuarios utilizando código únicamente en el lado del cliente. Como punto negativo no permite el uso de nombres de usuarios únicos por defecto, será necesario crear una base de datos y evaluar en el registro si el nombre de usuario indicado por el usuario ya existe.

Figura 13: Logo Firebase

- **Firebase Firestore**: *Firestore* es una base de datos NoSQL que se caracteriza por ser flexible, escalable y en la nube. Está optimizada y dispone

de funciones para poder sincronizar información en el lado del cliente y del servidor de forma veloz.

Si los usuarios de tu aplicación necesitan disponer de búsqueda sobre la información almacenada en este servicio, los propios desarrolladores de Firebase recomiendan el uso de un servicio externo para esta tarea. Las dos opciones principales son: Algolia, que es un servicio que al igual que Firebase dispone de unos límites de uso gratuitos y a continuación pago por uso, y ElasticSearch que tiene tanto un servicio igual al de Algolia como el propio código para que puedas desplegarlo.

- **Firebase Storage**: No sólo para mejorar la interactividad y utilidad de la aplicación añadiendo la posibilidad de agregar imagenes a las anotaciones sino también para permitir al usuario personalizar su cuenta con un avatar es necesario el uso de Storage. Al igual que las otras herramientas de esta plataforma su integración es sencilla y directa.

5.1.5 Algolia

Algolia ha sido la herramienta escogida para realizar la tarea de búsqueda en texto. La forma de trabajar con Algolia consiste en *queries* (consultas) que se realizan de forma asíncrona al servidor de Algolia que al igual que Firestore utiliza una base de datos NoSQL.

Dispone de cliente web en el que puedes hacer consultas y probarlas. Hay que tener en cuenta de que uno de los problemas que tendrás si intentas integrar este servicio en una aplicación ya existente es que necesitarás un script que migre la información de Firestore a Algolia.

Es posible especificar en las propias queries diferentes tipos de filtros para refinar la búsqueda del usuario. Además desde la interfaz web podrás elegir el orden de los resultados de búsqueda.

5.1.6 Librerías

En este apartado se especifican las diferentes librerías de terceros de las que hace uso este proyecto:

- **Firebase**: El paquete de Firebase que existe para Android incorpora una gran variedad librerías que implementan cada una de las funciones que ofrece la plataforma. En nuestro caso usaremos Firestore y Storage.

- **FirebaseUI:** Librería que implementa el *flow* de *login* con diferentes proveedores. Hay que tener en cuenta que en el caso de usar proveedores externos a Google es posible que sea necesario obtener una clave API desde el proveedor y añadirla en el proyecto.

- **Facebook Login**: Librería necesaria para poder incluir Facebook como uno de los métodos disponibles para loguearse.

- **Play Services**: Librería usada para obtener actualizaciones de la ubicación del usuario.

- **Play Maps**: Permite añadir fragmentos que contienen un mapa a tus actividades.

- **Android Maps Utils**: Esta es la librería que permite realizar Map Clustering agrupando las anotaciones en clusters haciendo la interfaz más usable.

- **Glide**: Permite utilizar fácilmente las imágenes almacenadas en Storage en la aplicación asignando de forma automática la URI que devuelve una petición a un ImageView.

- **AutoLinkTextView**: Esta librería se encarga de añadir *listeners* a las diferentes menciones o referencias que puedes encontrar en una anotación.

- **Algolia Search**: Se usa para realizar búsquedas por texto en la información almacenada referente a las anotaciones.

- **PrettyTime**: Humaniza las fechas con las que trabaja la aplicación a la hora de mostrarlas al usuario. Convierte el formato técnico de fechas utilizado por Java a uno más intuitivo al usuario. *("Publicado.. h*ace unos instantes, hace una semana")

5.2 Arquitectura de la aplicación

La arquitectura de una aplicación Android se basa en un conjunto de *activities*[1], cada una de las cuales representa una interfaz gráfica disponible para el usuario. En otras palabras, es precisamente en las *activities* donde ocurre la interacción entre el usuario y la aplicación.

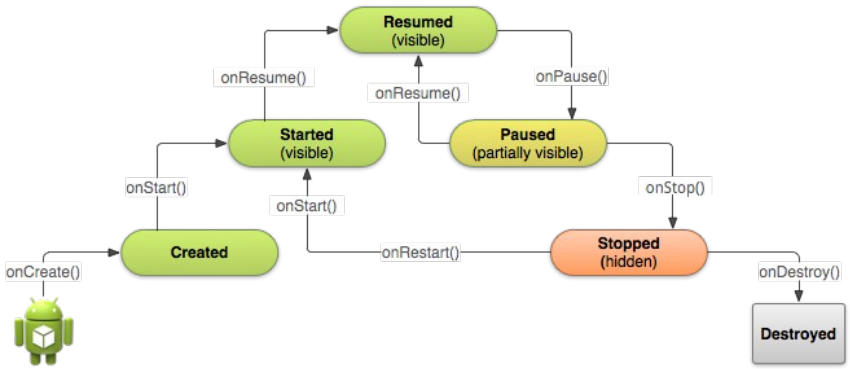


Figura 14: Ciclo de vida de un actividad

Cada una de estas pantallas con las que el usuario puede interactuar tiene un ciclo de vida con las diferentes etapas que se muestran en la Figura 5.2.1. Estas etapas corresponden con las acciones habituales que realizan los usuarios en una aplicación: iniciarla, que corresponde con *onCreate*, dejar la aplicación en segundo plano, correspondiente a *onPause*, volver a la aplicación, *onResume*, cerrar la aplicación, *onDestroy*. Los distintos *callbacks* disponibles permiten realizar acciones

1 Android Activities-
https://developer.android.com/guide/components/activities?hl=es-419

en cada uno de estos momentos.

Las activities están formadas por dos partes: gráfica y lógica. La gráfica se define mediante un fichero en formato XML que organiza los distintos elementos gráficos que componen la activity, mientras que la lógica es una clase Java que extiende la clase Activity e implementa la funcionalidad de cada elemento de la vista.

Los *intents* son una descripción abstracta de una operación a realizar. Tienen tres usos principales: iniciar una actividad, iniciar un servicio y la entrega de mensajes. También permiten transportar información (tipos de datos primitivos y clases que implementen Parcelable) para su uso por parte del receptor del *intent*. Android mantiene una pila de actividades para poder gestionar de forma sencilla algo tan típico como permitir al usuario volver atrás, es decir, a la actividad anterior.

TextAt está formado por las siguientes actividades:

- **LoginActivity**: La aplicación comenzará su funcionamiento por esta actividad cuyo objetivo es permitir al usuario hacer login y registrarse en la aplicación a partir de cuentas existentes en las principales redes sociales o correo electrónico. En el caso de que un usuario se *loguee* de forma correcta y tenga un nombre de usuario asignado se cargará MainActivity mediante un intent que contendrá el usuario logueado. Si por el contrario el usuario no tuviera un nick asignado se iniciará *SettingsActivity* para asignar su nombre de usuario.

- **MainActivity**: En esta actividad el usuario podrá ver el mapa centrado en su ubicación y las anotaciones que estén alrededor. Asi mismo, desde ella también podrá pasar a todas las demás *activities* existentes en la aplicación. Contiene un botón en la parte inferior que iniciará *AddMarkActivity,* por otra parte al hacer click en una de las anotaciones en el mapa podrás ir a *MarkDetailActivity*. MarkListActivity y SettingsActivity serán accesibles desde las opciones existentes en el menú lateral de la interfaz.

- **AddMarkActivity**: Está *activity* permite al usuario añadir anotaciones al sistema. MainActivity pasa a esta actividad la información relacionada al usuario y las coordenadas en el momento de hacer click en el botón de añadir una anotación. Los demás atributos que tiene una anotación son rellenados por el usuario manualmente en el formulario que contiene esta vista.

- **MarkDetailActivity**: Esta actividad es la encargada de mostrar los detalles y atributos de una anotación a los usuarios. La interfaz mostrará los siguientes elementos: título, descripción, *timestamp*, privacidad, visibilidad, puntuación, autor, coordenadas geográficas y descripción, en la que pueden existir menciones o referencias otros usuarios, anotaciones o hashtags. Si se contiene alguna referencia, al hacer click en ella se iniciará *MarkDetailActivity* o un *MarkListActivity*. Si el usuario es el creador de la anotación podrá eliminarla desde esta actividad.

- **MarkListActivity**: Permite listar anotaciones en función de un criterio concreto. Esta actividad será la encargada de funcionalidades como “Mis anotaciones”, “Mis menciones”, “Anotaciones de @usuario” y “#hashtag”.

- **SettingsActivity**: Actividad que muestra los ajustes de los que dispone el usuario, con la que se puede actualizar la información relacionada con su perfil. Esta actividad puede ser iniciada desde *MainActivity* al elegir la opción *Ajustes* o tras registrarse, ya que con esta actividad el usuario puede especificar su nombre de usuario.

5.3 Modelo de datos

Para gestionar las anotaciones, TextAt, dispone de una clase Mark de tipo POJO[2] que extiende a Parcelable, para poder adjuntar esta clase como Extra en Intents y ClusterItem, de forma que las anotaciones puedan ser añadidas en el mapa.

Al ser una clase POJO implementa getters y setters para cada uno de los atributos de la clase. Los atributos que tiene esta clase son:

- **Rating**: La puntuación es un atributo de tipo long que toma 3 posibles valores: -1 (Puntuación negativa), 0 (Puntuación neutra) y 1 (Puntuación positiva).

- **Privacy**: Al igual que rating es otro atributo de tipo long que representa los tres casos de privacidad: 0 (Pública), 1 (Privada) y 2 (Visible a distancia).

- **Visibilidad**: El valor de este atributo sólo es usado en el caso de que privacy sea 2. Puede tomar los valores 50, 500 y 1000. Estos valores representan la distancia en metros a la que hay que estar como máximo de la anotación para poder visualizarla.

- **Timestamp**: Fecha y hora en la que se crea la anotación.

- **Location**: Ubicación de la anotación.

- **User**: Nombre de usuario del reador de la anotación.

- **Uri**: Atributo opcional que almacena un enlace a la entidad.

- **Descripción**: Permite describir el lugar donde se deja la anotación e incluir menciones y etiquetas. Se almacena con un string y los enlaces son detectados automáticamente por la librería AutoLinkTextView.

- **Título**: Campo obligatorio limitado a 80 caracteres que da nombre a la anotación.

- **hasImages**: Este campo no visible al usuario indica si la anotación tiene adjuntas imágenes. En ese caso será necesario hacer una query Storage.

- **ID**: Es el identificador único de cada una de las anotaciones, generado automáticamente por Firestore.

A la hora de almacenar y recuperar la información relacionada con la aplicación de Firebase no es necesario realizar creación de tablas. Firestore es una base de datos NoSQL que funciona con solo dos elementos: colecciones y documentos. Por tanto bastará con crear una [2]colección, por ejemplo, anotaciones y añadir en ella un documento. Una clase como Mark, que cumple las características de una clase POJO, es decir, que tiene definidos getters y setters para todos sus atributos podrá ser almacenada directamente en Firebase a partir de estos.

En Firestore, además de los atributos típicos como cadenas de caracteres o enteros, encontramos otros más complejos como GeoPoint que nos permitirán almacenar coordenadas.

2 Clases POJO - http://carlospesquera.com/que-es-un-pojo-ejb-y-un-bean/

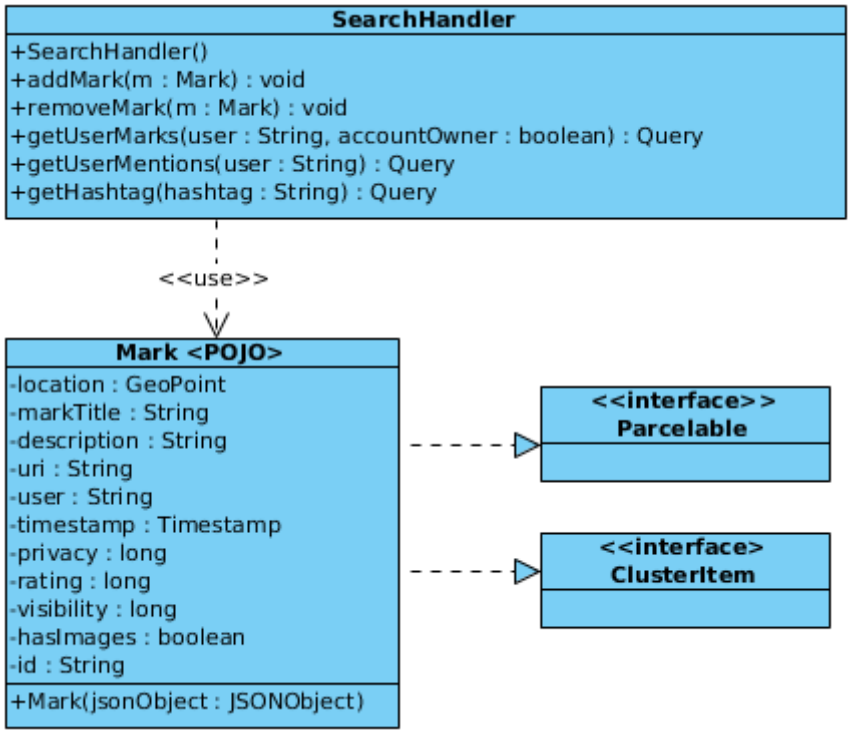


Figura 15: Diagrama UML que muestra la clase Mark

Para almacenar anotaciones necesitaremos una colección a la que ir añadiendo documentos. En la Figura 5.3.2 se presenta un ejemplo.

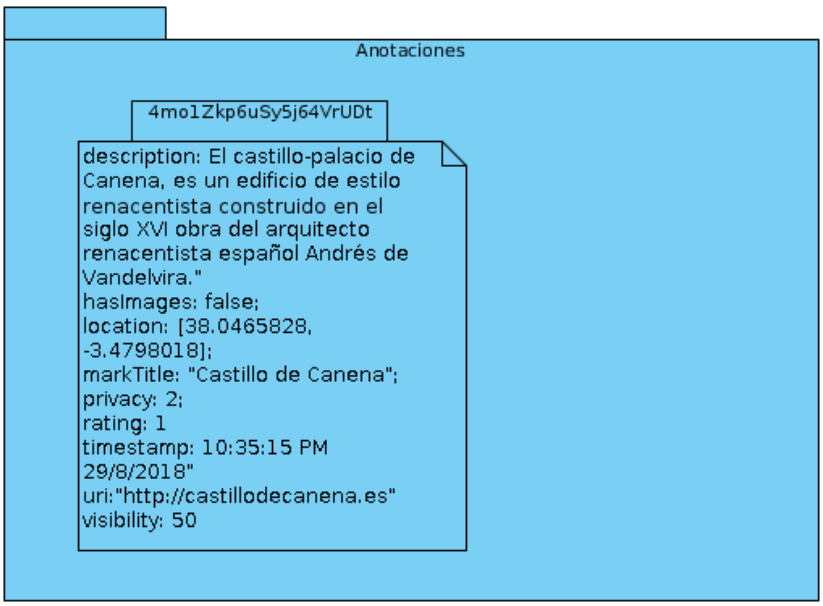


Figura 16: Representación colección Anotaciones

La clase SearchHandler, que servirá de *helper* para Algolia,t implica la creación de un constructor en la clase Mark que permite construir un objeto a partir de un JSONObject. Al crear una nueva anotación esta será añadida tanto a Firestore como a Algolia. Los diferentes métodos que hay en la clase SearchHandler permiten hacer consultas predefinidas que busquen por contenido y que además estén filtradas por atributos.

Aparte de la colección de anotaciones, existen otras dos que sirven para gestionar usuarios. Aunque Firebase Auth almacene la información que devuelvan los proveedores,, para que un usuario pueda disponer de un nombre de usuario único hace falta disponer de una colección llamada usuarios que habrá que consultar para ver si un nombre de usuario ya está ocupado. Para asociar la UID (Identificador de usuario) generada por Firebase Auth a un nombre de usuario hará falta el uso de otra colección llamada UIDs que contenga objetos con el nombre de usuario escogido como atributo.

5.4 Implementación

5.4.1 Estructura del proyecto

Al crear un proyecto Android se generan de forma automática multitud de ficheros, no sólo ficheros *.java* que contienen código fuente. Por tanto podemos distinguir dos grandes secciones en este proyecto: archivos que pertenecen al paquete java de la aplicación clasificados en actividades, *helpers*, *handlers* y modelos, y archivos que forman parte de los *resources* que pueden ser a su vez clasificados en *layouts*, *drawables*, menús y *values*.

5.4.1.1 Actividades

La función de cada una estas actividades ya ha sido comentado en el apartado 5.2.

- **MainActivity.java**

- **LoginActivity.java**
- **AddMarkActivity.java**
- **MarkDetailActivity.java**
- **MarkListActivity.java**
- **SettingsActivity.java**

5.4.1.2 Helpers

- **CustomListAdapater.java**: Por defecto un ListView en Android sólo tiene un título y un subtítulo los cuales son insuficientes para la cantidad de información que contiene una anotación. Esta clase se crea con el propósito de mostrar título, coordenadas, fecha de publicación y autor en cada elemento de la lista mostrada.

- **CustomClusterRenderer.java**: Esta clase extendiende la clase DefaultClusterRenderer que es la encargada de decidir si un elemento debe o no estar en un cluster. Con esto lo que se pretende es tanto editar este algoritmo para permitir que se creen Clusters con menos de los elementos permitidos por defecto como poder cambiar los iconos de las marcas en el mapa de forma dinámica según su valoración.

5.4.1.3 Handlers

- **MapHandler.java**: Para añadir un mapa a un proyecto Android basta con añadir un *MapFragment* a la actividad donde vaya a ser mostrado. Al tener que interactuar con Firebase y dado el hecho de que los puntos serán manejados dentro de un *ClusterMap* se ha decidido crear este *fragment* que realizará este trabajo y dejar separado las actividades con sólo los métodos relacionados con la lógica de la interfaz.

- **SearchHandler.java**: En esta clase se crean los atributos necesarios para poder hacer queries a Algolia y se definen varios métodos con queries por defecto útiles para la aplicación que veremos más

adelante.

5.4.1.4 Modelos

- **Mark.java**: La clase Mark como se ha dicho anteriormente representa el modelo de datos correspondiente a una anotación por tanto contiene los atributos necesarios para poder representar este objeto en su totalidad. Además, tienen algunos métodos aparte que ayudan a la hora de presentar la información como pueden ser *getDate()* que devuelve un String con la fecha de creación en formato humano.

5.4.1.5 Resources

- ***Drawables***: Contienen los iconos e imágenes usados en la aplicación. Se incluyen tanto imágenes *raster* como vectorizadas.

- ***Layouts***: Son los archivos .XML que definen las vistas relativas a las *activities* de la aplicación.

- **Menús**: Los menús deben ser definidos de forma independiente en archivos aparte del layout en el que van a estar contenidos.

- **Values**: En este apartado se almacenan los archivos que contienen los distintos literales que aparecen en la aplicación y los archivos de estilo en los que se definen colores y temas.

5.4.2 Ejemplos de implementación

En esta apartado se comentarán algunas partes de la implementación que me parezcan interesantes.

- **Conexión y registro de usuarios**: La conexión y el registro de los usuarios

con FirebaseUI es muy sencilla. Consiste solamente en definir los proveedores desde los que el usuario podrá loguearse y lanzar una actividad con resultado, de todo lo demás se encarga la propia librería.

Cuando tengamos el resultado de esta actividad disponible podremos comprobar si el usuario ha conseguido hacer login en cuyo caso comprobaremos si tiene un nombre de usuario ya asignado o no. Si el usuario no tiene un nombre de usuario asignado, se iniciará la actividad SettingsActivity que permitirá al usuario elegir un nombre, en caso contrario simplemente se iniciara MainActivity.

```
List<AuthUI.IdpConfig> providers = Arrays.asList(
       new AuthUI.IdpConfig.EmailBuilder().build(),
       new AuthUI.IdpConfig.FacebookBuilder().build(),
       new AuthUI.IdpConfig.GoogleBuilder().build());

// Start intent with the specified providers
startActivityForResult(
       AuthUI.getInstance()
               .createSignInIntentBuilder()
               .setTheme((R.style.LoginTheme))
               .setLogo(R.drawable.textat)
               .setAvailableProviders(providers)
               .build(),
       RC_SIGN_IN);

@Override
    protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == RC_SIGN_IN) {
            IdpResponse response = IdpResponse.fromResultIntent(data);

            if (resultCode == RESULT_OK) {

                final FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();


db.collection("uids").document(user.getUid()).get().addOnCompleteListener
(new OnCompleteListener<DocumentSnapshot>() {
                    @Override
                    public void onComplete(@NonNull
Task<DocumentSnapshot> task) {
                        if (task.isSuccessful()) {
                            DocumentSnapshot d = task.getResult();
                            if (d.exists()) {
                                // User logged and has username asigned,
```

```
start MainActivity
                                switchToHome(user);
                            }
                            else {
                                switchUserSettings(user);
                            }
                        }
                    }
                });
        }

    }
```

- **Recuperado de anotaciones**: En cuanto al recuperado de puntos, se declara un listener a la colección llamada “Anotaciones” en Firestore que realiza dos funciones: descargar todas las anotaciones al iniciar la aplicación y recibir actualizaciones que ocurran en esta colección, es decir, si son añadidos puntos nuevos o alguno de los existentes es eliminado.

  Las anotaciones se reciben y son almacenadas en dos estructuras de datos, un HashMap que almacenará todos los puntos disponibles para el usuario por id de anotación y un ArrayList que tendrá solamente aquellos puntos que sea necesario comprobar si deben ser mostrados según la localización del usuario y su cercanía al punto. Esto último es debido a que el servicio de localización de Android no se inicia instantáneamente, por tanto almacenamos los puntos para poder procesarlos cuando éste esté disponible.

  Por último los puntos también son añadidos o eliminados del *Cluster* que será la clase encargada de mostrar los puntos en el mapa. Tras hacer una operación con el Cluster se llama al método cluster para que el mapa sea regenerado si es necesario.

```
// Set up listener that parses initially everything and then receive
updates
updateListener = db.collection("anotaciones")
       .addSnapshotListener(new EventListener<QuerySnapshot>() {
           @Override
           public void onEvent(@Nullable QuerySnapshot snapshots,
                               @Nullable FirebaseFirestoreException e) {
```

```
                if (e != null) {
                    Log.w("TAG", "listen:error", e);
                    return;
                }

                for (DocumentChange document :
snapshots.getDocumentChanges()) {
                    Log.w("TAG", "Añadiendo " +
document.getDocument().getId(), e);
                    DocumentSnapshot d = document.getDocument();
                    String id = d.getId();
                    Mark m = document.getDocument().toObject(Mark.class);
                    m.setId(id);
                    switch (document.getType()) {
                        case ADDED:

                            // Case: Private marks
                            if (m.getPrivacy() == 1 && !
m.getUser().equals(usernick)) {
                                continue;
                            }

                            // Case: Nearby marks
                            if (m.getPrivacy() == 2) {
                                marks.put(id, m);
                                nearbyMarks.add(m);
                                continue;
                            }

                            mapHandler.getmClusterManager().addItem(m);
                            marks.put(id, m);
                            break;

                        case REMOVED:

mapHandler.getmClusterManager().removeItem(marks.get(id));
                            marks.remove(id);
                            Iterator<Mark> i2 = nearbyMarks.iterator();
                            while (i2.hasNext()) {
                                Mark mark = i2.next();
                                if (mark.getId() == id) {
                                    nearbyMarks.remove(i2);
                                }
                            }
                    }
                }
                mapHandler.getmClusterManager().cluster();
            }
        });
```

- **Preparación del mapa e interacción entre usuario y mapa**: El siguiente código es ejecutado tras llamar al *getMapAsync* de *SupportMapFragment* y representa el momento en el que el mapa ya está listo y podemos trabajar

con él.

Al hacer uso de clusters para evitar agrupaciones de anotaciones en lugares muy cercanos debemos, delegar la mayoría de funciones del mapa al propio ClusterManager. Además se define tanto lo que pasa al hacer click en un cluster (calcular los límites geográficos del conjunto de elementos pertenecientes al mismo y mover la cámara de tal forma que contenga solamente esos elementos) como lo que pasa al hacer click en uno de los elementos, que iniciará MarkDetailActivity mostrando los datos del elemento correspondiente.

```
@Override
public void onMapReady(GoogleMap googleMap) {

   if (map == null) {

        map = googleMap;
        mClusterManager = new ClusterManager<>(getActivity(), googleMap);
        map.setOnMapLoadedCallback(this);


        map.setMyLocationEnabled(true);
        map.setOnCameraIdleListener(mClusterManager);
        map.setOnMarkerClickListener(mClusterManager);
        map.setOnInfoWindowClickListener(mClusterManager);
        map.getUiSettings().setMapToolbarEnabled(false);

        // CustomRenderer in order to decrease number of elements needed
to start clustering

        CustomClusterRenderer renderer = new
CustomClusterRenderer(getActivity(), map, mClusterManager,
map.getCameraPosition().zoom, map.getMaxZoomLevel());
        map.setOnCameraMoveListener(renderer);

        mClusterManager.setRenderer(renderer);
        if (getActivity() instanceof MarkListActivity) {
            ((MarkListActivity) getActivity()).setMarks();
        }

        mClusterManager
                .setOnClusterClickListener(new
ClusterManager.OnClusterClickListener<Mark>() {
                    @Override
                    public boolean onClusterClick(final Cluster<Mark>
cluster) {
                        LatLngBounds.Builder builder =
LatLngBounds.builder();
                        for (ClusterItem item : cluster.getItems()) {
```

```
                            builder.include(item.getPosition());
                        }

                        final LatLngBounds bounds = builder.build();

getMap().animateCamera(CameraUpdateFactory.newLatLngBounds(bounds, 100));
                        return true;
                    }
                });

        mClusterManager.setOnClusterItemInfoWindowClickListener(new
OnClusterItemInfoWindowClickListener<Mark>() {
            @Override
            public void onClusterItemInfoWindowClick(Mark mark) {
                Intent intent = new Intent(getActivity(),
MarkDetailActivity.class);
                intent.putExtra("mark", mark);
                intent.putExtra("id", mark.getId());
                startActivity(intent);
            }

        });
    }
}
```

- **Búsqueda por contenido**: Dentro de la clase SearchHandler hay unos métodos predefinidos que permiten a las demás clases hacer diferentes tipos de búsquedas en Algolia.

  En el siguiente fragmento de código podemos observar dos tipos diferentes de queries que se realizan a este servicio. La primera se usa para buscar dentro del atributo descripción de las anotaciones públicas de otros usuarios menciones al usuario que se pase por parámetro y la segunda filtra las anotaciones por usuario aprovechando la arquitectura de Algolia, pudiendo definir por parámetro si se deben devolver tanto públicas como privadas o sólo las públicas.

```
// Get user mentions
public Query getUserMentions(String user) {
   Query q = new Query(user);
   q.setRestrictSearchableAttributes("description");
   q.setFilters("privacy: 0");
   return q;
}

// Returns everything with user at user field in Algolia
```

```
public Query getUserMarks(String user, boolean accountOwner) {
   Query q = new Query(user);
   q.setFacets("user");

   if (!accountOwner) {
       q.setFilters("privacy: 0");
   }

   return q;
}
```

5.5 Pruebas y validación

Las pruebas y validaciones que se llevan a cabo tras cada tarea tienen como objetivo comprobar que el software desarrollado funciona sin errores y verificar que se han cumplido todos los casos de uso que fueron establecidos. De esta forma se evita que la experiencia sea nefasta para los usuarios y se asegura que se cumplen todos los objetivos.

5.5.1 Pruebas unitarias

Durante el desarrollo del proyecto tras la implementación de cada una de las características de la aplicación se realizaron pruebas unitarias que permitieron asegurar su funcionamiento correcto.

Estas pruebas han permitido descubrir varios bugs en la funcionalidad que han sido arreglados a posteriori. A continuación comentaremos algunos de ellos:

- Se descubrió que, aunque estaba implementado el manejo de anotaciones que sólo son visibles a una determinada distancia, el código no actualizaba de forma correcta la visibilidad de este tipo de puntos. Esto sucede porque el servicio que es necesario para ubicar el dispositivo no está disponible en los primeros instantes del inicio de la aplicación.

  La solución consistió en comprobar si esta disponible este servicio en el momento de inicio y en caso contrario almacenar este tipo de

anotaciones en una lista para su posterior gestión cuando el servicio GPS esté disponible.

- Tras el diseño de la actividad que permite al usuario leer en detalle la información sobre una anotación, se descubrió que en ocasiones, al llegar a esta actividad, la aplicación se cerraba presentando un mensaje de error. Hay que tener cuenta que esta actividad era llamada mediante un *intent* que almacenaba la anotación. Tras debugear el código se detectó que el problema era producido en la lectura de la anotación tras iniciarse la actividad MarkDetailActivit, ya que el orden de de lectura de los atributos era distinto al usado en la escritura. A la hora de extender la clase Parcelable y definir el constructor y los métodos obligatorios es necesario que el orden de lectura y escritura de los atributos del objeto sea el mismo.

5.5.2 Validación final

Una vez terminado el desarrollo de la aplicación, se llevó a cabo una validación final con el objetivo de asegurar que todas las funcionalidades implementadas e independientemente probadas se integraban correctamente.

Además del dispositivo usado durante todo el desarrollo para esta tarea se usaron otros 2 dispositivos de la marca Xiaomi cuyo sistema operativo es Android: Xiaomi MI A1 y Xiaomi Redmi 5 Plus.

Estos dispositivos fueron usados de manera simultánea comprobando que al añadir o eliminar una anotación en uno de los dispositivos ésta apareciera de forma instantánea en la pantalla principal que contiene el mapa de los otros dispositivos. Se detectó que las anotaciones que aparecen a determinada distancia se actualizan de forma más rápida en el Xiaomi MI A1 porque el GPS de este dispositivo se inicia más rápido que en Redmi 5 Plus.

Tras probar extensivamente cada una de las funcionalidades existentes en la

aplicación se llegó a la conclusión de que la aplicación funciona correctamente en los 3 dispositivos. Además, al ser dispositivos con resoluciones diferentes era importante comprobar que la interfaz se adapta bien y es responsiva.

6. Explotación: Escalabilidad y monetización

Aunque el proceso de Ingeniería de Software no acaba ya que siempre es posible continuar añadiendo nuevas funcionalidades y arreglando futuros problemas que puedan surgir en un software. Si este proyecto fuera un proyecto de una empresa en búsqueda de usuarios fieles y beneficios para la compañía los siguientes pasos serían claros: añadir la aplicación a Google Play Store, intentar atraer usuarios y estudiar diferentes formas de monetizar la aplicación.

Otra idea que también hay que tener presente es que si se ha realizado un estudio de mercado y se ha llegado a la conclusión de que la idea en mente pueda dar beneficios es porque pertenece a un nicho en el que hay usuarios activos. Entra en juego la escalabilidad, que se define como la capacidad de un negocio, o en nuestro caso, aplicación de crecer en magnitud cuando crezca el trabajo a realizar, sin degradar la eficacia ni la eficiencia.

6.1. Escalabilidad

TextAt se beneficia de la plataforma Firebase ya que ésta, en su servicio Firestore, ofrece unas cualidades muy interesantes en cuanto a escalabilidad se refiere.

Firestore es el hermano mayor de Firebase Realtime Database y las diferencias que tiene frente a este son precisamente la introducción de una escalabilidad automática que no tiene límites en comparación con la segunda que son 100.000 conexiones simultáneas y 1.000 escrituras por segundo. Firestore es capaz de escalar automáticamente a cualquier número conexiones y escrituras.

Las características de Firestore hablan por sí solas: réplica de datos multiregión, consistencia de información garantizada, operaciones atómicas en lote y soporte de transacciones.

6.2. Monetización

Tras añadir la aplicación a Google Play Store, desde la propia documentación de Google nos recomiendan varias tipos de opciones entre las que se incluye:

- **Compras directas en la aplicación (*In-app purchases*)**: Esta **opción** nos permite ofrecer a los usuarios funcionalidades adicionales que no se incluyen por defecto. De esta forma la aplicación sería gratuita para los usuarios, ofreciendo funcionalidades extra a aquellos a los que les interesara. Una de las **opciones** más llamativas que barajamos en este apartado será la unión entre la **opción** de mostrar publicidad y permitir a los usuarios pagar por eliminar los anuncios.

- **Suscripciones**: Permite que los usuarios puedan pagar una cuota fija para disfrutar de la aplicación. Esto genera un abanico de posibilidades: ofrecer promociones a los usuarios para que puedan disponer de un tiempo probando la aplicación de forma gratuita al igual que hacen otros servicios como Netflix y beneficiandonos mientras tanto del feedback que nos proporcionen.

- **Publicidad:** Los anuncios en aplicaciones y páginas web son la forma más usada para monetizar estas. Hoy en día el negocio de la publicidad está en una fase muy avanzada en el que los contenidos mostrados a los usuarios son a partir de la actividad que genera en Internet incrementado así la posibilidad de que lo mostrado en un anuncio sea del agrado del usuario y se interese por él.

- **Aplicación de pago**: Consiste en establecer un precio para la aplicación en la tienda de aplicaciones para que el usuario antes de poder descargarla y usarla tenga que pagar ese precio una única vez.

7. Conclusión

7.1 Conclusión final

Como broche final a este proyecto, me gustaría dar una reflexión referente a si he disfrutado programando este proyecto, mi opinión sobre programar tanto para dispositivos móviles como para dispositivos Android concretamente y la puntuación que le pongo a Firebase tras indagar en él.

Respecto a lo primero, puedo afirmar que sí, aunque un proyecto de este tipo es siempre bastante estresante al fin y al cabo Android es una tecnología en mi opinión muy avanza y bien diseñada. En cambio la experiencia con Android Studio fue menos gratificante y he sufrido algún que otro bug.

La plataforma Firebase me parece impresionante. TextAt delega toda la gestión de la infraestructura, acondicionamiento de recursos, escalabilidad y orquestación en esta plataforma y la integración de esto en la aplicación ha sido muy sencilla. El tier gratuito permite tener aplicaciones con una cantidad de usuarios razonables funcionando a coste cero. Por tanto esta no será la última vez que use esta plataforma.

Si un T.F.G. no tiene solo por objetivo que un alumno demuestre lo que ha aprendido durante 4 años de universidad sino también para aprender cosas nuevas puedo decir que me alegro mucho de haber escogido este proyecto. He aprendido tanto de desarrollo de aplicaciones para teléfonos móviles como el uso de servicios basados en la nube que eran dos ramas de la informática que me parecían muy interesantes.

7.2 Mejoras futuras

Aunque la realización de un T.F.G. es una actividad que llena de gran satisfacción a la persona que lo realiza factores como la falta de tiempo al estar en el último año de carrera pueden ser clave para lograr un producto bueno y que

funciona a la perfección pero no es un software a la altura de otras aplicaciones del mercado que tienen grandes compañías y multitud de ingenieros detrás.

Dentro de las funcionalidades que echo en falta a este proyecto son principalmente dos: hacer aún más social la aplicación fomentando permitiendo a los usuarios dejar comentarios a anotaciones de otros usuarios. La aplicación final cumple los requisitos especificados por la memoria pero este es un paso más allá con el que poder competir en el mercado. Otra elemento de este mismo tipo es la falta de personalización de la interfaz. Hoy en día las aplicaciones sociales como Instagram o Twitter además de tener una interfaz usable es agradable a la vista.

Personalmente he intentado cuidar el aspecto de la aplicación respetando el tipo de diseño que tienen apps similares en la tienda de aplicaciones pero echo en falta un toque que, además de ser diferente en estilo a las creadas por defecto en Android Studio, dé personalidad que la diferencie del resto.

Por otra parte y siguiendo la ideología de que todo puede ser mejorado, me hubiese gustado dedicar más tiempo a la optimización de la aplicación mejorando así el rendimiento de esta. Herramientas como Android Profiler[3] permiten depurar tu aplicación proporcionando datos en tiempo real relacionados con la CPU, la memoria y la actividad de red de tu app. De la misma forma con la comunicación que hace la aplicación con Firebase y Algolia, intentando optimizar el número de conexiones y de queries a estas plataformas.

3 Android Profiler - https://developer.android.com/studio/profile/android-profiler?hl=es-419

8. Manual de usuario

Al estar ante una aplicación no disponible en Google Play Store se instalará directamente con el archivo *.apk* generado con Android Studio. Para ello, es necesario conceder los permisos para instalar aplicaciones provenientes de orígenes desconocidos.

Encontraremos la opción “Fuentes desconocidas” disponible en el apartado de Seguridad dentro del menú de ajustes de Android. Tras este habilitar dicha opción, lo único restante es descargar la aplicación y ejecutarla.

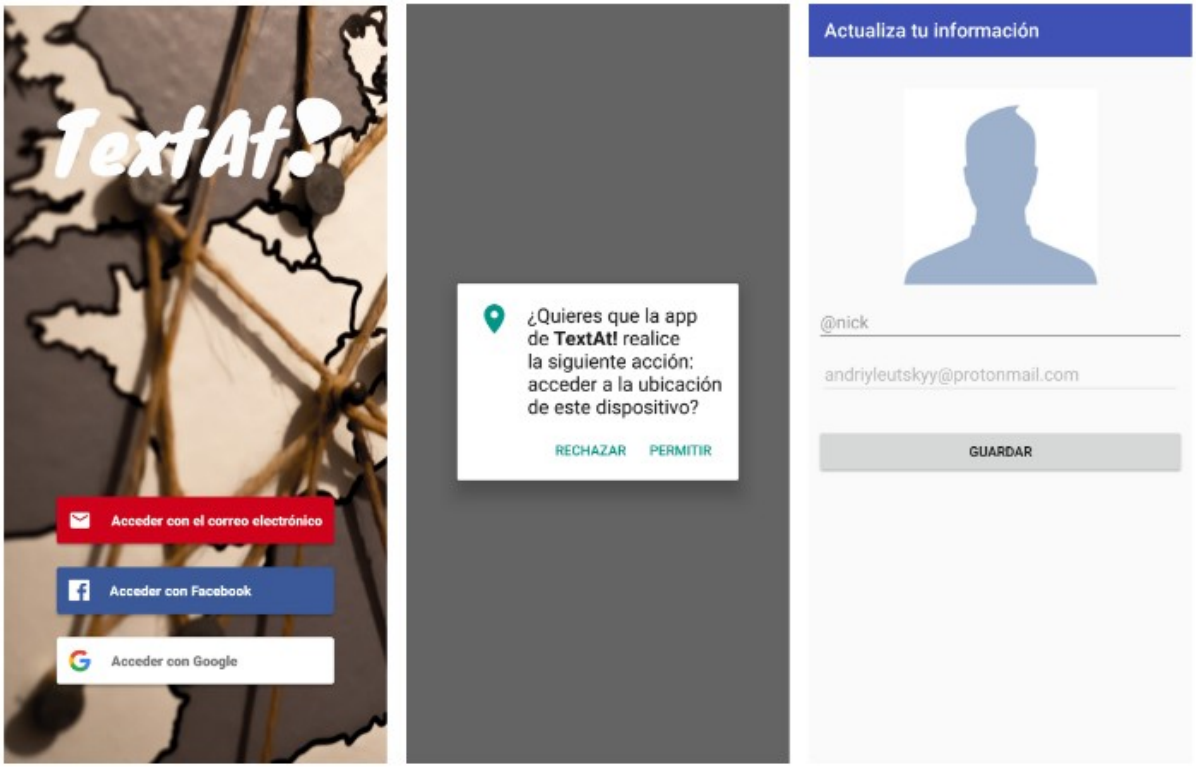


Figura 17: Inicio de aplicación, request de permisos al usuario y elección de nombre de usuario

Al iniciar la aplicación el usuario se encontrará con la posibilidad de elegir entre loguearse usando su correo electrónico, o su cuenta en Google o Twitter. En el caso de que sea la primera vez que inicie sesión en TextAt, el usuario tendrá que

completar la información requerida (nombre usuario, avatar, etc...). Antes de terminar el proceso de identificación y pasar a la pantalla principal se le pedirán al usuario los permisos necesarios para poder usar el servicio de localización.

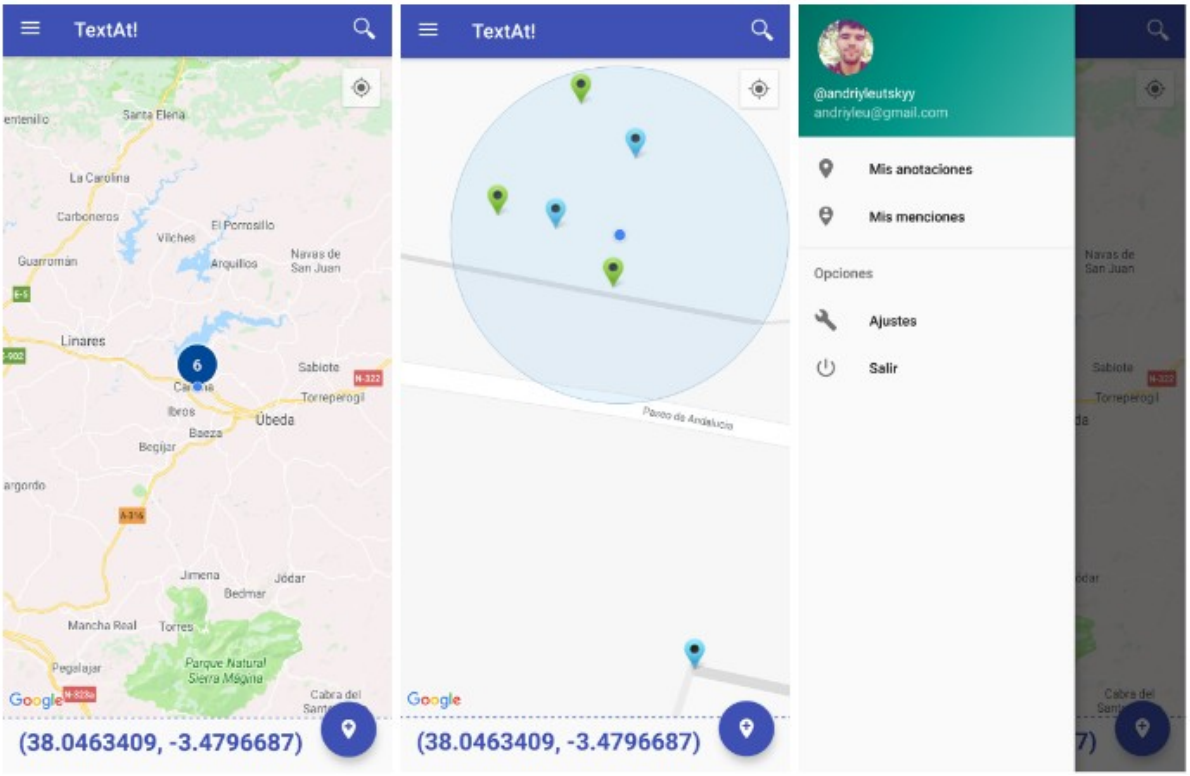


Figura 18: Pantalla principal, pantalla principal tras hacer click en un cluster y menú lateral.

En la pantalla principal tenemos varios elementos con los que podemos interactuar. Empezando por el mapa donde el usuario verá tanto su localización actual como anotaciones. Al hacer click en una de ellas aparecerá un bocadillo con su título y descripción. Si presiona encima pasará a ver todos los detalles.

En la parte de abajo encontrará un botón que le permitirá añadir una anotación en su ubicación actual, reflejada en el texto situado a la izquierda de este botón.

El menú lateral permite al usuario moverse entre las diferentes anotaciones que tienen relación con él, ya sea por propias (“Mis anotaciones”) o por ser anotaciones donde le hayan mencionado (“Mis menciones”). Además, tendrá acceso a los ajustes y a desconectarse de la aplicación.

El elemento restante de la interfaz principal es el buscador, situado en la parte de la derecha de la barra de navegación. Si el usuario busca algo en él, la interfaz cambiará para mostrar los resultados de la búsqueda en tiempo real.

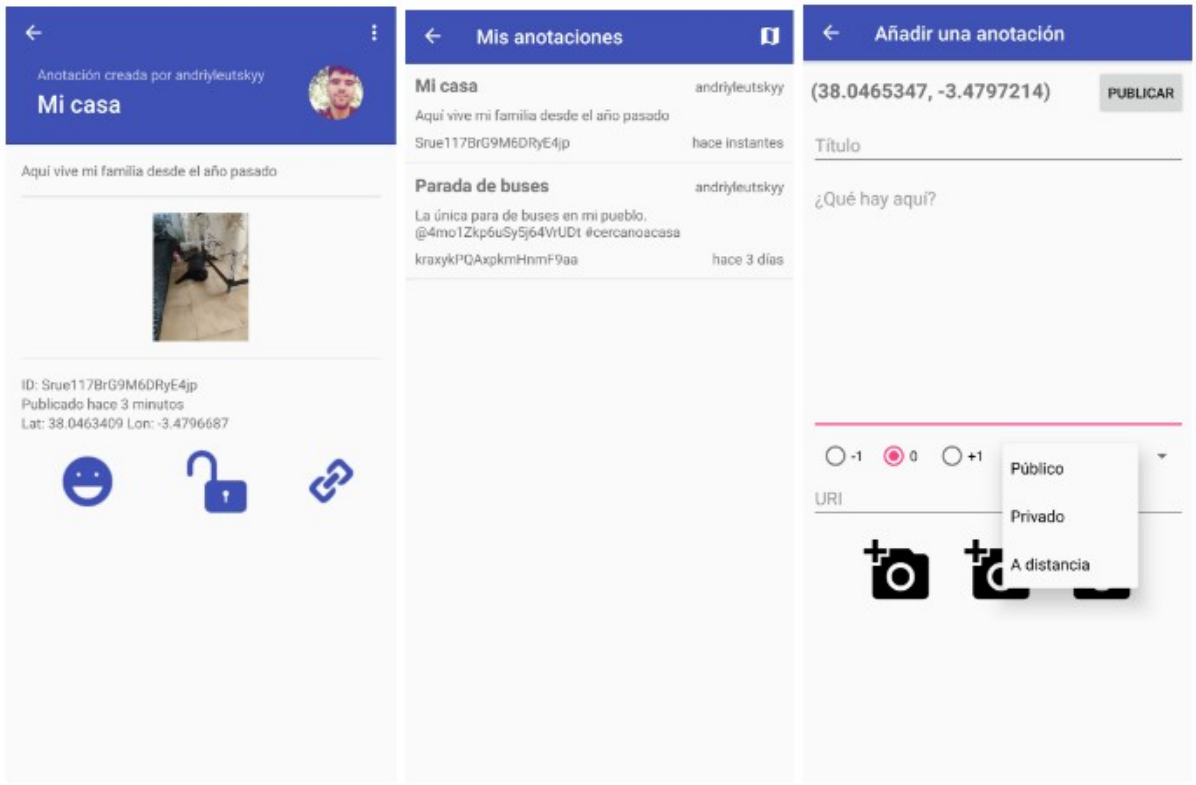


Figura 19: Detalles de una anotación, listado de anotaciones y vista que permite añadir anotaciones.

En la pantalla de detalles de una anotación el usuario podrá leer éstos, borrar la anotación desde el menú de opciones en el caso de ser suya, copiar el ID de la anotación haciendo click en el metadato, acceder a las imágenes en pantalla completa y hacer click en el icono del enlace para abrirlo en un navegador externo. La descripción puede contener enlaces a otras anotaciones, usuarios o etiquetas.

La pantalla de añadir una anotación contiene un formulario donde el usuario tiene que rellenar obligatoriamente tanto título como descripción. La descripción permite al usuario etiquetar la anotación y mencionar a otros usuarios y anotaciones:

- Para etiquetar una anotación el usuario deberá escribir una almohadilla (#) seguido el nombre con el con el que se desee etiquetar.

- Para mencionar a otros usuarios deberá escribir un arroba (@) seguido del nombre de usuario que desee mencionar.

- Para hacer referencia a otra anotación deberá escribir un arroba (@) seguido del ID de la anotación que desee mencionar. Para facilitar esta tarea puede copiar el ID haciendo click en en el metadato correspondiente en los detalles de esa anotación.

Es posible añadir tantas menciones y etiquetas en una anotación como se desee.

El usuario podrá elegir entre las distintas opciones disponibles para la privacidad de la anotación y escribir una enlace con más información sobre la anotación si lo considera necesario.

Las opciones de “Mis anotaciones” y “Mis menciones” mostrarán una vista similar a las que se encuentre el usuario tras hacer click una mención o etiqueta. Esta vista consiste en una lista de anotaciones con un resumen de los metadatos de cada una de ellas. El usuario podrá interactuar con esta lista haciendo click en cualquiera de los elementos o cual mostrará la anotación seleccionada en el mapa. El usuario podrá realizar las mismas acciones disponibles que en la pantalla principal.

La pantalla de ajustes permitirá al usuario cambiar de nombre de usuario y avatar.

Si el usuario desea cerrar sesión deberá hacer click en la última opción disponible del menú de la pantalla principal.

9. Referencias

[1] App Annie Content - Top Predictions for the App Economy in 2018. Disponible en línea: *https://www.appannie.com/en/insights/market-data/predictions-app-economy-2018/*

[2] Stack Overflow - Stack Overflow Developer Survey 2018 - Disponible en línea: *https://insights.stackoverflow.com/survey/2018/#developer-profile*

[3] Statista - Number of apps available in leading app stores as of 1st quarter 2018 - Disponible en línea: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[4] lifewire - How Many Apps Are in the App Store? - Disponible en línea: https://www.lifewire.com/how-many-apps-in-app-store-2000252

[5] Stack Overflow - Stack Overflow Developer Survey 2016 - Disponible en línea: https://insights.stackoverflow.com/survey/2016#developer-profile-developer-occupations

[6] Indeed - Sueldos en Analista programador/a en España - Disponible en línea: https://www.indeed.es/salaries/Analista-programador/a-Salaries