ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

Aplicación móvil para la gestión de viveros virtuales de plantas

AUTOR: Manuel Salvador Olmos Giménez

DIRECTORES: Fernando Cerdán Cartagena y Diego García Sánchez

Julio /2017

AGRADECIMIENTOS

Quiero agradecer a mis directores de proyecto, Fernando y Diego, su dirección para poder llevar a cabo este proyecto y por permitirme continuar con este reto y sobre todo por ayudarme a poder presentarlo con tan poca antelación.

Agradezco a mi familia que me haya ayudado a llegar hasta aquí, con sus ánimos, su esfuerzo, sus consejos y sobre todo con su constante exigencia, para que no me conforme y lleve este trabajo hasta el final.

También quería agradecer su esfuerzo y ayuda a mis compañeros de trabajo y amigos, que han soportado mis agobios, lamentos y quejas, y además me han ayudado a probar la aplicación y han aceptado aparecer en la aplicación de una forma u otra.

Por último, quiero dar las gracias de manera especial a Guille, ya que sin él este proyecto no sería el que es, gracias por todas las horas que ha invertido en este proyecto, convirtiéndolo en algo tan suyo como mío, con sus conocimientos y sus detalles, para convertir esta aplicación en algo vivo y tangible.

"Sólo podemos ver un poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer"

Alan Turing

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE PARTES DE CÓDIGO

1. INTRODUCCIÓN

En este capítulo se dará entrada al resto del documento, proporcionando una visión general sobre el proyecto y explicando los objetivos y motivaciones que me han llevado a la realización del mismo, así como una pequeña descripción del resto de contenidos.

1.1 Planteamiento inicial y motivación

La realización de esta plataforma surge de la necesidad de una herramienta para poner en contacto y situar distintas variedades hortofrutícolas autóctonas, así como a sus propietarios, para permitir el conocimiento e intercambio de estas variedades.

El problema que se intenta resolver con esta plataforma es la dificultad que tienen estos aficionados o interesados a la agricultura autóctona para conseguir ciertas variedades no comerciales, que difícilmente pueden encontrar en una tienda o en un vivero. Con este sencillo propósito se ha desarrollado este proyecto, que consistirá en la creación de la plataforma interactiva de búsqueda y localización de especies para la creación del denominado vivero virtual. Además, para hacerlo aún más comercial, existe la posibilidad de añadir y compartir también diferentes especies modernas.

De aquí surge el proyecto bautizado como TreeMapMobile, que busca integrar herramientas tecnológicas con la naturaleza.

Figura 1: *Logo proyecto TreeMapMb*

1.2 Objetivos

El presente Trabajo Fin de Máster se centra en el estudio de la plataforma Android y demás tecnologías utilizadas alrededor de esta plataforma para llevar a cabo el desarrollo de una aplicación sencilla e intuitiva para ser usada por todo tipo de usuarios sin importar sus conocimientos, que ofrezca la posibilidad de registrar y visualizar todo tipo de árboles sobre un mapa para simplificar la interfaz.

Entre otras cosas, los principales objetivos del presente trabajo son los siguientes:

1. Crear una solución que permita a los usuarios comunicarse con otros interesados en la agricultura y horticultura.

2. Ayudar a aficionados a encontrar variedades autóctonas a través de otros miembros de la comunidad.

3. Informar a los usuarios de la existencia de distintas variedades a su alrededor, independientemente de su ubicación, que a menudo son desconocidas por su escasa comercialización.

4. Formar una comunidad de aficionados o entusiastas de estas variedades, facilitando el contacto entre ellos.

5. Dotar de una base de datos de variedades autóctonas de la zona.

6. Contribuir al mantenimiento y protección de variedades que se encuentran cerca de desaparecer por no ser comerciales.

1.3 Estructura del trabajo

El trabajo se encuentra estructurado de la siguiente forma:

1. **Introducción**: En este capítulo se realiza una introducción al proyecto, describiendo la motivación y los objetivos del mismo, sobre qué trata, qué problema se resuelve y un desglose del resto del documento.

2. **Tecnologías utilizadas**: En este apartado se realiza un ligero estudio sobre la plataforma Android y su entorno, describiendo brevemente los recursos utilizados, tanto lenguajes de programación, como APIs, programas, etc.

3. **Entorno geográfico y variedades autóctonas:** En este capítulo se profundizará en las características de la zona en la que nos encontramos, donde se estudiarán diversos aspectos determinantes para las especies como puede ser el clima, el suelo, etc. que nos ayudarán a entender el problema de la desaparición de estas variedades y la aparición de otras nuevas.

4. **Metodología y desarrollo del proyecto**: En este apartado se detallan los procedimientos realizados para llegar finalmente al apartado siguiente, los resultados.

5. **Resultados:** En este capítulo se mostrará el diseño y la realización de la aplicación, ofreciendo una descripción/explicación de las distintas opciones y herramientas de la misma y creando una guía de uso de la aplicación.

6. **Conclusiones y trabajo futuro:** Aquí se detallan las conclusiones finales obtenidas tras la finalización del proyecto y las posibles líneas de trabajo para el futuro.

7. **Anexo:** En este apartado se ampliarán algunas partes del desarrollo del proyecto, profundizando en la instalación y puesta en marcha del entorno.

8. **Referencias bibliográficas.**

2. TECNOLOGÍAS UTILIZADAS

1.1 Introducción

Como el objetivo del presente proyecto es el desarrollo de una aplicación para móviles con sistema operativo Android, con acceso a información almacenada en una base de datos, he utilizado la pareja de lenguajes **JAVA + XML**, que son la base para la aplicación móvil.

Como la aplicación se basa en la localización de árboles, se ha realizado todo el trabajo de mapas y ubicaciones con la **API** de **Google Maps** para Android, una herramienta muy útil. En la parte del modelo de datos, como backend, se ha optado por utilizar **Parse** mediante un gestor, para simplificar la utilización de este BaaS, siendo el elegido **Sashido**, que para mi gusto ha resultado una gran elección tanto por la documentación existente como por el precio tan económico que tiene.

Por último, el entorno de desarrollo utilizado ha sido **Android Studio**, elegido por su multitud de características, su continua actualización, su manejabilidad, sencillez y potencial, además de ser actualmente el único con soporte por parte del grupo de Google.

1.2 Android

Figura 2*: logo de Android*

1.2.1 Definición

Android es un sistema operativo móvil basado en Linux, que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como teléfonos

inteligentes, tabletas, google TV y otros dispositivos. El sistema operativo de Android es de código abierto en su totalidad, lo que significa que cualquiera (incluso la competencia de Android) puede descargar, instalar, modificar y distribuir su código fuente de forma gratuita.

El anuncio del sistema Android se realizó el 5 de noviembre de 2007 junto con la creación de la Open Handset Alliance, un consorcio de 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles. Google liberó la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto.

Android es el nombre de un sistema operativo que se emplea en dispositivos móviles, por lo general con pantalla táctil. De este modo, es posible encontrar tabletas (tablets), teléfonos móviles (celulares) y relojes equipados con Android, aunque el software también se usa en automóviles, televisores y otras máquinas.

Tiene una gran comunidad de desarrolladores desarrollando aplicaciones para extender la funcionalidad y usabilidad de los dispositivos. En el año 2014 ya se había sobrepasado el millón de aplicaciones (de las cuales dos tercios eran gratuitas) disponibles para la tienda de aplicaciones oficial de Android: Play Store, sin tener en cuenta aplicaciones de otras tiendas no oficiales para Android, como pueden ser la App Store de Amazon o la tienda de aplicaciones Samsung Apps de Samsung. Play Store es la tienda de aplicaciones en línea administrada por Google, aunque existe la posibilidad de obtener software externamente, por lo que no se puede asegurar que sea un sistema operativo libre de malware, aunque la mayoría del existente se encuentra en las aplicaciones que se descargan de sitios de terceros. [1]

Android puede adaptarse a múltiples resoluciones de pantalla y soporta conexiones WiFi, Bluetooth, LTE, CDMA, GSM/EDGE, HSPA+ y UMTS, entre otras. También permite el envío de mensajes MMS y SMS, cuenta con navegador web, posibilita el desarrollo de streaming y está capacitado para trabajar con archivos MP3, GIF, JPEG, PNG, BMP, WAV, MIDI, MPEG-4 y otros formatos multimedia. [2]

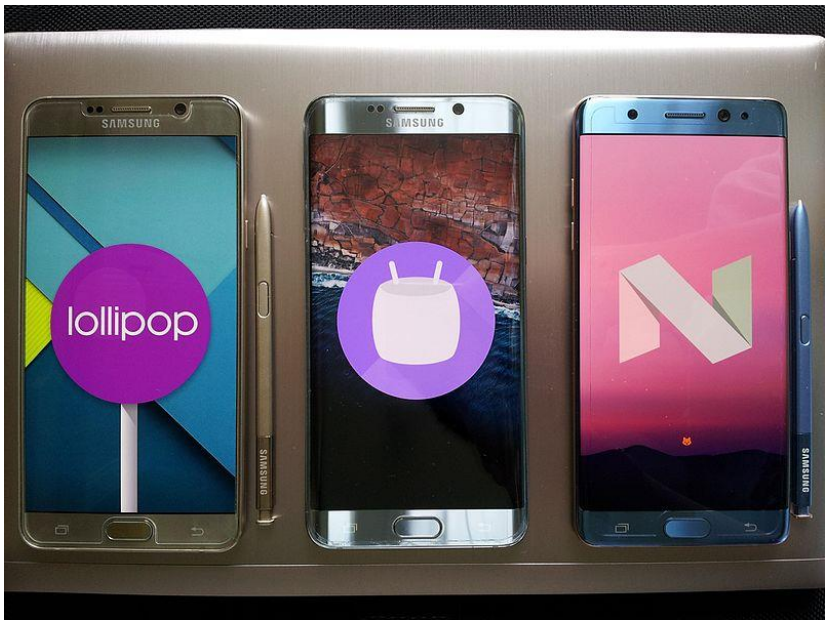

Figura 3: *Teléfonos con Android*

1.2.2 Ventajas

Si miramos diez años atrás en el tiempo, en 2006 solo un 1% de la población mundial tenía un Smartphone, que aunque para la época eran impresionantes, tenían un precio muy elevado. Los fabricantes de dispositivos tenían que pagar las licencias del sistema operativo (SO) o hacerse cargo del coste derivado de desarrollar el suyo propio desde cero. Además, los desarrolladores y fabricantes de terceros no tenían acceso a la mayoría de sistemas operativos. En otras palabras, los smartphones no eran para todos.

En consecuencia, Google se alió en 2007 con el sector de los dispositivos móviles para crear la alianza empresarial Open Handset Alliance (OHA). Su misión era establecer Android como un SO de código abierto. Esto significaba que todo el mundo podría acceder al código fuente de Android y descargarlo y modificarlo de forma gratuita para poder desarrollar aplicaciones, dispositivos móviles o incluso un SO competidor.

Las ventajas fueron inmediatas. Las empresas pudieron personalizar Android para desarrollar experiencias únicas para sus clientes, los desarrolladores de aplicaciones pudieron acceder a una audiencia global y, quizás lo más impactante, los fabricantes de dispositivos pudieron instalar Android gratis sin pagar licencias ni desarrollar su propio SO.

Gracias a Android, hemos reducido la inversión en desarrollo en un 30% y el ciclo de desarrollo en un 25%, permitiendo a los fabricantes reducir los costes de desarrollo, lo que ayudó a

rebajar el precio de los smartphones a nivel mundial. Entre 2011 y 2013, el precio medio de los smartphones cayó un 25% en todo el mundo.

Mientras esta tendencia se mantenga, estaremos más cerca de conseguir que cualquier persona que quiera aprender, crecer y tener éxito pueda acceder a información de cualquier parte del mundo gracias a la tecnología móvil asequible.

Precios más competitivos en todo el mundo

Al proporcionar un SO abierto y gratuito, Android ha ayudado directamente a que haya más dispositivos móviles asequibles en todo el mundo. En 2015, el coste medio de un dispositivo Android era de 208 USD. Esta cifra contrasta enormemente con el coste de los teléfonos en plataformas cerradas, que pueden llegar a tener un precio medio de 651 USD. Google se ha asociado con fabricantes locales de África y la India para proporcionar smartphones de alta calidad por menos de 100 USD, como el dispositivo Fero i401, que se ha puesto a la venta en Nigeria por menos de 60 USD.

Figura 4: *Teléfono móvil Fero i401*

En muchos mercados, como el filipino, los fabricantes locales han presentado smartphones basados en Android con un precio inferior a 50 USD.

Gracias a dispositivos como este, usuarios de todo el planeta tienen acceso a nuevas oportunidades, como crear una empresa, compartir un vídeo con el mundo o recibir educación para ellos o sus hijos. Nada de esto sería posible si todos los smartphones costaran 600 dólares.

Por este motivo, la mayoría de smartphones de África y la India utilizan Android. No es porque no haya más alternativas ni porque la competencia haya obligado a ello, sino porque Android es la opción más accesible económicamente para gran parte de la población mundial.

La conclusión es que todos tienen derecho a las mismas oportunidades. Los usuarios de Lagos (Nigeria) merecen tener el mismo acceso a la información que los de Los Ángeles (EEUU). Dentro de poco habrá un smartphone (es decir, una puerta a la información mundial) que se podrá permitir todo aquel que tenga 50 USD y un ápice de curiosidad.

Más aplicaciones

Las ventajas sin precedentes que ofrece Android no solo se limitan a los dispositivos. Solo en Google Play Store hay más de 1 millón de aplicaciones disponibles y esa variada gama de aplicaciones abre un nuevo mundo de experiencias.

Figura 5: *Play Store*

Pongamos que quieres aprender un idioma. Play Store ofrece más de 270 aplicaciones para ello y más del 70% se descargan gratuitamente. Hay aplicaciones para comprar y vender, leer y escribir, encontrar trabajo, contratar empleados e, incluso, tocar el piano como un virtuoso. En Android puedes hacer prácticamente cualquier cosa y a un precio muy asequible.

Democratización del acceso a la información en todo el mundo

Los smartphones no son solo teléfonos, permiten hacer nuevos descubrimientos, conseguir objetivos ambiciosos y potenciar el conocimiento. Gracias a los dispositivos móviles, toda una generación puede acceder a Internet por primera vez en zonas de todo el mundo en las que esta acción no es tan habitual. Los smartphones ya no son un símbolo de estatus porque cada vez más gente tiene acceso a ellos.

Este es el motivo por el que Android se ha convertido en una plataforma que cualquier fabricante de dispositivos o desarrollador puede utilizar y en la que se puede ejecutar cualquier aplicación. También es la razón por la que Android se ha comprometido a seguir siendo una plataforma abierta, transparente y accesible para desarrollares y usuarios de todo el mundo.

En muchas zonas, el móvil es la única manera de acceder a Internet. El 95% de la población mundial tiene acceso a una red móvil. Sin embargo, sólo el 10,8% de los hogares de todo el mundo cuenta con una suscripción fija a banda ancha. Como dijo Sir Tim Berners-Lee (padre de la Web) sobre Internet: "Es para todos".

Lo mismo ocurre con Android.

1.2.3 Versiones

En cuanto a las distintas versiones de Android, cabe mencionar que se denominan con nombres de postres, cuyas iniciales se ordenan alfabéticamente. Así, la primera versión de Android se llamó Apple Pie, la segunda Banana Bread y así sucesivamente. Esto permite reconocer las versiones y determinar cuáles son las más recientes de acuerdo a su letra inicial.

Las versiones lanzadas al mercado (a partir de la tercera) son las siguientes:

- **Android Cupcake 1.5**: el sistema operativo que desde el 2009 llegó con los nombres de dulces o postres que Google le coloca a sus sistemas para Android. Este trae una amplia gama de despliegue en su plataforma donde los usuarios y desarrolladores encuentran desde las aplicaciones antiguas a las características más nuevas, y facilita un teclado en la propia pantalla de inicio, junto con una interfaz Bluetooth ampliando la gestión en las aplicaciones junto con las que Google a través del tiempo sigue aportando.

- **Android Donut 1.6**: rosquillas o buñuelos son los nombres que se le dio a este nuevo sistema operativo de Google, donde con código abierto salía al mercado en el año 2009 la nueva versión, la cual se creó para los llamados teléfonos inteligentes o de nueva generación, este tenía un buen soporte en CDMA, corriendo en los celulares con distintas pantallas y variedad de tamaños, innovando con la búsqueda y mensajería por voz, generaba indicación de la vida de uso de la batería del celular.

- **Android Eclair 2.0/2.1**: sistema operativo donde Google desarrolló una experiencia diferente al usuario, teniendo una barra constante que se podía localizar en la parte superior de la pantalla del móvil, aportando la facilidad y la accesibilidad al usuario de entrar al buscador con un solo clic, la cámara llega con nuevas funciones donde el flash, el zoom digital, las diferentes escenas de tomas de la foto, los balances y los efectos de colores fueron la estrella del sistema, con ayuda para los inexpertos en fotografías como el marco de enfoque y edición de las fotografías, los fondos de pantallas aparecieron en vivos colores y con movimientos, salió al mercado en el 2009.

- **Android Froyo 2.2.x**: versión de sistema operativo llamado yogur helado, de Android que Google desarrolló donde las mejoras en los puertos USB y WI-Fi fueron liberadas, dando mejor apoyo en sus usos, aumentando la velocidad, la mensajería y almacenaje de información en la nube de Android a través de Google, una versión que se podía ejecutar desde Google Play, teniendo más memoria, mejor rendimiento, velocidad, funcionalidad en el Bluetooth, la galerías de imágenes y el zoom fueron algunas mejoras desde que salió en el 2010, sus versión 2.2 y 2.2.3 fueron sus secuelas.

- **Android Gingerbread 2.3.x**: siguiendo con su forma de nombrar a sus sistemas operativos en nombres de postres nos llega al mercado el pan de jengibre, que Google sacó al mercado junto con el teléfono Nexus S en el 2010, manteniendo las características de sus versión más antigua que va desde su buen navegador visual de páginas, el calendario sincronizado de google y el muy famoso Google Maps entre otras, llega con nueva resolución de imagen de pantallas extra grandes, mejorando el copia y pega sobre textos seleccionados, nuevos efectos de sonidos mejorando su

ecualizador, el auricular y los bajos, dando una buena calidad al audio, los gráficos y los juegos.

- **Android HoneyComb 3.x**: panal en español, caracterizado en tener variedad de idiomas y creado solo para ser usado en tablets, como la que usaron para su lanzamiento la tablets Motorola Xoom, esta aplicación tiene en su haber dos actualizaciones más aparte de su original la Android 3.0, cuenta con Android 3.1 y Android 3.2.

- **Android Ice Cream Sandwich 4.0.x**: sándwich helado es su nombre en español, es un sistema operativo que aparece en el 2011, que a diferencia de sus pasadas versiones se creó para que se pueda utilizar tanto en dispositivos móviles como en sus tablets, creado por Google para facilitar y abarcar un mercado más amplio y competitivo de accesibilidad a la información y el manejo de las nuevas gamas de la tecnología, tratando de crear una plataforma unificada entre dispositivos inteligentes.

- **Android Jelly Bean 4.1/4.2/4.3**: nombre que significa gomitas, un sistema operativo el cual ha dejado las características de versiones pasadas como Google Maps y más, aparece en esta nueva versión con Bluetooth con bajo consumo de energía y ampliando la variedad de idiomas como el Hebreo, Árabe, Africano, Hindú, Suajili, Zulú entre otros; utilizando la interfaz de WI-FI pero con la seguridad en los datos sobre los perfiles de usos personales, con acceso restringidos. Tiene una gama limpia en la escritura con diferentes tamaños, con manejo de la tecla de marcación de rápida o la llamada Dial Pad, un asistente personal sobre la información guardada y el poder ver contenido de TV por internet, es de mucha utilidad para los desarrolladores de app mejorando las codificaciones sobre los videos.

- **Android KitKat 4.4**: es una interfaz de código abierto donde con un convenio con la empresa Nestlé, para que pudiera llevar el nombre de uno de sus productos estrella como lo es el KitKat, mejorando el diseño, la funcionalidad del sistema, el rendimiento en tan solo 512 MB de memoria RAM sin alterar el consumo de la batería, es un sistema que se ha mejorado en nuevas versiones que son 4.4.1, 4.4.2, 4.4.3 y 4.4.4.

- **Android Lollipop 5.0**: piruleta o paleta, nombres en español con la cual se identifica dicho sistema operativo, es usado en móviles Android, con la finalidad de incluir una nueva interfaz llamada Material Design, funcionando en cualquier plataforma de Google dando una consistencia entre las diferentes aplicaciones y plataformas creadas, dando una mejoría a la notificaciones accediendo desde la pantalla de bloqueo, otra mejora es optimizar el uso y consumo de la batería dando un mayor rendimiento de un 90 % de uso diario.

- **Android Marshmallow 6.0**: Conocido como malvavisco, un sistema operativo creado para los dispositivos móviles Android donde su principal característica es no ceder todo el permiso de uso a las aplicaciones, utilizando un opt-in como sistema regulador donde el usuario al bajar una app decide que permite y que no, poniendo o quitando accesibilidades en el permiso de uso como compartir información de ubicación, fotografías o la cámara, videos o sonido entre otros, esta aplicación recuerda mediante avisos qué debe permitir y qué no. [3]

- **Android Nougat 7.0:** Su nombre en español significa Turrón. Se libera la OTA el 22 de Agosto del 2016 de momento para los Nexus 6, 5x, 6P, 9, Nexus Player, Pixel C, Huawei P8 Lite 2017 y Android One. Incluye mejoras en las animaciones, opción multiventana de forma nativa y un nuevo centro de notificaciones entre otras cosas.

1.3 Android Studio

Figura 6: *Logo Android Studio*

1.3.1 Definición

Android Studio es un entorno de desarrollo integrado (IDE), basado en IntelliJ IDEA de la compañía JetBrains, que proporciona varias mejoras con respecto al plugin ADT (Android Developer Tools) desarrollado para Eclipse. Android Studio utiliza una licencia de software libre Apache 2.0, está programado en Java y es multiplataforma.

Fue presentado por Google el 16 de mayo del 2013 en el congreso de desarrolladores Google I/O, con el objetivo de crear un entorno dedicado en exclusiva a la programación de aplicaciones para dispositivos Android, proporcionando a Google un mayor control sobre el proceso de producción. Se trata pues de una alternativa real (y para mi gusto, superior) a Eclipse, que presentaba problemas debido a su lentitud en el desarrollo de versiones que solucionaran las carencias actuales (es indispensable recordar que Eclipse es una plataforma de desarrollo, diseñada para ser extendida a través de plugins).

Android Studio se ha mantenido durante todo este tiempo en versión beta, pero desde el 8 de diciembre de 2014, en que se liberó la versión estable de Android Studio 1.0, Google ha pasado a recomendarlo como el IDE para desarrollar aplicaciones para su sistema operativo, dejando el plugin ADT para Eclipse de estar en desarrollo activo. [4]

1.3.2 Características

Principales características que incluye Android Studio:

- Soporte para programar aplicaciones para Android Wear (sistema operativo para dispositivos corporales como por ejemplo un reloj).
- Compatibilidad con C++ y NDK
- Herramientas Lint (detecta código no compatible entre arquitecturas diferentes o código confuso que no es capaz de controlar el compilador) para detectar problemas de rendimiento, usabilidad y compatibilidad de versiones.
- Utiliza ProGuard para optimizar y reducir el código del proyecto al exportar a APK (muy útil para dispositivos de gama baja con limitaciones de memoria interna).
- Integración de la herramienta Gradle encargada de gestionar y automatizar la construcción de proyectos, como pueden ser las tareas de testing, compilación o empaquetado.
- Nuevo diseño del editor, con soporte para la edición de temas.
- Nueva interfaz específica para el desarrollo en Android.

- Permite la importación de proyectos realizados en el entorno Eclipse, que a diferencia de Android Studio (Gradle) utiliza ANT.
- Posibilita el control de versiones accediendo a un repositorio desde el que poder descargar Mercurial, Git, Github o Subversion.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento antes de compilar la aplicación.
- Vista previa en diferentes dispositivos y resoluciones.
- Integración con Google Cloud Platform, para el acceso a los diferentes servicios que proporciona Google en la nube.
- Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo xml. [5]

1.3.3 Ventajas

Las ventajas de uso de Android Studio son las siguientes:

- Android Studio ha pasado a ser el entorno recomendado para el desarrollo de aplicaciones en Android, al tratarse de un IDE oficial de Google en colaboración con JetBrains (compañía de desarrollo software especializada en diseño de IDEs).
- Android Studio permite la creación de nuevos módulos dentro de un mismo proyecto, sin necesidad de estar cambiando de espacio de trabajo para el manejo de proyectos, algo habitual en Eclipse.
- Con la simple descarga de Android Studio se disponen de todas las herramientas necesarias para el desarrollo de aplicaciones para la plataforma Android.
- Su nueva forma de construir los paquetes .apk, mediante el uso de Gradle, proporciona una serie de ventajas más acorde a un proyecto Java:
    - Facilita la distribución de código, y por lo tanto el trabajo en equipo.
    - Reutilización de código y recursos.
    - Permite compilar desde línea de comandos, para aquellas situaciones en las que no esté disponible un entorno de desarrollo.
    - Mayor facilidad para la creación de diferentes versiones de la misma aplicación, que proporciona numerosas ventajas como puede ser la creación de una versión de pago y otra gratuita, o por ejemplo diferentes dispositivos o almacén de datos.

1.3.4 Desventajas

Las desventajas de uso de Android Studio son las siguientes:

- Aunque ya ha se han lanzado versiones estables, la v1.0, al estar en una fase inicial, siempre es susceptible de introducirse más cambios que puedan provocar inestabilidad entre proyectos de diferentes versiones.
- Curva de aprendizaje más lenta para nuevos desarrolladores de Android.
- El sistema de construcción de proyectos Gradle puede resultar complicado inicialmente.
- En comparativa con Eclipse, menor número de plugins. [4]

1.3.5 Comparativa entre Android Studio y ADT Eclipse

Para una mayor comprensión de las diferencias y novedades que presenta Android Studio con respecto al IDE Eclipse, y más concretamente con el ADT para Android, se propone la siguiente tabla comparativa entre ambas opciones:

Características	Android Studio	ADT
Sistema de construcción	Gradle	ANT
Construcción y gestión de proyectos basado en Maven (herramienta de software para la gestión y construcción de proyectos Java, similar a Apache ANT, pero su modelo es más simple ya que está basado en XML)	Si	No (es necesario instalar un plugin auxiliar)
Construir variantes y generación de múltiples APK (muy útil para Android Wear)	Si	No
Refactorización y completado avanzado de código Android	Si	No
Diseño del editor gráfico	Si	Si

Firma APK y gestión de almacén de claves	Si	Si
Soporte para NDK (Native Development Kit: herramientas para implementar código nativo escrito en C y C++)	Próximas versiones	Si
Soporte para Google Cloud Platform	Si	No
Vista en tiempo real de renderizado de layouts	Si	No
Nuevos módulos en proyecto	Si	No
Editor de navegación	Si	No
Generador de assets	Si	No
Datos de ejemplo en diseño de layout (sin renderizar en tiempo de ejecución)	Si	No
Visualización de recursos desde editor de código	Si (a la izquierda de la línea de asignación del recurso)	No

1.4 Java

Figura 7: *Logo Java*

1.4.1 Definición

Sun Microsystems desarrolló, en 1991, el lenguaje de programación orientado a objetos que se conoce como Java. El objetivo era utilizarlo en un set-top box, un tipo de dispositivo que se encarga de la recepción y la descodificación de la señal televisiva. El primer nombre del lenguaje fue Oak, luego se conoció como Green y finamente adoptó la denominación de Java.

Figura 8: ***Logo Sun microsystems***

La intención de Sun era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel.

Los pilares en los que se sustenta Java son cinco: la programación orientada a objetos, la posibilidad de ejecutar un mismo programa en diversos sistemas operativos, la inclusión por defecto de soporte para trabajo en red, la opción de ejecutar el código en sistemas remotos de manera segura y la facilidad de uso.

Lo habitual es que las aplicaciones Java se encuentren compiladas en un bytecode (un fichero binario que tiene un programa ejecutable), aunque también pueden estar compiladas en código máquina nativo.

Sun por su parte, controla las especificaciones y el desarrollo del lenguaje, los compiladores, las máquinas virtuales y las bibliotecas de clases a través del Java Community Process. En los últimos años, esta empresa (que fue adquirida por Oracle) ha liberado gran parte de las tecnologías Java bajo la licencia GNU GPL.

La aplicación de Java es muy amplia. El lenguaje se utiliza en una gran variedad de dispositivos móviles, como teléfonos y pequeños electrodomésticos. Dentro del ámbito de Internet, Java permite desarrollar pequeñas aplicaciones (conocidas con el nombre de applets) que se incrustan en el código HTML de una página, para su directa ejecución desde un navegador, donde cabe mencionar que es necesario contar con el plug-in adecuado para su funcionamiento, pero la instalación es liviana y sencilla. [6]

1.4.2 Características

Sun describe al lenguaje Java de la siguiente manera:

- Simple
- Orientado a Objetos
- Tipado estáticamente
- Distribuido
- Interpretado
- Robusto
- Seguro
- De arquitectura neutral
- Multihilo
- Con recolector de basura (Garbage Collector)
- Portable
- De alto rendimiento: sobre todo con la aparición de hardware especializado y mejor software
- Dinámico

Aunque Sun admite que lo dicho anteriormente son un montón de halagos por su parte, el hecho es que todas esas características pueden servir para describir este lenguaje. Todas ellas son importantes, sin embargo cabe destacar tres, que son las que han proporcionado tanto interés por el lenguaje: la portabilidad, el hecho de que sea de arquitectura neutral y su simplicidad. Java ofrece toda la funcionalidad de los lenguajes potentes, pero sin las características menos usadas y más confusas de éstos.

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas especificaciones del lenguaje y añadir características muy útiles como el recolector de basura. No es necesario preocuparse de liberar memoria, el recolector se encarga de eliminar la memoria asignada. Gracias al recolector, sólo te tienes que preocupar de crear los objetos relevantes de tu sistema ya que él se encarga de destruirlos en caso de no ser reutilizados.

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++. Entre las características más "indeseables" de C++ que se han evitado en el diseño de Java

destacan: ficheros de cabecera, aritmética de punteros, sobrecarga de operadores, estructuras, uniones, conversión implícita de tipos, clases base virtuales, pre-procesador, etc. [7]

1.4.3 Ventajas

Las ventajas de programar en Java son las siguientes:

1. Lenguaje independiente de la plataforma o multiplataforma: El código que se escribe en java es leído por un intérprete, por lo que su programa funcionará en cualquier plataforma.

2. Manejo automático de la memoria (para los que vienen de C/C++), ya que se hace automáticamente y utilizando el "garbage collector".

3. Es gratuito.

4. Permite el desarrollo de aplicaciones web dinámicas que, mediante XML, ofrecen un diseño mucho más atractivo que una página estática. Además permite incluir sonido y objetos multimedia así como bases de datos y otras funcionalidades.

5. Permite desarrollar aplicaciones de servidor para foros en línea, almacenes, encuestas, procesamiento de formularios HTML y mucho más.

1.4.4 Inconvenientes

Algunas desventajas de utilizar Java son las siguientes:

1. Lentitud a la hora de ejecutar las aplicaciones (aunque ha mejorado mucho con el tiempo).

2. La portabilidad no existe sin la máquina virtual (JVM).

3. Requiere un intérprete. El browser tiene que interpretar los ficheros de las clases antes de que se ejecuten. Utilizando un lenguaje de programación tradicional como puede ser C++, el ordenador puede ejecutar directamente el código generado. Sin embargo, debido a la interpretación que el browser tiene que hacer de los ficheros, los

programas escritos en Java tienden a ejecutarse bastante más lentos que con otros lenguajes de programación (p.e. C++).

4. Algunas implementaciones y librerías pueden tener código rebuscado.

5. Una mala implementación de un programa en java puede resultar en algo muy lento.

6. Algunas herramientas tienen un coste adicional.

1.5 XML

Figura 9: *Logo XML*

1.5.1 Definición

XML proviene de eXtensible Markup Language (“Lenguaje de Marcas Extensible”). Se trata de un metalenguaje, utilizado para decir algo acerca de otro, extensible, de etiquetas que fue desarrollado por el Word Wide Web Consortium (W3C), que es una sociedad mercantil internacional que elabora recomendaciones para la World Wide Web.

Figura 10: *Logo World Wide Web Consortium*

XML es una adaptación del SGML (Standard Generalized Markup Language), un lenguaje que permite la organización y el etiquetado de documentos. Esto quiere decir que XML no es un

lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo a las necesidades. XHTML, MathML y SVG son algunos de los lenguajes que XML tiene la capacidad de definir.

Las bases de datos, los documentos de texto, las hojas de cálculo y las páginas web son algunos de los campos de aplicación de XML. El metalenguaje aparece como un estándar que estructura el intercambio de información entre las diferentes plataformas.

Entre los lenguajes creados con XML, destacan XSL (Extensible Stylesheet Language) y XLINK (que intenta trascender las limitaciones de los enlaces de hipertexto en HTML).

La validez de los documentos (es decir, que su estructura sintáctica se encuentre desarrollada correctamente) depende de la relación especificada entre los distintos elementos a partir de una definición o documento externo. [8]

1.5.2 Características

Sus características más relevantes son:

1. XML es un estándar para escribir datos estructurados en un fichero de texto. XML provee un conjunto de reglas, normas y convenciones para diseñar formatos de texto para datos estructurados que van desde las hojas de cálculo, o las libretas de direcciones de Internet, hasta parámetros de configuración, transacciones financieras o dibujos técnicos.

   Los programas que los generan, utilizan normalmente formatos binarios o de texto. XML permite resolver problemas comunes, como la falta de extensibilidad, carencias de soporte debido a características de internacionalización o problemas asociados a plataformas específicas.

2. XML parece HTML pero no lo es. Tanto XML como HTML usan marcas y atributos, pero su diferencia radica en que en HTML cada marca y atributo establece un significado a la vez que incluye el aspecto que debe tener cuando se ve en un navegador, mientras que en XML sólo se usan las marcas para delimitar fragmentos de datos, dejando la interpretación de éstos a la aplicación que los lee.

3. XML está en formato texto, pero no para ser leído. El formato texto puede ser usado en cualquier plataforma, esto le da innumerables ventajas de portabilidad, depuración,

independencia de plataforma e incluso de edición, pero su sintaxis es más estricta que la de HTML: una marca olvidada o un valor de atributo sin comillas convierten el documento en inutilizable. No hay permisividad en la construcción de documentos, ya que esa es la única forma de protegerse contra problemas más graves.

4. XML consta de una familia de tecnologías. La definición (estándar) de XML 1.0 viene de Febrero de 1998 pero su desarrollo se ha ido enriqueciendo paulatinamente a medida que se veían sus posibilidades: de esa forma, contamos con una especificación Xlink, que describe un modo estándar de añadir hipervínculos a un documento XML. XPointer y XFragments son especificaciones para establecer la forma de vincular partes de un documento XML.

   Incluso el lenguaje de hojas de estilo (CSS) se puede utilizar con XML al igual que se hace con HTML. XSL es precisamente, una extensión del anterior, en la que se dispone de todo un lenguaje de programación exclusivamente para definir criterios de selección de los datos almacenados en un documento XML, y que funciona conjuntamente con las CSS o con HTML para suministrar al programador y al usuario mecanismos de presentación y selección de información, que no requieran de la intervención constante del servidor. Se basa en un lenguaje anterior para transformación (XSLT) que permite modificar atributos y marcas de forma dinámica.

5. Los ficheros resultantes son casi siempre mayores que sus equivalentes binarios. Esto es intencionado, y las ventajas ya las hemos comentado más arriba, mientras que las desventajas, siempre pueden ser soslayadas mediante técnicas de programación que permite comprimir los datos.

6. XML es nuevo, pero no tanto. El estándar empezó a diseñarse en 1996, y se publicó la recomendación en Febrero de 1998. Como ya hemos comentado, eso no significa que la tecnología no esté suficientemente madura, ya que el estándar SGML en el que se basa, data de una especificación ISO del año 1986.

7. XML no requiere licencia. Es un estándar abierto independiente de la plataforma y tiene un amplio soporte extendido a un sinnúmero de herramientas y desarrolladores. [9]

1.5.3 Ventajas

Las ventajas de XML son las siguientes:

1. Se trata de un lenguaje fácilmente procesable tanto por humanos como por software.

2. Separa radicalmente la información o el contenido de su presentación o formato.

3. Está diseñado para ser utilizado en cualquier lenguaje o alfabeto.

4. Su análisis sintáctico es fácil debido a las estrictas reglas que rigen la composición de un documento.

5. Estructura Jerárquica.

6. El número de marcas es ilimitado.

1.5.4 Inconvenientes

La posibilidad de construir sistemas acordes a nuestras necesidades para el intercambio de datos podría llevarnos a la proliferación de versiones incompatibles y si esto llegase a suceder, entonces la solución que plantea XML ante la búsqueda de intercambio universal de información lo llevaría a su opuesto; en vez de unificar todo un lenguaje, nos encontraríamos con lenguajes muy específicos y cada vez más alejados de la "universalidad". [10]

1.6 Parse

Figura 11: *Logo Parse*

1.6.1 Definición

Parse es lo que se denomina un Backend as a Service (BaaS) que nos va a hacer disfrutar de múltiples funcionalidades en la nube, permitiéndonos el desarrollo de aplicaciones que requieran un backend con muy poco esfuerzo.

Como buen Backend, Parse nos provee de herramientas dentro de un servidor web, para poder implementar en nuestras aplicaciones unas determinadas funcionalidades.

Los servicios que ofrece Parse principalmente son:

- Modelo de datos en la nube: creación de tablas no-SQL en la nube y capacidades para inserción, modificación y consulta vía API.
- Notificaciones Push: posibilidad de envío de notificaciones push a nuestros usuarios, previa aceptación por parte del usuario. En este apartado Parse destaca notablemente, pues de otra manera, sin usar Parse, el envío de notificaciones a los usuarios de nuestras Apps se convierte en una tortura, por ejemplo, en Apple, que controla mucho la seguridad, para enviar notificaciones desde tu propio servidor tienes que tener autenticación segura, establecer un protocolo cifrado con los servidores de Apple, etc. Con Parse todo es mucho más sencillo, sólo debes subir a su web unos certificados que previamente has de crear y listo.
- Cloud Code: capacidades para la ejecución de código en el servidor, muy útil para la realización de validaciones de seguridad, o procesos automáticos por cambios en los datos.
- Analytics: permite analizar el uso que los usuarios le dan a tu App.

Así mismo, Parse nos ofrece SDK para utilizar el servicio en múltiples plataformas como son: iOS, OSX, Android, Javascript, Windows Phone 8, Windows 8, .NET, o de forma estándar vía REST API. [11]

En definitiva, podríamos hacer lo mismo sin Parse, pero realmente sería mucho más lento y costoso hacerlo por nuestra cuenta en nuestro propio servidor.

1.7 Sashido

Figura 12: *Logo Sashido*

1.7.1 Definición

Sashido es un MBaaS (mobile backend as a service), concepto simple que nos abastrae de la mayoría de procesos complejos, permitiendo al usuario desconocer gran parte de la técnica, para que se pueda centrar en la funcionalidad de su aplicación, dejando a terceros al cuidado de la infraestructura, el escalado, la seguridad, la base de datos, etc.

Sashido es una herramienta que nos proporciona el panel de control (o dashboard) desde el cual podemos crear la base de datos, controlar las estadísticas de nuestra aplicación, etc.

Figura 13: *Cabecera dashboard Sashido*

Sashido fue fundado en 2016 y desde entonces ha crecido de una forma espectacular, como podemos ver en la siguiente ilustración:

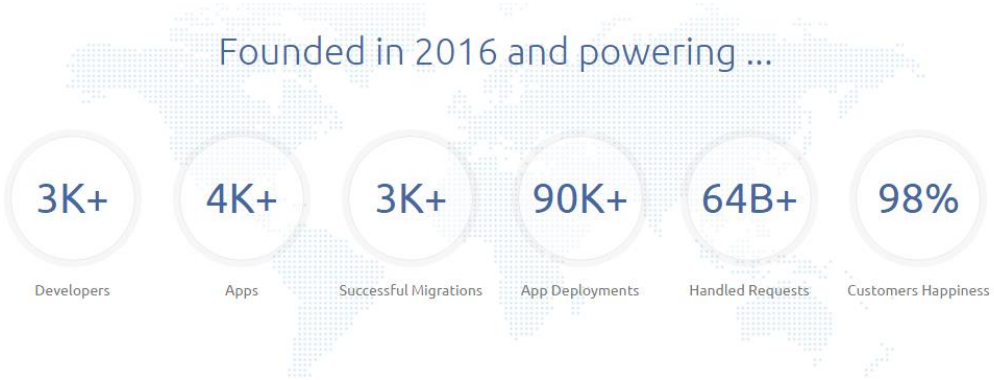

Figura 14: *Estadísticas Sashido*

1.7.2 Ventajas

El motivo por el cual se utiliza es simple: proporciona una forma simple de gestionar y manejar tu propio servidor Parse, que puede ser bastante complejo y suponer bastante tiempo, por lo que Sashido proporciona un servicio muy útil a un precio muy competitivo (unos 5$ al mes más impuestos), además de tener soporte en cualquier momento que se necesite y de contar con sistemas de escalado cuando aumenta el número de peticiones.

Figura 15: *Ventajas de Sashido*

1.8 Sinch

Figura 16: *Logo Sinch!*

1.8.1 Definición

Sinch es una plataforma de comunicaciones móviles basada en la nube que permite incorporar voz, video y verificación en aplicaciones de la manera más sencilla posible. Originalmente formada a partir de Rebtel en mayo de 2014, el equipo de Sinch tiene más de 10 años de experiencia de desarrollo en la industria de voz y aporta esta experiencia y conocimiento a su poderoso abanico de SDK y API para que descarguen e integren los desarrolladores.

En un corto periodo de tiempo, Sinch ha conseguido como clientes a empresas tan importantes como Tango, Truecaller, Easy Taxi, Glide, Nimbuzz, y muchos más, como podemos ver en la siguiente ilustración.

En diciembre de 2016, Sinch fue adquirido por CLX Communications, que cotiza en la bolsa de Estocolmo (NASDAQ) y dispone de más de $250M de ingresos anuales y más de 1000 clientes de distintas empresas. Los servicios de comunicación móviles de CLX permiten a las empresas comunicarse de forma rápida, fácil y rentable con clientes y dispositivos a escala mundial - Internet de las Cosas (IoT).

Juntos, CLX y Sinch están firmemente comprometidos en el suministro de la comunicación sin fisuras en la nube y están seguros de que esta filosofía compartida permitirá a Sinch llevar la "plataforma de comunicaciones como un servicio" (CPaaS) a un nuevo nivel, proporcionando la calidad de voz, vídeo y SMS más alta, y a la verificación en una escala global.

Los productos y servicios de Sinch serán aún mejores bajo el paraguas de CLX. Además del servicio de voz con calidad HD y VoIP ofrecidos, ahora proveen también servicios de SMS a precios competitivos, siempre con una misión, hacerlo fácil para los desarrolladores y así enriquecer las aplicaciones móviles.

1.9 API (Application Programming Interface) de Google Maps

Figura 17*: imagen API*

1.9.1 ¿Qué es una API?

Una **API** es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas: sirviendo de interfaz entre programas diferentes de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

Las API pueden servir para comunicarse con el sistema operativo (WinAPI), con bases de datos (DBMS) o con protocolos de comunicaciones (Jabber/XMPP). En los últimos años, por supuesto, se han sumado múltiples redes sociales (Twitter, Facebook, Youtube, Flickr, LinkedIn, etc) y otras plataformas online (Google Maps, WordPress, etc), lo que ha convertido el social media marketing en algo más sencillo, más rastreable y, por tanto, más rentable.

Las API son valiosas, ante todo, porque permiten hacer uso de funciones ya existentes en otro software (o de la infraestructura ya existente en otras plataformas) para no estar reinventando la rueda constantemente, reutilizando así código que se sabe que está probado y que funciona correctamente. En el caso de herramientas propietarias (es decir, que no sean de código abierto), son un modo de hacer saber a los programadores de otras aplicaciones cómo incorporar una funcionalidad concreta sin por ello tener que proporcionar información acerca de cómo se realiza internamente el proceso. Este último es el caso de la API de Google Maps que he utilizado para el desarrollo de la Plataforma Interactiva. [12]

1.9.2 Google Maps

Figura 18*: Logo Google Maps*

La **API de Google Maps** es a su vez un conjunto de APIs que te permiten superponer tus propios datos sobre un mapa de Google Maps personalizado. Puedes crear atractivas aplicaciones web y móviles con la potente plataforma de mapas de Google, incluso con

imágenes de satélite, Street View, perfiles de elevación, indicaciones sobre cómo llegar, mapas con estilos, demografía, análisis y una amplia base de datos de ubicaciones. Con la cobertura global más precisa del mundo y una comunidad de mapas activa que incorpora actualizaciones diarias, los usuarios se beneficiarán de un servicio que mejora constantemente.

Google Maps fue desarrollado originalmente por dos hermanos Daneses, Lars y Jens Rasmussen, co-fundadores de Where 2 Technologies una empresa dedicada a la creación de soluciones de mapeo. La empresa fue adquirida por Google en octubre de 2004, y los dos hermanos luego crearon Google Maps.

Se trata del servicio de mapas más usado del mundo. Más de 800 000 sitios utilizan la API de Google Maps y hay más de 250 millones de usuarios activos contando solamente los dispositivos móviles. Con la API de Google Maps, puede proporcionar la misma experiencia conocida a sus usuarios. [13]

1.9.3 Función de Google Maps

Con la API de Google Maps Android puedes agregar mapas de Google Maps a tu aplicación. La API administra de forma automática el acceso a los servidores, las descargas de datos, la visualización de mapas y la respuesta a gestos de mapas de Google Maps. También puedes usar llamadas a la API para agregar marcadores, polígonos y superposiciones a un mapa básico, además de poder cambiar la vista del usuario de modo que se muestre un área del mapa en particular. Estos objetos proporcionan información adicional de ubicaciones en el mapa y permiten la interacción del usuario con este. La API te permite agregar los siguientes gráficos a un mapa:

- Iconos anclados en posiciones específicas del mapa (marcadores).
- Conjuntos de segmentos de líneas (polilíneas).
- Segmentos cerrados (polígonos).
- Gráficos de mapa de bits anclados en posiciones específicas del mapa (marcadores).
- Conjuntos de imágenes que se muestran sobre los mosaicos de mapas básicos (superposiciones de mosaicos).

El API consiste en archivos JavaScript que contienen las clases, métodos y propiedades que se usan para el comportamiento de los mapas. [14]

1.9.4 Coordenadas

Las coordenadas están expresadas usando números decimales separados por coma. La latitud siempre precede a la longitud. La latitud es positiva si va después del punto mostrado en el mapa y negativo si va antes. La longitud es positiva si va arriba del punto y negativa si va debajo.

En los mapas físicos, las coordenadas están expresadas en grados, así que la posición de Puerto Rico sería:

18°14'70" N 66°29'68" W

La forma de convertir estos datos a decimales sería:

(18°14'70" N) = (18 + (14 / 60) + (70 / 3600)) = 18.252

(66°29'68" W) = -(66 + (29 / 60) + (68 / 3600)) = -66.8627

La longitud se multiplica por negativo, porque está a la izquierda (oeste) del punto 0,0. [15]

1.9.5 Tipos de mapas

Existen muchos tipos de mapas disponibles en la Google Maps Android API. El tipo de un mapa determina la representación general de este. Por ejemplo, un atlas generalmente contiene mapas políticos que se centran en la visualización de fronteras. En los mapas de carreteras se muestran todas las carreteras de una ciudad o región.

La API ofrece cuatro tipos de mapas y una opción para que no se muestre ninguno:

- **Normal**. Mapa de carreteras típico. Muestra carreteras, algunas características creadas por el hombre y características naturales importantes, como ríos. También se ven etiquetas de carreteras y elementos.

- **Híbrido**. Datos de fotos satelitales con mapas de carreteras agregados. También se ven etiquetas de carreteras y elementos.

- **Satélite**. Datos de fotos satelitales. No se ven etiquetas de carreteras y elementos.

- **Tierra**. Datos topográficos. En el mapa se incluyen colores, líneas de contornos y etiquetas, y sombreados de perspectiva. También se pueden ver carreteras y etiquetas.

- **Ninguno**. Ausencia de mosaicos. El mapa se representa como una cuadrícula vacía sin mosaicos cargados. [16]

3. ENTORNO GEOGRÁFICO Y VARIEDADES AUTÓCTONAS

3.1 Introducción

Como primer paso a la hora de realizar este proyecto, se ha hecho un análisis de las características de la zona de desarrollo del mismo, muy importante para conocer el problema y poder afrontarlo. Se han considerado distintos aspectos: relieve, geología, vegetación natural, etc. En segundo lugar se ha abordado el problema de la pérdida de variedad de especies hortofrutícolas, en detrimento de las variedades locales.

En el presente proyecto se realiza una recopilación de variedades hortofrutícolas autóctonas que serán añadidas a la aplicación, para que puedan ser censadas y añadidas por los usuarios de dicha aplicación. También, de forma general, se permite que los usuarios puedan añadir otras variedades "modernas", para ampliar la utilidad de la aplicación, ya que puede haber usuarios interesados en estas variedades, que buscan intercambiar al igual que el resto.

3.2 Variedades autóctonas

Las vegas de los ríos de la cuenca del río Segura se han caracterizado siempre por la gran diversidad genética de sus huertas, contando con numerosas variedades de frutas y hortalizas y siendo en su mayor parte variedades que han ido llegando y aclimatándose a lo largo de la historia e incluso, la prehistoria.

Figura 19: *Manzano de secano, autóctono del campo de Cartagena y Águilas* [17]

El cultivo continuado de algunas de estas especies ha dado lugar a la aparición de formas locales que bien podrían denominarse autóctonas. Esta riqueza, acumulada a lo largo de los siglos, se ha ido reduciendo de una forma dramática en las últimas décadas debido a la sustitución por variedades más comerciales y el abandono de los cultivos con el traslado de la población a las ciudades, encontrándose en la actualidad al borde dela extinción.

La huerta de Murcia es una de las huertas más emblemáticas del Mediterráneo debido a su extensión y antigüedad, ya que todavía podemos encontrar una gran biodiversidad agraria por la estructura minifundista de la propiedad, abundando sobre todo las plantaciones de limones y naranjas, mezcladas con diversas variedades de melocotoneros, albaricoqueros, ciruelos, membrilleros, manzanos, perales, granados, higueras, palmeras, etc. En menor cantidad se encuentran el cultivo de lechuga, coliflor (pava), patata, berenjena, judía (bajoca), brócoli, tomate, etc.

La falta de relevo generacional, la poca rentabilidad de los productos y el desarrollo urbano de la ciudad y las pedanías están poniendo en peligro la supervivencia de este paisaje cultural de enorme valor. [18]

3.3 Variedades modernas

En la región de Murcia se han desarrollado multitud de variedades modernas, entre otros motivos, para conseguir variedades más resistentes o con una cosecha mayor. Entre ellas cabe destacar la del albaricoquero, que contaba con una oferta varietal muy reducida, una producción irregular, un tiempo de oferta muy pequeño (mayo-junio) y una escasa rentabilidad para la industria, además de ser muy susceptible al virus de la sharka, por lo que se comenzó a mejorar a partir del año 1990. [19]

Figura 20: *Mejora genética del albaricoquero*

Otra variedad que se ha visto beneficiada con la llegada de variedades nuevas a la región ha sido la del almendro, al que la expansión de las variedades tradicionales de floración temprana a zonas del interior más frías puso de manifiesto la falta de adaptación de estas variedades a estas zonas, donde las heladas durante la floración o las primeras fases de formación del fruto acababan año tras año con las cosechas.

Las variedades, ANTOÑETA y MARTA obtenidas en el CEBAS-CSIC son de floración tardía y autocompatibles, y pueden ocupar un lugar destacado dentro de la actual oferta varietal de almendro en nuestro país.

La época de maduración de las dos variedades es temprana, sobre todo ANTOÑETA que es una de las variedades de almendro de maduración más temprana, que la hace muy interesante sobre todo en las zonas más frías, donde la recolección de otras variedades se suele atrasar notablemente, coincidiendo con las lluvias otoñales.

Figura 21: *Almendra de la variedad MARTA*

Además de esto, se trata de variedades bastante resistentes a las principales enfermedades criptogámicas que afectan al almendro. [20]

3.4 Características del área de estudio

Nuestro proyecto se centra en la zona del sureste de España, por ser la zona en la que nos encontramos y una de las más perjudicadas por el desarrollo, la llamada provincia Murciano-Almeriense, que abarca la zona de costa entre Altea (Alicante) y Castell de Ferro (Granada) y que penetra hacia el interior a través de las cuencas de los ríos (Almanzora, Andárax, Segura y Vinalopó).

A su vez, posee ciertas similitudes con las provincias Mediterráneo-Ibérica Central, Catalana-Provenzal-Balear y Bética, y con el norte de África, debido a la cercanía y al clima parecido.

Figura 22: *Regiones biogeográficas de España*

En el interior, los límites de la provincia normalmente se encuentran en laderas medias meridionales de montañas, a veces por cambio de margas (tipo de roca sedimentaria compuesta principalmente de calcita y arcillas) a otro tipo de sustrato y otras por un aumento orográfico de lluvias o descenso de temperaturas.

En esta zona, como casi en toda la península se da un **clima** conocido como **mediterráneo**, extendido a lo largo del litoral, el interior de la Península y el archipiélago balear.

Figura 23: *Climas de España*

Sin embargo, existen considerables diferencias entre unas zonas y otras, lo que da lugar a tres subdivisiones:

- **El clima mediterráneo típico**. Abarca gran parte de la costa del mismo nombre, algunas zonas del interior, Ceuta, Melilla y Baleares. Las lluvias son irregulares, entre los 400 mm y los 700 mm anuales, y se concentran especialmente en otoño y primavera. Los inviernos son cortos y suaves mientras que los veranos son largos y calurosos. La temperatura media anual ronda entre los 15 °C y los 18 °C. Las ciudades representativas son Denia, Valencia, Castellón de la Plana, Oliva...

- **El clima mediterráneo con invierno frío** (erróneamente denominado "mediterráneo continentalizado"). Se localiza en la Meseta, la depresión del Ebro, parte del Guadalquivir y la zona del norte de la provincia de Alicante. Se caracteriza por tener unas temperaturas muy extremas, entre 25 °C y los -13 °C. Los inviernos son largos y muy fríos, donde las temperaturas mínimas pueden bajar hasta los -5 grados o más, y los veranos muy calurosos, donde todos los años se sobrepasan los 35 grados, e incluso los 40 grados en algunas ocasiones. Además, las precipitaciones son escasas, en torno a los 400 mm, y aparecen en forma de tormenta en los meses de julio y agosto. Las ciudades representativas son Hellín, Albacete, Alcoy, Villena, Requena...

- **El clima mediterráneo seco** o **clima semidesértico**. Aparece sobre todo en el sureste del territorio, en las zonas de Murcia, Alicante y Almería (la provincia Murciano-Almeriense). Las lluvias son extremadamente escasas, menos de 300 mm al año, lo que convierte estas zonas en áreas muy áridas, debido a que esta zona se sitúa al abrigo de las Cordilleras Béticas y no le influyen las borrascas atlánticas y a que llegan con frecuencia las masas de aire secas de África . Son frecuentes los períodos largos de sequía. Las temperaturas son semejantes a las del mediterráneo típico, aunque el calor en verano suele ser más intenso. Las ciudades representativas son Alicante, Murcia, Almería, Lorca, Cartagena... [21]

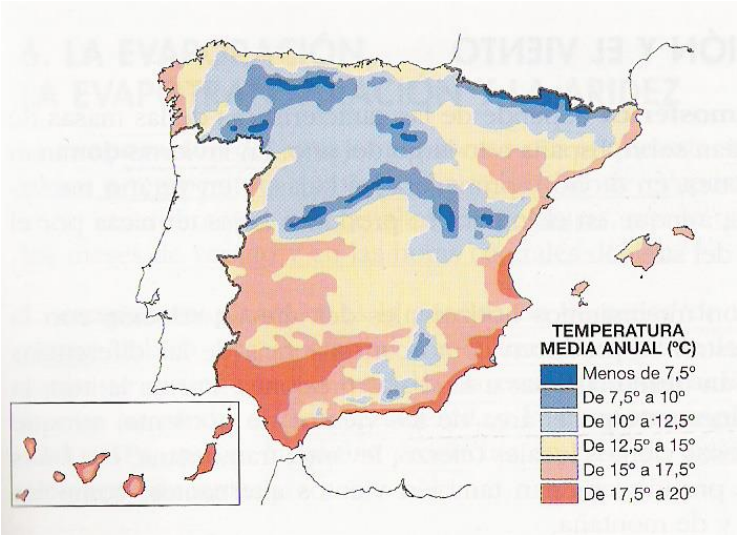

Figura 24: *Temperatura media anual de España*

3.5 Pérdida de variedades hortofrutícolas locales

Hasta el año 2025, la producción mundial de alimentos debe incrementarse más de un 75% para poder abastecer a la población mundial, la cual está previsto que crezca en torno a 2600 millones de habitantes más. Sin embargo, los recursos fitogenéticos (la suma de todas las combinaciones de genes resultantes de la evolución de una especie) de los cuales depende la seguridad alimentaria, están desapareciendo a un ritmo cada vez mayor. Según informes de la FAO (Organización de las Naciones Unidas para la Alimentación y la Agricultura), unas 50.000 variedades de interés para el sector agrario se pierden cada año en el mundo.

Durante las últimas décadas la agricultura ha experimentado una profunda transformación, que ha supuesto un incremento de la producción, con una especialización en la producción, con el empleo de semillas mejoradas, comercializadas por unas pocas empresas multinacionales. Los agricultores han sustituido el uso de cultivares tradicionales por otros mejorados que ha supuesto una fuerte erosión genética del patrimonio agrícola. En principio, los nuevos cultivares suponían una mayor producción, uniformidad, calidad externa y resistencia a ciertas enfermedades, por lo que se fueron marginando las variedades tradicionales. La implantación masiva de variedades mejoradas ha tenido efectos negativos graves como la perdida abundante de diversidad genética y una reducción de la rusticidad de los cultivos, aumentando la vulnerabilidad de los mismos frente a inesperados cambios ambientales o a la aparición de nuevas plagas y enfermedades.

Figura 25: ***Multinacionales poseedoras de la gran mayoría de semillas a nivel mundial***

El reconocimiento de la erosión genética como un grave problema tiene lugar durante los años 50, cuando el desarrollo agrícola llega a las zonas del planeta con mayor diversidad genética, siendo en este momento cuando se comienza a poner en marcha medidas globales para la conservación de los recursos fitogenéticos aún existentes. En la actualidad, la necesidad de preservar estos recursos es aceptada de forma generalizada, implicando las estrategias de conservación numerosos aspectos, desde los técnicos a los políticos.

La pérdida de las variedades, además, supone inevitablemente la pérdida de la cultura tradicional de los campesinos, sus conocimientos y costumbres, así como de los paisajes agrarios tradicionales. Por todo ello, rescatar las variedades locales que aún perviven, conservadas sobre todo en zonas marginales, requiere esfuerzos importantes y urgentes.

Consciente de este problema, la Unión Europea está cambiando sus políticas agrarias, potenciando un modelo agrícola sostenible, que priorice un mayor uso de prácticas respetuosas con el medio ambiente. En este proceso se ha consolidado la agricultura ecológica como una opción productiva diversificada y sostenible. Este tipo de agricultura ha experimentado un crecimiento positivo, tanto en número de agricultores, cooperativas y empresas de conservas de productos vegetales, como en la superficie de producción. La superficie dedicada a este tipo de cultivo no ha dejado de aumentar y se presenta como la mejor alternativa para la recuperación ambiental y social de las zonas rurales, así como para obtener productos de una calidad superior.

Sin embargo, a pesar de existir espacio potencial para el uso de variedades locales en agricultura ecológica, lo cierto es que hay una escasa disponibilidad de semilla ecológica de

variedades locales en España, lo que obliga al empleo mayoritario de variedades convencionales, producidas por multinacionales de semillas en lugares muy alejados de las prácticas hortícolas ecológicas aplicables en el ámbito del mediterráneo. Es importante destacar los graves problemas que presenta para un agricultor la reproducción de las variedades tradicionales, a causa de la degeneración varietal que puede producirse si no se realiza con estrictos criterios científicos, lo que aumenta también la dependencia de los agricultores respecto a empresas productoras de estas semillas.

España es uno de los países de la Unión Europea con un patrimonio agrícola más rico y diversificado. Ello es consecuencia de poseer una gran diversidad de condiciones agroclimáticas y de haber sido un lugar de establecimiento de culturas muy diversas. Desde los tiempos prehistóricos nuestro país ha sido siempre un pueblo abierto y permeable a todo tipo de influencias culturales, pueden destacarse como las más importantes, las primeras aportaciones culturales procedentes de Oriente Próximo, la romanización, la llegada de los árabes que introdujeron nuevos cultivos hortofrutícolas (berenjenas, calabazas, higos, limones, granadas, etc.) procedentes de la India y Pakistán y finalmente el descubrimiento de América, que supone la difusión en España de nuevas plantas hortícolas como el tomate, pimiento, patata, etc. Todo ello junto a la selección realizada por los propios agricultores en pequeños huertos familiares han generado una enorme Biodiversidad Agrícola, que constituye un importante patrimonio genético y cultural de nuestra nación.

Figura 26: *Granada (Punica granatum)*

Cabe señalar, que desde hace bastantes años, el material recolectado está siendo utilizado por distintos agricultores especializados en mercados locales para su suministro a restaurantes. En los últimos años el auge experimentado por la agricultura ecológica ha permitido que muchas

de las variedades tradicionales conservadas en el IMIDA vuelvan a ser utilizadas en este tipo de cultivo.

La recuperación de la cultura tradicional agraria debe constituir un objetivo prioritario ante el peligro de la desaparición de los portadores de esta cultura. Esta situación requiere un mayor apoyo de las instituciones públicas y privadas para la recuperación y conservación de variedades locales, a través de ayudas a los agricultores, cooperativas de productores, consumidores, etc., que tengan como objetivo la comercialización de variedades locales en peligro de extinción. [22]

4. METODOLOGÍA Y DESARROLLO DEL PROYECTO

4.1 Introducción

En este apartado se tratará en detalle cómo se han realizado las diferentes ramas del proyecto, explicando en profundidad diferentes partes del proyecto, como las bases de datos necesarias para su funcionamiento y el código de las herramientas más importantes que dan forma a esta plataforma.

Como el código de la aplicación es tan extenso, para poder explicar en profundidad algunas partes, la mayoría del código de la aplicación no será comentado, aunque sí todas las partes más importantes y esenciales de la aplicación.

4.2 Bases de datos

Todos los datos necesarios en la aplicación van a residir en la base de datos "TreeMap" de Sashido, tanto para los árboles, como para marcadores o chats. Esta base de datos fue diseñada para la aplicación web de TreeMap y se ha adaptado para funcionar en Android (versión Mobile de TreeMap).

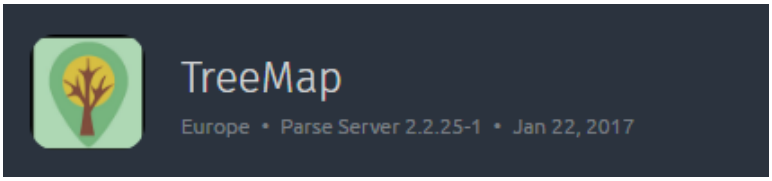

Figura 27: *Base de datos "TreeMap"*

4.2.1 TreeMap Mb

Tenemos por un lado los datos necesarios para el funcionamiento de la aplicación, en tres tablas principales.

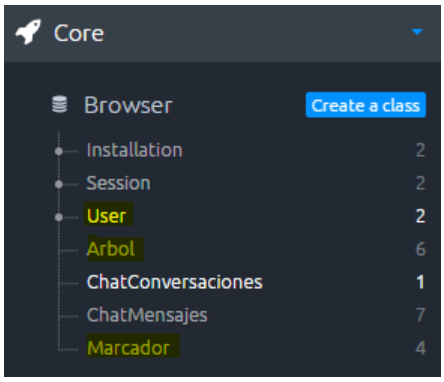

Figura 28: *Tablas de la base de datos de TreeMapMb*

La tabla "Arbol" se encarga de proporcionar a nuestra plataforma los datos de los diferentes árboles que podrán ser posteriormente añadidos por los usuarios, siendo sus campos principales el de Familia (división más general entre árboles), Especie (siendo este campo una lista con las diferentes variedades existentes de cada familia), además de los campos autogenerados por Sashido.

Por otro lado, en la tabla "Marcador" se encuentran los datos de los marcadores, el usuario que lo ha añadido, el tipo de árbol que es (es decir, a que familia y especie pertenece), la ubicación (con latitud y longitud), una descripción del árbol y una foto, estos dos últimos campos opcionales.

Por último tenemos la tabla "User" que recoge los datos de los usuarios registrados en la plataforma, tales como su nombre y apellidos, nombre de usuario, su correo electrónico, su contraseña encriptada, su foto (de manera opcional), además de otros campos generados de forma automática como la fecha de creación, la de modificación, el objectId, etc.

A continuación podemos ver el ejemplo de la tabla "Marcador" con algunos datos introducidos a modo de prueba:

Figura 29: *Tabla "Marcador", parte 1*

Usuari.. Pointer <_Use...	createdAt Date	Especie String	Familia String	+ Add a new column
GHDNOmRcKE	28 May 2017 at 19:16:20...	Otra	Manzano	
GHDNOmRcKE	28 May 2017 at 18:53:47...	Marranero	Manzano	
GHDNOmRcKE	28 May 2017 at 15:32:06...	aaaaa	Palmera	
GHDNOmRcKE	28 May 2017 at 15:15:48...	gggg	Jinjolero	

Figura 30: ***Tabla "Marcador", parte 2***

Sashido nos proporciona otras herramientas, como el uso de notificaciones para avisar a los usuarios de cualquier evento de una forma rápida y sencilla:

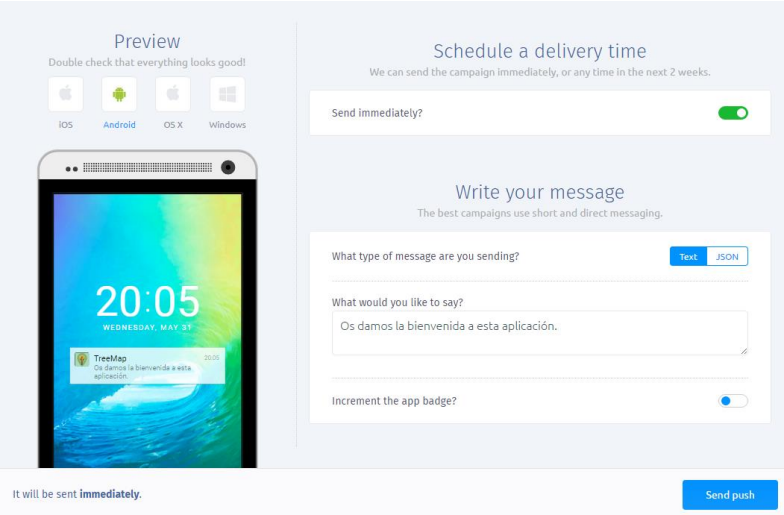

Figura 31: ***Notificaciones push***

Además, podemos ver una serie de estadísticas así como el estado de la cuenta:

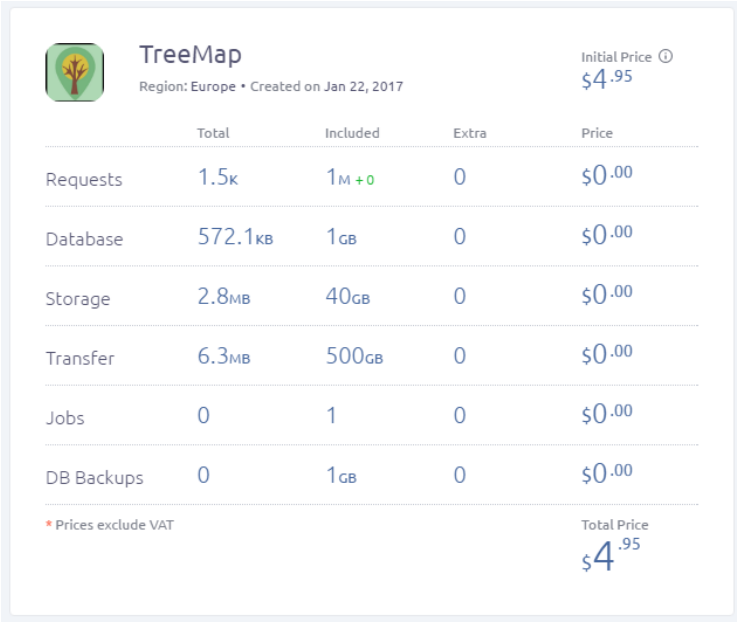

Figura 32: ***Tarifas Sashido***

4.2.2 Chat

Por otro lado, tenemos las bases de datos necesarias para el funcionamiento del chat, que se pueden separar en dos tablas, para gestionarlo de una forma sencilla.

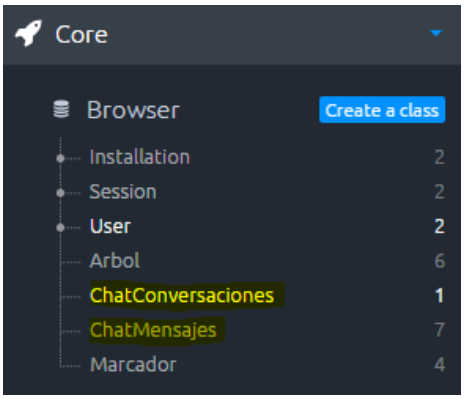

Figura 33: *Base de datos chat*

En la tabla "ChatConversaciones" se encuentran todas las conversaciones disponibles del chat, cuenta con campos como el emisor y el receptor, y además, la clave de la conversación, el elemento que los une, que no es otro que el propio marcador (árbol) por el cual comienzan la conversación.

Los mensajes de cada una de las conversaciones los podemos encontrar en la tabla "ChatMensajes", en la que encontramos columnas para saber si el mensaje ha sido leído, a qué conversación pertenece el mensaje, la hora a la que se envió, etc.

Podemos ver a modo de ejemplo, el formato de la tabla ChatMensajes.

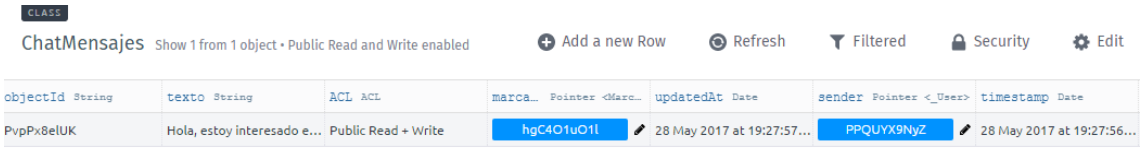

Figura 34: *Tabla "ChatMensajes", parte 1*

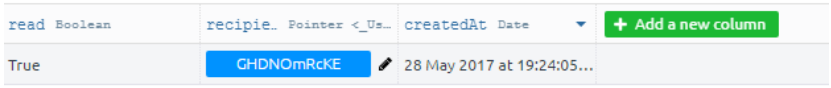

Figura 35: *Tabla "ChatMensajes", parte 2*

4.2.3 Conexión

Para realizar la conexión con la base de datos, al utilizar Parse, realizaremos la conexión dentro del paquete ParseHelper. Dentro de este paquete encontraremos la clase ParseApplication, en la que tenemos que añadir las Keys de nuestra cuenta para poder realizar una conexión segura. Por seguridad, las claves han sido tapadas para mantener el acceso a la base de datos de la forma más segura posible.

```
public class ParseApplication extends Application {

    public static final String PARSE_APPLICATION_ID = "[redacted]";
    public static final String PARSE_CLIENT_KEY = "[redacted]";
    public static final String PARSE_SERVER_URL = "[redacted]";

    public static final String GCM_SENDER_ID = "[redacted]";

    @Override
    public void onCreate() {
        super.onCreate();

        Parse.addParseNetworkInterceptor(new ParseLogInterceptor());
        Parse.initialize(new Parse.Configuration.Builder(getApplicationContext())
                .applicationId(PARSE_APPLICATION_ID)
                .clientKey(PARSE_CLIENT_KEY)
                .server(PARSE_SERVER_URL)
                .build()
        );

        registerActivityLifecycleCallbacks(new AdjustLifecycleCallbacks());

    }
```

Figura 36: *Código para la conexión con la base de datos*

Esta clase contiene los parámetros de configuración para conectarnos a la base de datos Parse y poder acceder a ella desde todo nuestro código.

4.3 Estructura

Todas las diferentes páginas que conforman la plataforma siguen una estructura bien determinada, que hace que sea escalable y modificable, ya que está todo segmentado y bien identificado, además de ser útil para futuras mejoras y actualizaciones.

Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

- Módulos de apps para Android.

- Módulos de bibliotecas.
- Módulos de Google App Engine.

De manera predeterminada, Android Studio muestra los archivos del proyecto en la vista de proyectos de Android. Esta vista se organiza en módulos para proporcionar un rápido acceso a los archivos de origen clave de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de **Secuencias de comando de Gradle** y cada módulo de la aplicación contiene las siguientes carpetas:

- **manifests**: contiene el archivo AndroidManifest.xml.
- **java**: contiene los archivos de código fuente de Java, incluido el código de prueba JUnit.
- **res**: Contiene todos los recursos, como diseños XML, cadenas de IU e imágenes de mapa de bits.

Todas las Activitys, páginas o ventanas de la aplicación se encuentran perfectamente identificados con el nombre de la acción que realizan dentro de la APP, dentro de la carpeta java.

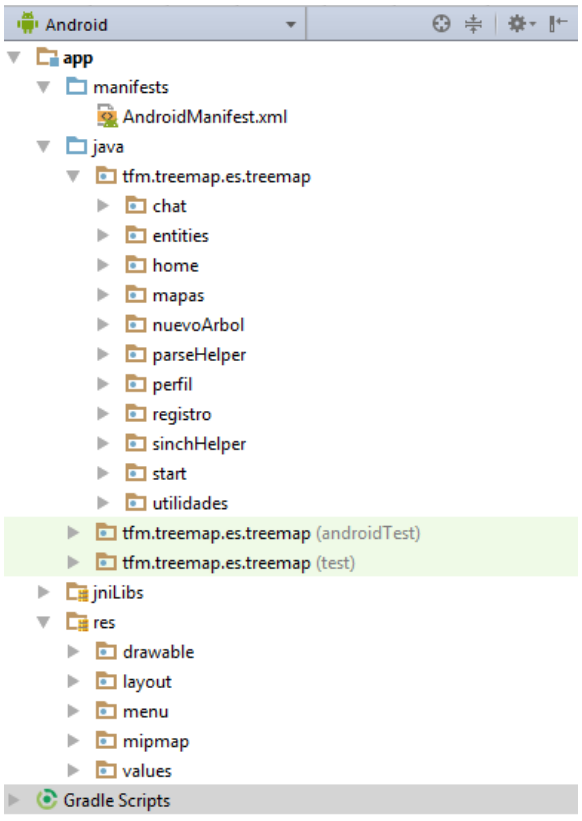

Figura 37: *Estructura del proyecto*

4.3.1 Pantalla de inicio

La primera pantalla que vemos en la aplicación es el símbolo del proyecto y el nombre de la aplicación, que además de servir de bienvenida, nos permite identificar si se ha iniciado sesión en ese teléfono móvil, o si no lo ha hecho, lo redirigiremos a la pantalla para registrarse o iniciar sesión.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_splash);

    currentUser = ParseUser.getCurrentUser();

    new android.os.Handler().postDelayed(
            new Runnable() {
                public void run() {

                    if (currentUser == null) {

                        finish();
                        startActivity(new
Intent(getApplicationContext(), LoginHomeActivity.class));

                    } else {

                        currentUser.fetchInBackground(new
GetCallback<ParseUser>() {
                            public void done(ParseUser user,
ParseException e) {
                                if (e == null) {
                                    if
(currentUser.getBoolean("Activo")) {

                                        setInstallation();

                                        //SINCH
                                        Intent serviceIntent = new
Intent(SplashActivity.this, SinchService.class);
                                        startService(serviceIntent);

                                        Date myDate =
Calendar.getInstance().getTime();
                                        currentUser.put("Updated",
myDate);

currentUser.saveInBackground();

                                        finish();
                                        Intent intent = new
Intent(mContext, HomeActivity.class);
                                        startActivity(intent);

                                    } else {

                                        Toast.makeText(

getApplicationContext(),
```

```
getString(R.string.userNotActive),

Toast.LENGTH_LONG).show();

                                        ParseInstallation installation
= ParseInstallation.getCurrentInstallation();
                                        installation.put("user",
JSONObject.NULL);

installation.put("GCMSenderId", ParseApplication.GCM_SENDER_ID);

installation.saveInBackground(new SaveCallback() {
                                            @Override
                                            public void
done(ParseException e) {
                                                if (e == null) {

ParseUser.logOut();

                                                    finish();
                                                    startActivity(new
Intent(getApplicationContext(), LoginHomeActivity.class));

                                                } else {
                                                    finish();
                                                }
                                            }
                                        });
                                    }
                                } else if (e.getCode() == 209){

                                    Toast.makeText(
                                            getApplicationContext(),

getString(R.string.sorryError),
                                            Toast.LENGTH_LONG).show();

                                    ParseUser.logOut();

                                    finish();
                                    startActivity(new
Intent(getApplicationContext(), LoginHomeActivity.class));

                                }
                            }
                        });
                    }
                }
            }, 2500
    );
}
```

Código de la aplicación 1

Además, comprueba si el usuario ha sido baneado o no (cuenta suspendida por el administrador) y si todo va bien, actualiza los datos referentes a la sesión del usuario, como por ejemplo la última vez que ha accedido a la página.

4.3.2 Menú

El menú o navBar es la parte de la aplicación que permite ir cambiando de ventanas, en función de lo que queramos hacer. Este navBar se construye de una forma sencilla y como vemos consta de: Imagen de perfil del usuario y su nombre (a modo informativo), los botones para acceder a las diferentes pantallas de la aplicación y el botón de cerrar sesión.

```
private void setNavBar() {

    RelativeLayout relativeHeader = (RelativeLayout)
findViewById(R.id.relativeHeader);
    imagenPerfil = (ImageView) findViewById(R.id.imagenPerfil);
    tvNombreUsuario = (TextView) findViewById(R.id.tvNombreUsuario);

    RelativeLayout relativeMapa = (RelativeLayout)
findViewById(R.id.relativeMapa);
    RelativeLayout relativeForo = (RelativeLayout)
findViewById(R.id.relativeChat);
    RelativeLayout relativeContacto = (RelativeLayout)
findViewById(R.id.relativeContacto);
    RelativeLayout relativeAnadir = (RelativeLayout)
findViewById(R.id.relativeAnadir);

    Button btLogOut = (Button) findViewById(R.id.btLogOut);

    //SETTING USER DATA
    setUserData();

    //CLICK BUTTONS
    relativeHeader.setOnClickListener(new managementButtons());
    relativeMapa.setOnClickListener(new managementButtons());
    relativeForo.setOnClickListener(new managementButtons());
    relativeContacto.setOnClickListener(new managementButtons());
    relativeAnadir.setOnClickListener(new managementButtons());
    btLogOut.setOnClickListener(new managementButtons());

}
```

Código de la aplicación 2

Para poder realizar este cambio de pantalla, tenemos un detector de pulsaciones, para identificar donde ha pulsado el usuario y redirigirlo a la página que desee, que es principalmente la función del menú, que podrá ser visible siempre desde la ventana Home o principal.

```
private class managementButtons implements View.OnClickListener {

    Intent intent;

    @Override
    public void onClick(View v) {
        switch (v.getId()) {

            case R.id.relativeHeader:

                intent = new Intent(mContext, PhotoActivity.class);
```

```
                intent.putExtra("FROM_PROFILE", true);
                startActivity(intent);

                break;

            case R.id.relativeMapa:
                startActivity(new Intent(mContext,
MapsActivity.class));
                break;

            case R.id.relativeAnadir:
                Intent intent = new Intent(mContext,
AnadirActivity.class);
                startActivityForResult(intent, REQUEST_AÑADIR_ARBOL);
                break;

            case R.id.relativeChat:
                startActivity(new Intent(mContext,
ChatMainActivity.class));
                break;

            case R.id.relativeContacto:
                Intent intentModificarPerfil2 = new Intent(mContext,
PerfilUsuarioActivity.class);
                startActivityForResult(intentModificarPerfil2,
REQUEST_MODIFICAR_PERFIL);
                break;

            case R.id.btLogOut:
                closeSesion();
                break;

            case R.id.relativeActionMenu:
                if (mDrawerLayout.isDrawerOpen(Gravity.RIGHT)) {
                    mDrawerLayout.closeDrawer(Gravity.RIGHT);
                } else {
                    mDrawerLayout.openDrawer(Gravity.RIGHT);
                }

                break;

        }
    }
}
```

Código de la aplicación 3

En este código podemos ver cómo va buscando el lugar de la pulsación, para redirigir a la página a excepción de dos casos, el de cerrar sesión, que llama al método directamente y que veremos algún apartado más adelante y el que muestra la foto de nuestro perfil.

El último caso hace que el menú se despliegue de derecha a izquierda cuando se hace el gesto de deslizar, ya que el menú se puede abrir pulsando sobre el icono o deslizando el dedo.

4.3.3 Pantalla de bienvenida

La página principal o pantalla de bienvenida una vez iniciada la sesión, es una lista con los árboles que hemos añadido en la aplicación, de una forma sencilla, por lo que en el código debemos crear la lista.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);

    //DECLARATIONS
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setTitle(R.string.misArboles);
    setSupportActionBar(toolbar);
    toolbar.setNavigationIcon(R.mipmap.ic_launcher);

    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    treeList = (ListView) findViewById(R.id.treeList);
    LinearLayout linearEmpty = (LinearLayout)
findViewById(R.id.linearEmpty);

    //IMPLEMENTATION
    treeList.setEmptyView(linearEmpty);
    treeList.setOnItemClickListener(this);
    adapter = new AdapterListArbolesHome(mContext, listaArboles);

    progressDialog = new ProgressDialog(HomeActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    DownloadArboles();
    setNavBar();

    treeList.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
        @Override
        public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
int pos, long id) {

            showPromptDeleteItem(mContext,
adapter.getItem(pos).getArbolId(), pos);
            return true;
        }
    });
```

Código de la aplicación 4

Al principio del código vemos como se le da forma a la pantalla con el "Toolbar", que es la barra fija que aparece en la aplicación en las diferentes pantallas, sobre la cual aparecen los botones para volver atrás, los títulos de las pantallas, etc.

También tenemos “progressDialog”, que es una ventana (popUp) que muestra un mensaje, en este caso, el de Cargando, mientras se accede a la base de datos y se crea la lista. Por su parte, el “adapter” se encarga de generar las filas de la tabla.

En la parte final del código, podemos ver la acción al realizar una pulsación larga, que permite seleccionar si se elimina el árbol, pudiendo así borrar un árbol.

También podemos ver como después de crear la estructura, se llama al método “DownloadArboles” que hace una consulta a la base de datos y va recogiendo los datos de todos los árboles añadidos por el usuario.

```
private void DownloadArboles() {

    listaArboles.clear();

    final ParseQuery<ParseObject> query =
ParseQuery.getQuery("Marcador");
    query.whereEqualTo("Usuario", currentUser);
    query.orderByDescending("createdAt");
    query.findInBackground(new FindCallback<ParseObject>() {
        public void done(final List<ParseObject> arbolesRecibidos,
ParseException e) {
            if (e == null && !arbolesRecibidos.isEmpty()) {

                String fechaCreacion, address;
                ItemHomeObject arbolItem;

                for (int i = 0; i < arbolesRecibidos.size(); i++) {

                    fechaCreacion =
getDateComplete(arbolesRecibidos.get(i).getCreatedAt());


if(arbolesRecibidos.get(i).getParseGeoPoint("Localizacion") != null) {
                        address =
getCompleteAddressString(arbolesRecibidos.get(i).getParseGeoPoint("Loc
alizacion").getLatitude(),
arbolesRecibidos.get(i).getParseGeoPoint("Localizacion").getLongitude(
), mContext, TAG);
                    } else {
                        address = "";
                    }

                    arbolItem = new ItemHomeObject(
                            arbolesRecibidos.get(i).getObjectId(),

arbolesRecibidos.get(i).getString("Familia"),

arbolesRecibidos.get(i).getString("Especie"),
                            fechaCreacion,
                            address
                    );

                    listaArboles.add(arbolItem);
                }
```

```
                treeList.setAdapter(adapter);
                progressDialog.dismiss();

            } else if (e != null){

                Log.e(TAG, "DownloadChats: " + e.getMessage());
                progressDialog.dismiss();
            } else {
                progressDialog.dismiss();
            }
        }
    });
}
```

Código de la aplicación 5

Se puede ver como se construye la query mediante Parse, sobre la tabla query, filtrando por el usuario actual y listando por orden de creación, añadiendo cada árbol a una lista con todos los valores que se muestran al usuario: Familia, Especie, Localización y Fecha.

4.3.4 Perfil

En esta pantalla el usuario puede ver y editar sus datos personales, a través del botón “Editar”, actualizando en la base de datos todas las modificaciones.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_perfil_usuario);

    //DECLARATION
    RelativeLayout btAtrasPerfil = (RelativeLayout)
findViewById(R.id.btAtrasPerfil);
    btEditarPerfil = (RelativeLayout)
findViewById(R.id.btEditarPerfil);
    btGuardarPerfil = (RelativeLayout)
findViewById(R.id.btGuardarPerfil);

    LinearLayout btCambiarContraseña = (LinearLayout)
findViewById(R.id.btCambiarContraseña);
    imagenPerfil = (ImageView) findViewById(R.id.imagenPerfil);

    etNombreApellidos = (EditText)
findViewById(R.id.etNombreApellidos);
    etNombreUsuario = (EditText) findViewById(R.id.etNombreUsuario);
    etEmail = (EditText) findViewById(R.id.etEmail);

    //IMPLEMENTATION
    setPerfil();

    //EDIT TEXT
    imagenPerfil.setEnabled(false);

    etNombreApellidos.setFocusableInTouchMode(false);
    etNombreApellidos.setFocusable(false);

    etNombreUsuario.setFocusableInTouchMode(false);
    etNombreUsuario.setFocusable(false);
```

```
    etEmail.setFocusableInTouchMode(false);
    etEmail.setFocusable(false);

    //BUTTON TO CHANGE PHOTO
    imagenPerfil.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            clickImage = true;

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                if
(!Settings.System.canWrite(getApplicationContext())){
                    requestPermissions(new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.READ_EXTERNAL_STORAGE}, 2909);
                    selectImage();
                }
                else {
                    selectImage();
                }
            } else {
                selectImage();
            }

        }
    });

    btCambiarContraseña.setOnClickListener(new View.OnClickListener()
{
        public void onClick(View v) {
            cambiarContraseña();
        }
    });

    btAtrasPerfil.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            finish();
        }
    });

    btEditarPerfil.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {

            imagenPerfil.setEnabled(true);
            btEditarPerfil.setVisibility(View.GONE);
            btGuardarPerfil.setVisibility(View.VISIBLE);

            etNombreApellidos.setFocusableInTouchMode(true);
            etNombreApellidos.setFocusable(true);

            etNombreUsuario.setFocusableInTouchMode(true);
            etNombreUsuario.setFocusable(true);

            etEmail.setFocusableInTouchMode(true);
            etEmail.setFocusable(true);

        }
    });

    btGuardarPerfil.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
```

```
            if (!validate()) {
                return;
            }

            saveChanges();
        }
    });

}
```

Código de la aplicación 6

Se declaran ambos botones, el de editar y también el de guardar, además de todos los campos de texto que se convertirán en campos modificables al pulsar el botón de "Editar".

En la parte del código de la imagen, se puede ver como se solicitan los permisos para acceder a la cámara o a la galería, que en el caso de ser la primera vez, aparecerá una notificación solicitando el permiso.

Al final del código, encontramos el método "`btEditarPerfil.setOnClickListener`" que se encarga de esperar el click sobre el botón editar, para permitir editar todos los campos, incluido el de cambiar la foto.

Para guardar los cambios tenemos el siguiente código:

```
btGuardarPerfil.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        if (!validate()) {
            return;
        }

        saveChanges();
    }
});
```

Código de la aplicación 7

Que espera el click sobre el botón para llamar al método "saveChanges()" que guarda los datos en la base de datos:

```
protected void saveChanges(){

    progressDialog = new ProgressDialog(PerfilUsuarioActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.guardandoCambios));
    progressDialog.setCancelable(false);
    progressDialog.show();

    nombreApellidos = etNombreApellidos.getText().toString();
    nombreUsuario = etNombreUsuario.getText().toString();
    email = etEmail.getText().toString();

    currentUser.put("Nombre", nombreApellidos);
```

```
    currentUser.put("username", nombreUsuario);
    currentUser.put("email", email);

    final ParseFile parseThumbnail = new ParseFile(imageBytes);

    if(clickImage){
        parseThumbnail.saveInBackground(new SaveCallback() {
            public void done(ParseException e) {
                // If successful add file to user and
signUpInBackground
                if (null == e) {

                    currentUser.put("Imagen", parseThumbnail);
                    currentUser.saveInBackground(new SaveCallback() {
                        public void done(ParseException e) {

                            if(e == null){

                                Intent returnIntent = new Intent();

setResult(Activity.RESULT_OK,returnIntent);
                                finish();
                                progressDialog.dismiss();

                                Toast.makeText(getBaseContext(),
getString(R.string.cambiosGuardados), Toast.LENGTH_LONG).show();

                            }else{
                                progressDialog.dismiss();
                                Log.e(TAG,
"saveChanges:imageParse.saveInBackground:currentUser.saveInBackground:
" + e.toString());
                                Toast.makeText(getBaseContext(), "Lo
siento, ha habido un error: " + e.getMessage(),
Toast.LENGTH_LONG).show();
                            }
                        }
                    });
                } else {
                    progressDialog.dismiss();
                    Log.e(TAG,
"saveChanges:imageParse.saveInBackground: " + e.toString());
                    Toast.makeText(getBaseContext(), "Lo siento, ha
habido un error: " + e.getMessage(), Toast.LENGTH_LONG).show();
                }
            }
        });
    } else {
        currentUser.saveInBackground(new SaveCallback() {
            public void done(ParseException e) {

                if(e == null){

                    Intent returnIntent = new Intent();
                    setResult(Activity.RESULT_OK,returnIntent);
                    finish();

                    progressDialog.dismiss();
                    Toast.makeText(getBaseContext(),
getString(R.string.cambiosGuardados), Toast.LENGTH_LONG).show();
```

```
                } else{
                    progressDialog.dismiss();
                    Log.e(TAG,
"saveChanges:currentUser.saveInBackground: " + e.toString());
                    Toast.makeText(getBaseContext(), "Lo siento, ha
habido un error: " + e.getMessage(), Toast.LENGTH_LONG).show();
                }
            }
        });
    }
}
```

Código de la aplicación 8

Como hemos visto en otros métodos encargados de guardar datos en Parse, utilizamos el objeto "currentUser" al que vamos añadiendo uno a uno todos los datos, que se corresponden con columnas de la tabla "User".

Se puede ver que gran parte del método se encarga de la gestión de errores, ya que en caso de haberlos, es muy importante tener un buen "Log", que no es más que un resumen de lo que vamos haciendo en cada parte del código para poder aislar el error, ya que el acceso a base de datos es una parte bastante delicada de la aplicación.

Por último, podemos ver el método al que se llama desde el "onCreate", que es el que se encarga de rellenar los campos y mostrarlos en la pantalla de una forma sencilla.

```
private void setPerfil(){

    currentUser = ParseUser.getCurrentUser();

    try {

        Bitmap thumbnail;

        if(currentUser.getParseFile("Imagen") != null){
            thumbnail =
BitmapFactory.decodeByteArray(currentUser.getParseFile("Imagen").getDa
ta(), 0, currentUser.getParseFile("Imagen").getData().length);
        }else{
            thumbnail =
BitmapFactory.decodeResource(getApplicationContext().getResources(),
R.drawable.ic_default);
        }

        imagenPerfil.setImageBitmap(thumbnail);

    } catch (ParseException e) {
        e.printStackTrace();
    }

    etNombreApellidos.setText(currentUser.getString("Nombre"));
    etNombreUsuario.setText(currentUser.getUsername());
    etEmail.setText(currentUser.getEmail());

}
```

4.3.4.1 Modificar contraseña

En esta pantalla dentro del perfil, el usuario puede modificar de forma sencilla su contraseña. Como hemos visto en el apartado anterior, hay un método llamado "btCambiarContraseña.setOnClickListener" que se encarga de actuar cuando se pulsa el botón, llamando al método "cambiarContraseña()".

```
public void cambiarContraseña() {

        LayoutInflater li = LayoutInflater.from(mContext);
        View promptsView =
li.inflate(R.layout.prompt_cambiar_contrasena, null);

        android.support.v7.app.AlertDialog.Builder alertDialogBuilder
= new android.support.v7.app.AlertDialog.Builder(
                mContext);
        alertDialogBuilder.setView(promptsView);

        //PROMPT DECLARATIONS
        final EditText etContraseña = (EditText)
promptsView.findViewById(R.id.etContraseña);
        final EditText etRepetirContraseña = (EditText)
promptsView.findViewById(R.id.etRepetirContraseña);

        RelativeLayout btCerrar = (RelativeLayout)
promptsView.findViewById(R.id.btCerrar);
        Button btCambiarContrasena = (Button)
promptsView.findViewById(R.id.btCambiarContrasena);

        // create alert dialog
        alertDialogBuilder
                .setCancelable(false);
        final android.support.v7.app.AlertDialog alertDialog =
alertDialogBuilder.create();
//        alertDialog.getWindow().setGravity(Gravity.BOTTOM);
        alertDialog.show();

        btCerrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                alertDialog.cancel();
            }
        });

        //SETTING FUNCIONALITY BUTTONS
        btCambiarContrasena.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {

                if
(!validateContraseña(etContraseña.getText().toString(),
etRepetirContraseña.getText().toString())) {
                    return;
```

```
                }

                progressDialog = new
ProgressDialog(PerfilUsuarioActivity.this);
                progressDialog.setIndeterminate(true);

progressDialog.setMessage(getString(R.string.guardandoCambios));
                progressDialog.setCancelable(false);
                progressDialog.show();

                final String nameUser = currentUser.getUsername();


currentUser.setPassword(etContraseña.getText().toString());
                currentUser.saveInBackground(new SaveCallback() {
                    public void done(ParseException e) {

                        if(e == null){
                            currentUser.logOut();
                            currentUser.logInInBackground(nameUser,
etContraseña.getText().toString(), new LogInCallback() {
                                @Override
                                public void done(ParseUser parseUser,
ParseException e) {

                                    if(e == null){

                                        progressDialog.dismiss();

Toast.makeText(getBaseContext(), getString(R.string.cambiosGuardados),
Toast.LENGTH_LONG).show();
                                        alertDialog.cancel();
                                    } else {
                                        progressDialog.dismiss();
                                    }
                                }
                            });
                        } else{
                            progressDialog.dismiss();
                            Log.e(TAG, "error:" + e.toString());
                            Toast.makeText(getBaseContext(), "Lo
siento, ha habido un error: " + e.getMessage(),
Toast.LENGTH_LONG).show();
                        }
                    }
                });

            }
        });
    }
```

Código de la aplicación 10

En este método se crea una nueva ventana (un popUp) que muestra dos campos, en ambos, podemos introducir texto que aparecerá oculto bajo puntitos (****), y que será comprobado antes de registrarlo en la base de datos. Si el proceso se realiza de forma correcta, después de la pantalla que nos indica que se están guardando los cambios, aparecerá un mensaje (Toast) confirmando que se han guardado los cambios.

4.3.5 Iniciar sesión

En esta pantalla se hará simplemente un acceso a base de datos con los datos introducidos por el usuario en la pantalla, en el que se buscará si existe el usuario con la contraseña introducida, si es así, llamará al método "onLoginSuccess" que muestra un mensaje de bienvenida y redirige a la pantalla de inicio.

```
private void login(){

    if (!validate()) {
        return;
    }

    btLogin.setEnabled(false);

    progressDialog = new ProgressDialog(LoginActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    // Retrieve the text entered from the EditText
    nombreUsuario = etNombreUsuario.getText().toString();
    contraseña = etContraseña.getText().toString();

    ParseUser.logInInBackground(nombreUsuario, contraseña, new
LogInCallback() {
        public void done(ParseUser user, ParseException e) {
            if (user != null) {
                onLoginSuccess();
            } else {

                btLogin.setEnabled(true);
                progressDialog.dismiss();
                Toast.makeText(
                        getApplicationContext(),
                        e.getMessage(),
                        Toast.LENGTH_LONG).show();

            }
        }
    });
}
```

Código de la aplicación 11

Antes de acceder a la base de datos, para evitar que se realicen multitud de accesos sin éxito, por motivos como puede ser olvidar rellenar un campo del formulario, se realiza una validación con el método "validate".

```
public boolean validate() {
    boolean valid = true;

    nombreUsuario = etNombreUsuario.getText().toString();
```

```
    contraseña = etContraseña.getText().toString();

    if (nombreUsuario.isEmpty() || nombreUsuario.length() < 4) {
        Toast.makeText(getApplicationContext(),
                getString(R.string.nombreUsuario4char),
                Toast.LENGTH_LONG).show();

        btLogin.setEnabled(true);
        valid = false;
    }else if (contraseña.isEmpty()) {
        Toast.makeText(getApplicationContext(),
                getString(R.string.contraseñaVacia),
                Toast.LENGTH_LONG).show();

        btLogin.setEnabled(true);
        valid = false;
    }

    return valid;
```

Código de la aplicación 12

En el que se comprueba, además de que no esté vacío ninguno de los dos campos, la longitud del nombre de usuario, ya que al registrarse, se exige que sea mayor de 4 caracteres. Si la comprobación es correcta, se devolverá un booleano con valor “true”.

4.3.6 Registro

La pantalla de registro tiene varios campos que el usuario debe rellenar, como son el nombre de usuario, el nombre y apellidos, el correo electrónico, la contraseña por duplicado y la opción de añadir una foto.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_registro);

    //DECLARATIONS
    etNombre = (EditText) findViewById(R.id.etNombre);
    etNombreUsuario = (EditText) findViewById(R.id.etNombreUsuario);
    etEmail = (EditText) findViewById(R.id.etEmail);
    etContraseña = (EditText) findViewById(R.id.etContraseña);
    etRepetirContraseña = (EditText)
findViewById(R.id.etRepetirContraseña);

    imagenUsuario = (ImageView) findViewById(R.id.imagenUsuario);

    containerPic = (RelativeLayout) findViewById(R.id.containerPic);
    btFinalizar = (Button) findViewById(R.id.btFinalizar);

    //FUNCIONALITY
    btFinalizar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            signup();
        }
    });
```

```
    imagenUsuario.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                if
(!Settings.System.canWrite(getApplicationContext())){
                    requestPermissions(new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.READ_EXTERNAL_STORAGE}, 2909);
                    selectImage();
                }
                else {
                    selectImage();
                }
            } else {
                selectImage();
            }

        }
    });
}
```

Código de la aplicación 13

Podemos ver que en el método "onCreate" se ponen todos los campos que va a tener la pantalla, incluido el de la foto y el botón que finalizará el registro, siendo campos de texto los que incluyen el parámetro "EditText" y estando colocada la cámara en una situación especial, arriba en el centro de la pantalla, siendo por este motivo necesario usar el "RelativeLayout" para colocarlo en esa posición.

Como vemos desde ese método se llama al de "signup".

```
public void signup() {

    if (!validate()) {
        return;
    }

    btFinalizar.setEnabled(false);

    progressDialog = new ProgressDialog(RegistroActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    saveUserInParse();
}
```

Código de la aplicación 14

Que realiza una llamada al método "validate" para realizar una serie de comprobaciones antes de guardar el usuario en la base de datos (como hemos visto en el apartado de Iniciar sesión) y crea un popUp mientras carga la petición a la base de datos, para posteriormente ir al método "saveUserInParse".

```
public void saveUserInParse() {

    final ParseFile parseThumbnail = new ParseFile(imageBytes);

    parseThumbnail.saveInBackground(new SaveCallback() {
        public void done(ParseException e) {
            // If successful add file to user and signUpInBackground
            if (null == e) {

                // Save new user data into Parse.com Data Storage
                ParseUser user = new ParseUser();

                user.put("username", nombreUsuario);
                user.put("password", contraseña);
                user.put("Nombre", nombre);
                user.put("email", email);
                user.put("Activo", true);
                user.put("Imagen", parseThumbnail);
                Date myDate = Calendar.getInstance().getTime();
                user.put("Updated", myDate);

                user.signUpInBackground(new SignUpCallback() {
                    public void done(ParseException e) {
                        if (e == null) {

                            onSignupSuccess();

                        } else {
                            btFinalizar.setEnabled(true);
                            progressDialog.dismiss();

                            Toast.makeText(getBaseContext(),
getString(R.string.RegisterFailed) + " : " + e.getMessage(),
Toast.LENGTH_LONG).show();
                            Log.e(TAG, "Register:SignUpInBackGround:"
+ e.getMessage());
                        }
                    }
                });
            } else{
                btFinalizar.setEnabled(true);
                progressDialog.dismiss();
            }

        }
    });
}
```

Código de la aplicación 15

Este método se encarga de insertar, dentro de la tabla “User” una nueva fila con todos los datos que ha introducido el usuario en la pantalla y algún campo más para el administrador de la aplicación, como puede ser el de la fecha de actualización o el campo Activo, por si hay que cerrar la cuenta de algún usuario.

4.3.7 Cerrar sesión

Para cerrar sesión, tenemos un método desde el menú, como hemos visto antes, que llama a un método para realizar la acción.

```
private void closeSesion() {
    android.support.v7.app.AlertDialog.Builder builder = new
android.support.v7.app.AlertDialog.Builder(this);

    builder.setMessage(R.string.sureCloseSesion)
            .setCancelable(false)
            .setPositiveButton(R.string.OK,
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog,
int id) {

                            final ProgressDialog progressDialog = new
ProgressDialog(HomeActivity.this);
                            progressDialog.setIndeterminate(true);

progressDialog.setMessage(getString(R.string.closingSession));
                            progressDialog.setCancelable(false);
                            progressDialog.show();

                            ParseInstallation installation =
ParseInstallation.getCurrentInstallation();
                            installation.put("user", JSONObject.NULL);
                            installation.saveInBackground(new
SaveCallback() {
                                @Override
                                public void done(ParseException e) {
                                    if (e == null) {
                                        ParseUser.logOut();
                                        progressDialog.dismiss();

                                        Intent intent = new
Intent(mContext, LoginHomeActivity.class);

intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
Intent.FLAG_ACTIVITY_NEW_TASK);
                                        startActivity(intent);

                                    }else{
                                        progressDialog.dismiss();


Toast.makeText(getApplicationContext(),

getString(R.string.errorClossingSession),

Toast.LENGTH_LONG).show();

                                    }
                                }
                            });

                        }
                    });
```

```
    builder.setNegativeButton(R.string.Cancel,
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });
    android.support.v7.app.AlertDialog alert = builder.create();
    alert.show();
}
```

Código de la aplicación 16

En primer lugar, se muestra un mensaje para asegurarnos de que el usuario realmente quiere cerrar sesión, si el usuario pulsa sobre el botón "Sí" se procede a cerrar sesión con el método "ParseUser.*logOut*()" y si no hay ningún error, se cerrará la sesión y se volverá a la ventana donde se tiene que elegir entre iniciar sesión o registrarse.

4.3.8 Gestión de imágenes en la aplicación

Para añadir una imagen de un árbol, o también para añadir la de un usuario al registrarse, usamos el método "selectImage" , donde podemos arrancar la cámara o abrir la galería del teléfono, hacer o seleccionar la foto y guardar la foto.

```
private void selectImage() {
    final CharSequence[] items = {getString(R.string.pickPhoto),
getString(R.string.choosePhotoFromGallery),
getString(R.string.Cancel)};
    AlertDialog.Builder builder = new
AlertDialog.Builder(AnadirActivity.this);
    builder.setTitle(getString(R.string.titleAddPhoto));
    builder.setItems(items, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int item) {

            if (items[item].equals(getString(R.string.pickPhoto))) {

                ContentValues values = new ContentValues();
                values.put(MediaStore.Images.Media.TITLE, "Tomar
foto");
                imageUri = getContentResolver().insert(
                        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
values);

                Intent intent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
                startActivityForResult(intent, REQUEST_CAMERA);

            } else if
(items[item].equals(getString(R.string.choosePhotoFromGallery))) {

                Intent intent = new Intent(
                        Intent.ACTION_PICK,
                        MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                intent.setType("image/*");
                startActivityForResult(
                        Intent.createChooser(intent, "Select File"),
```

```
                        SELECT_FILE);

            } else if (items[item].equals(getString(R.string.Cancel)))
{
                dialog.dismiss();
            }
        }
    });
    builder.show();
}
```

Código de la aplicación 17

Desde el siguiente método recogemos la foto que sube el usuario y configuramos todos los detalles, la redimensionamos y desde aquí, se llama al método de compresión.

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {

    super.onActivityResult(requestCode, resultCode, data);

    if (resultCode == RESULT_OK) {
        if (requestCode == REQUEST_CAMERA) {

            try {

                imageBytes = compressImage(mContext, imageUri);

                relativeContainerPhoto.setBackgroundColor(0);
                btFoto.getLayoutParams().height = (int)
mContext.getResources().getDimension(R.dimen.image_register);
                btFoto.getLayoutParams().width = (int)
mContext.getResources().getDimension(R.dimen.image_register);


btFoto.setImageBitmap(BitmapFactory.decodeByteArray(imageBytes, 0,
imageBytes.length));
                btFoto.setScaleType(ImageView.ScaleType.CENTER_CROP);

            } catch (Exception e) {
                e.printStackTrace();
            }

        } else if (requestCode == SELECT_FILE) {

            imageBytes = compressImage(mContext, data.getData());

            btFoto.getLayoutParams().height = (int)
mContext.getResources().getDimension(R.dimen.image_register);
            btFoto.getLayoutParams().width = (int)
mContext.getResources().getDimension(R.dimen.image_register);

            relativeContainerPhoto.setBackgroundColor(0);

btFoto.setImageBitmap(BitmapFactory.decodeByteArray(imageBytes, 0,
imageBytes.length));
            btFoto.setScaleType(ImageView.ScaleType.CENTER_CROP);

        } else if (requestCode == REQUEST_SET_LOCATION) {

            Intent intentReceived = getIntent();
```

```
            setResult(RESULT_OK, intentReceived);
            finish();

        }
    }
}
```

Código de la aplicación 18

Como hemos visto, la foto pasa a ser comprimida para poder subirla a la base de datos, ya que la imagen original sería demasiado grande y ocuparía demasiado espacio en la base de datos, además de retrasar la carga de la aplicación al necesitar mayor tiempo para descargarla.

```
    //RESIZE OF IMAGES
    public static byte [] compressImage(Context mContext, Uri
imageUri) {

        String filePath = getRealPathFromURI(mContext, imageUri);
        Bitmap scaledBitmap = null;

        BitmapFactory.Options options = new BitmapFactory.Options();

//      by setting this field as true, the actual bitmap pixels are
not loaded in the memory. Just the bounds are loaded. If
//      you try the use the bitmap here, you will get null.
        options.inJustDecodeBounds = true;
        Bitmap bmp = BitmapFactory.decodeFile(filePath, options);

        int actualHeight = options.outHeight;
        int actualWidth = options.outWidth;

//      max Height and width values of the compressed image is taken
as 816x612

        float maxHeight = 816.0f;
        float maxWidth = 612.0f;
        float imgRatio = actualWidth / actualHeight;
        float maxRatio = maxWidth / maxHeight;

//      width and height values are set maintaining the aspect ratio
of the image

        if (actualHeight > maxHeight || actualWidth > maxWidth) {
            if (imgRatio < maxRatio) {
                imgRatio = maxHeight / actualHeight;
                actualWidth = (int) (imgRatio * actualWidth);
                actualHeight = (int) maxHeight;
            } else if (imgRatio > maxRatio) {
                imgRatio = maxWidth / actualWidth;
                actualHeight = (int) (imgRatio * actualHeight);
                actualWidth = (int) maxWidth;
            } else {
                actualHeight = (int) maxHeight;
                actualWidth = (int) maxWidth;

            }
        }

//      setting inSampleSize value allows to load a scaled down
version of the original image
```

```
        options.inSampleSize = calculateInSampleSize(options,
actualWidth, actualHeight);

//      inJustDecodeBounds set to false to load the actual bitmap
        options.inJustDecodeBounds = false;

//      this options allow android to claim the bitmap memory if it
runs low on memory
        options.inPurgeable = true;
        options.inInputShareable = true;
        options.inTempStorage = new byte[16 * 1024];

        try {
//          load the bitmap from its path
            bmp = BitmapFactory.decodeFile(filePath, options);
        } catch (OutOfMemoryError exception) {
            exception.printStackTrace();

        }
        try {
            scaledBitmap = Bitmap.createBitmap(actualWidth,
actualHeight,Bitmap.Config.ARGB_8888);
        } catch (OutOfMemoryError exception) {
            exception.printStackTrace();
        }

        float ratioX = actualWidth / (float) options.outWidth;
        float ratioY = actualHeight / (float) options.outHeight;
        float middleX = actualWidth / 2.0f;
        float middleY = actualHeight / 2.0f;

        Matrix scaleMatrix = new Matrix();
        scaleMatrix.setScale(ratioX, ratioY, middleX, middleY);

        Canvas canvas = new Canvas(scaledBitmap);
        canvas.setMatrix(scaleMatrix);
        canvas.drawBitmap(bmp, middleX - bmp.getWidth() / 2, middleY -
bmp.getHeight() / 2, new Paint(Paint.FILTER_BITMAP_FLAG));

//      check the rotation of the image and display it properly
        ExifInterface exif;
        try {
            exif = new ExifInterface(filePath);

            int orientation = exif.getAttributeInt(
                    ExifInterface.TAG_ORIENTATION, 0);
            Log.d("EXIF", "Exif: " + orientation);
            Matrix matrix = new Matrix();
            if (orientation == 6) {
                matrix.postRotate(90);
            } else if (orientation == 3) {
                matrix.postRotate(180);
            } else if (orientation == 8) {
                matrix.postRotate(270);
            }
            scaledBitmap = Bitmap.createBitmap(scaledBitmap, 0, 0,
                    scaledBitmap.getWidth(), scaledBitmap.getHeight(),
matrix,
                    true);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }

        FileOutputStream out = null;
        String filename = getFilename();
        ByteArrayOutputStream stream = new ByteArrayOutputStream();

        try {
            out = new FileOutputStream(filename);

//          write the compressed bitmap at the destination specified
by filename.
            scaledBitmap.compress(Bitmap.CompressFormat.JPEG, 80,
out);

            //GENERATE BYTE ARRAY TO UPLOAD IN SASHIDO
            scaledBitmap.compress(Bitmap.CompressFormat.JPEG, 80,
stream);

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        return stream.toByteArray();

    }
```

Código de la aplicación 19

Se trata de una forma muy efectiva de compresión de imágenes, está inspirado en el sistema de compresión de imágenes de WhatsApp, en el que se realiza la mayor compresión posible que permita visualizar en el tamaño de la pantalla la foto sin ninguna pérdida de calidad aparente, por lo que para nuestro caso, se adapta perfectamente a nuestras necesidades y se ha podido encontrar por internet en el portal web de ayuda para programadores, stackoverflow.

4.4 Mapa

En esta pantalla, tendremos el mapa con todos los marcadores que han añadido los usuarios y un botón para filtrar los marcadores que aparecen en pantalla.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    //DECLARATION
    RelativeLayout btAtrasMaps = (RelativeLayout)
findViewById(R.id.btAtrasMaps);
    RelativeLayout btFiltro = (RelativeLayout)
findViewById(R.id.btFiltro);

    //MAP IMPLEMENTATION
    SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
```

```
    mapFragment.getMapAsync(this);

    //CLICK LISTENERS
    btAtrasMaps.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            finish();
        }
    });
    btFiltro.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            showPromptFilter();
        }
    });

}
```

Código de la aplicación 20

En esta pantalla utilizaremos la API de Google Maps para mostrar el mapa, por lo que como vemos en la Activity, es necesario gregar un objeto Fragment a la Activity que administrará el mapa, además de Implementar la interfaz "OnMapReadyCallback" y usar el método de callback "onMapReady(GoogleMap)" para administrar el objeto "GoogleMap", siendo este la representación interna del propio mapa. También se debe llamar a "getMapAsync()" en el fragmento para registrar el callback.

Para configurar las opciones de vista de un mapa, hay que modificar su objeto GoogleMap.

```
public void onMapReady(final GoogleMap map) {

    currentMap = map;

    map.getUiSettings().setZoomControlsEnabled(false);
    map.setInfoWindowAdapter(new CustomInfoWindow());
    map.setOnInfoWindowClickListener(new
GoogleMap.OnInfoWindowClickListener() {
        public void onInfoWindowClick(Marker marker) {

            StringTokenizer tokenArbol = new
StringTokenizer(marker.getTitle());

            //OPEN ARBOL
            Intent intent = new Intent(mContext,
DetalleArbolActivity.class);
            intent.putExtra("MARCADOR_ID", tokenArbol.nextToken());
            startActivity(intent);

        }
    });

    //TODOS LOS ARBOLES
    ParseQuery<ParseObject> query = ParseQuery.getQuery("Marcador");
    query.include("Usuario");
    query.findInBackground(new FindCallback<ParseObject>() {
        public void done(List<ParseObject> arbolesRecibidos,
ParseException e) {

            if (e == null) {
```

```
                CameraPosition cameraPosition = new
CameraPosition.Builder()
                        .target(new LatLng(40.416667, -3.703889))
// Sets the center of the map
                        .zoom(5)
// Sets the zoom
                        .build();
// Creates a CameraPosition from the builder

map.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition
));

                LatLng latLng;
                String arbol, usuario;

                for (int i = 0; i < arbolesRecibidos.size(); i++) {

                    //OBJECT_ID_MARCADOR , FAMILIA , ESPECIE
                    arbol = arbolesRecibidos.get(i).getObjectId() + "
" + arbolesRecibidos.get(i).getString("Familia") + " " +
arbolesRecibidos.get(i).getString("Especie");
                    //OBJECT_ID, USERNAME
                    usuario =
arbolesRecibidos.get(i).getParseUser("Usuario").getObjectId() + " " +
arbolesRecibidos.get(i).getParseUser("Usuario").getUsername();
                    latLng = new
LatLng(arbolesRecibidos.get(i).getParseGeoPoint("Localizacion").getLat
itude(),
arbolesRecibidos.get(i).getParseGeoPoint("Localizacion").getLongitude(
));

                    map.addMarker(new MarkerOptions().position(latLng)
                            .title(arbol)
                            .snippet(usuario)

.icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_marker)));
                }
            }
        }
    });
}
```

Código de la aplicación 21

En este método se encuentra la clave de la pantalla, ya que se gestiona el objeto "GoogleMap", donde se configura el zoom con el que se ve el mapa, la animación de zoom cuando abrimos la pantalla, la búsqueda de todos los marcadores para añadirlos en el mapa (con llamada a Parse) y también se configura el "infoWindow" que es la ventana de información que aparece al pulsar sobre cualquiera de los marcadores del mapa, con algunos detalles del árbol y un botón para ver en profundidad las características del árbol.

También podemos ver parte del código relativo al filtro, en el que en función de lo que seleccione el usuario para mostrar, se van filtrando los marcadores, mostrando sólo los deseados.

```
private void setSpinnerFamilia() {

    progressDialog = new ProgressDialog(MapsActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    final ParseQuery<ParseObject> query =
ParseQuery.getQuery("Arbol");
    query.findInBackground(new FindCallback<ParseObject>() {
        public void done(final List<ParseObject> arbolesRecibidos,
ParseException e) {
            if (e == null) {

                List<String> listaFamilias = new ArrayList<String>();

                for (int i = 0; i < arbolesRecibidos.size(); i++) {

listaFamilias.add(arbolesRecibidos.get(i).getString("Familia"));;
                }

                ArrayAdapter<String> spinnerArrayAdapter = new
ArrayAdapter<String>(mContext, android.R.layout.simple_spinner_item,
listaFamilias);

spinnerArrayAdapter.setDropDownViewResource(android.R.layout.simple_sp
inner_dropdown_item); // The drop down vieww
                spinnerFamilia.setAdapter(spinnerArrayAdapter);

                progressDialog.dismiss();

            } else {

                progressDialog.dismiss();
                Log.e(TAG, "setSpinnerFamilia: " + e.getMessage());
            }
        }
    });
}
```

Código de la aplicación 22

En este primer método, se crea una query dinámica en función de la familia que ha seleccionado el usuario, mostrando en el mapa todos los marcadores de ese tipo.

```
private void updateEspecie(String familia){

    ParseQuery query = ParseQuery.getQuery("Arbol");
    query.whereEqualTo("Familia", familia);
    query.getFirstInBackground(new GetCallback<ParseObject>() {
        public void done(ParseObject arbolRecibido, ParseException e)
{
            if (e == null) {

                List<String> listaEspecies = new ArrayList<String>();

                listaEspecies =  arbolRecibido.getList("Especie");

                ArrayAdapter<String> spinnerArrayAdapter = new
```

```
ArrayAdapter<String>(mContext, android.R.layout.simple_spinner_item,
listaEspecies);

spinnerArrayAdapter.setDropDownViewResource(android.R.layout.simple_sp
inner_dropdown_item); // The drop down vieww
                spinnerEspecie.setAdapter(spinnerArrayAdapter);

            } else {

                Log.e(TAG, "setSpinnerEspecie: " + e.getMessage());

            }
        }
    });
}
```

Código de la aplicación 23

En este segundo método para el filtrado, se crea otra query dinámica en función de la anterior, que nos da la lista de especies disponibles para la familia que se ha seleccionado anteriormente.

```
private void RefreshMap() {

    currentMap.clear();

    //ARBOLES FILTRADOS
    ParseQuery<ParseObject> queryFilter =
ParseQuery.getQuery("Marcador");
    queryFilter.whereEqualTo("Familia", itemFamilia);
    queryFilter.include("Usuario");

    if(especieSeleccionada) {
        queryFilter.whereEqualTo("Especie", itemEspecie);
    }

    queryFilter.findInBackground(new FindCallback<ParseObject>() {
        public void done(List<ParseObject> arbolesRecibidos,
ParseException e) {
            if (e == null && !arbolesRecibidos.isEmpty()) {

                LatLng latLng;
                String arbol, usuario;

                for (int i = 0; i < arbolesRecibidos.size(); i++) {
                    //OBJECT_ID_MARCADOR , FAMILIA , ESPECIE
                    arbol = arbolesRecibidos.get(i).getObjectId() + "
" + arbolesRecibidos.get(i).getString("Familia") + " " +
arbolesRecibidos.get(i).getString("Especie");
                    Log.e(TAG, arbol);
                    //OBJECT ID, USERNAME
                    usuario =
arbolesRecibidos.get(i).getParseUser("Usuario").getObjectId() + " " +
arbolesRecibidos.get(i).getParseUser("Usuario").getUsername();
                    latLng = new
LatLng(arbolesRecibidos.get(i).getParseGeoPoint("Localizacion").getLat
itude(),
arbolesRecibidos.get(i).getParseGeoPoint("Localizacion").getLongitude(
));
```

```
                    currentMap.addMarker(new
MarkerOptions().position(latLng)
                            .title(arbol)
                            .snippet(usuario)

.icon(BitmapDescriptorFactory.fromResource(R.mipmap.ic_marker)));
                }
            } else if (e != null){
                e.printStackTrace();
            }
        }
    });
}
```

Código de la aplicación 24

Una vez hecho el filtrado, procedemos a realizar la query y mostrar los árboles filtrados, siendo igual al método "onMapReady", sólo que en ese se mostraban todos, y aquí, solo los que han pasado el filtro.

4.4.1 Añadir árboles al mapa

En esta pantalla, tenemos dos desplegables para seleccionar la familia y la especie del árbol que vamos a introducir, también una descripción que podemos añadir y una fotografía.

```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_anadir);

    //DECLARATIONS
    RelativeLayout btAtrasAñadir = (RelativeLayout)
findViewById(R.id.btAtrasAñadir);

    spinnerFamilia = (Spinner) findViewById(R.id.spinnerFamilia);
    spinnerEspecie = (Spinner) findViewById(R.id.spinnerEspecie);

    etDescripcionArbol = (EditText)
findViewById(R.id.etDescripcionArbol);

    relativeContainerPhoto = (RelativeLayout)
findViewById(R.id.relativeContainerPhoto);
    btFoto = (ImageView) findViewById(R.id.btFoto);

    Button btSiguienteAñadir = (Button)
findViewById(R.id.btSiguienteAñadir);

    //IMPLEMENTATION
    progressDialog = new ProgressDialog(AnadirActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    btAtrasAñadir.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            finish();
        }
```

```
    });

    relativeContainerPhoto.setOnClickListener(new
View.OnClickListener() {
        public void onClick(View v) {
            clickImage = true;

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                if
(!Settings.System.canWrite(getApplicationContext())){
                    requestPermissions(new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
Manifest.permission.READ_EXTERNAL_STORAGE}, 2909);
                    selectImage();
                }
                else {
                    selectImage();
                }
            } else {
                selectImage();
            }

        }
    });

    setSpinnerFamilia();

    spinnerFamilia.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> parentView, View
selectedItemView, int myPosition, long myID) {

            String item = (String)
parentView.getItemAtPosition(myPosition);

            progressDialog = new ProgressDialog(AnadirActivity.this);
            progressDialog.setIndeterminate(true);
            progressDialog.setMessage(getString(R.string.cargando));
            progressDialog.setCancelable(false);
            progressDialog.show();

            updateEspecie(item);

        }

        @Override
        public void onNothingSelected(AdapterView<?> parentView) { }

    });

    btSiguienteAñadir.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            subirArbol();
        }
    });

}
```

Código de la aplicación 25

Podemos ver que al igual que en el filtro del Mapa, también tenemos un método que actualiza la lista de las especies en función de la familia que se ha seleccionado, además de la estructura de la pantalla, con los "spinner", que son las listas desplegables.

```
private void subirArbol() {

    progressDialog = new ProgressDialog(AnadirActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.subiendoArbol));
    progressDialog.setCancelable(false);
    progressDialog.show();

    if(clickImage) {

        final ParseFile imageParse = new ParseFile(imageBytes);

        imageParse.saveInBackground(new SaveCallback() {
            public void done(ParseException e) {
                if (null == e) {

                    if
(spinnerEspecie.getSelectedItem().toString().equals("Otra")) {
                        if (!validate()) {
                            return;
                        }
                    }

                    final ParseObject marcador = new
ParseObject("Marcador");

if(!etDescripcionArbol.getText().toString().isEmpty()) {
                            marcador.put("Descripcion",
etDescripcionArbol.getText().toString());
                    }
                    marcador.put("Especie",
spinnerEspecie.getSelectedItem().toString());
                    marcador.put("Familia",
spinnerFamilia.getSelectedItem().toString());
                    marcador.put("Foto", imageParse);
                    marcador.put("Usuario",
ParseUser.getCurrentUser());

                    marcador.saveInBackground(new SaveCallback() {
                        public void done(ParseException e) {

                            if(e == null){
                                Intent intent = new Intent(mContext,
SetLocationActivity.class);
                                intent.putExtra("ARBOL_ID",
marcador.getObjectId());
                                startActivityForResult(intent,
REQUEST_SET_LOCATION);

                                progressDialog.dismiss();

                            } else {
                                progressDialog.dismiss();
                                Log.e(TAG, "error:" + e.toString());
                                Toast.makeText(getBaseContext(), "Lo
```

```
siento, ha habido un error: " + e.getMessage(),
Toast.LENGTH_LONG).show();
                                }
                            }
                        });
                } else {

                    Toast.makeText(getBaseContext(), "Lo siento, ha
habido un error: " + e.getMessage(), Toast.LENGTH_LONG).show();
                    progressDialog.dismiss();

                }
            }
        });
    } else {

        if
(spinnerEspecie.getSelectedItem().toString().equals("Otra")) {
            if (!validate()) {
                return;
            }
        }

        final ParseObject marcador = new ParseObject("Marcador");
        if(!etDescripcionArbol.getText().toString().isEmpty()) {
            marcador.put("Descripcion",
etDescripcionArbol.getText().toString());
        }
        marcador.put("Especie",
spinnerEspecie.getSelectedItem().toString());
        marcador.put("Familia",
spinnerFamilia.getSelectedItem().toString());
        marcador.put("Usuario", ParseUser.getCurrentUser());

        marcador.saveInBackground(new SaveCallback() {
            public void done(ParseException e) {

                if(e == null){

                    Intent intent = new Intent(mContext,
SetLocationActivity.class);
                    intent.putExtra("ARBOL_ID",
marcador.getObjectId());
                    startActivityForResult(intent,
REQUEST_SET_LOCATION);

                    progressDialog.dismiss();

                } else {

                    progressDialog.dismiss();
                    Log.e(TAG, "error:" + e.toString());
                    Toast.makeText(getBaseContext(), "Lo siento, ha
habido un error: " + e.getMessage(), Toast.LENGTH_LONG).show();
                }
            }
        });
    }
}
```

Código de la aplicación 26

Con este método añadimos a la base de datos un nuevo marcador, llamado así de forma técnica, porque al fin y al cabo será un marcador en el mapa, pero representa a un árbol. Podemos ver que creamos el objeto de Parse "marcador" y le vamos añadiendo los diferentes datos que darán forma a este objeto.

Podemos ver alguna comprobación extra como por ejemplo el caso de que si el usuario selecciona la especie "Otra", la descripción pasa a ser obligatoria, ya que si no, faltarían datos.

El objeto "marcador" será el que se suba a la base de datos a falta de la localización, que se hace en el siguiente paso.

```
private void setLocationToTree() {

    progressDialog = new ProgressDialog(SetLocationActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.localizandoArbol));
    progressDialog.setCancelable(false);
    progressDialog.show();

    ParseQuery queryMarcador = ParseQuery.getQuery("Marcador");
    queryMarcador.whereEqualTo("objectId",
intentReceived.getStringExtra("ARBOL_ID"));
    queryMarcador.getFirstInBackground(new GetCallback<ParseObject>()
{
        public void done(ParseObject arbolRecibido, ParseException e)
{
            if (e == null) {

                arbolRecibido.put("Localizacion", new
ParseGeoPoint(latlng.latitude, latlng.longitude));
                arbolRecibido.saveInBackground(new SaveCallback() {
                    public void done(ParseException e) {
                        if (e == null) {

                            Intent intentReceived = getIntent();
                            setResult(RESULT_OK, intentReceived);
                            finish();

                            Toast.makeText(getBaseContext(), "¡Árbol
añadido con éxito!", Toast.LENGTH_LONG).show();

                            progressDialog.dismiss();

                        } else {

                            progressDialog.dismiss();
                            Toast.makeText(getBaseContext(), "Lo
siento, ocurrió un error. Inténtalo de nuevo",
Toast.LENGTH_LONG).show();
                            Log.e(TAG, "setLocation " +
e.getMessage());

                        }
                    }
                });
```

```
            } else {
                Toast.makeText(getBaseContext(), "Lo siento, ocurrió
un error. Inténtalo de nuevo", Toast.LENGTH_LONG).show();
                progressDialog.dismiss();
                Log.e(TAG, "setSpinnerEspecie: " + e.getMessage());

            }
        }
    });

}
```

Código de la aplicación 27

Como vemos, recoge el objeto y le añade la ubicación del lugar en el que se encuentra, ya que en este paso se carga el mapa y podemos mover el marcador donde deseemos.

El resto de métodos de esta ventana, son de la API de Google, y los utilizamos para buscar ubicaciones y movernos hasta ese sitio, poder poner el marcador en el mapa simplemente pulsando donde queramos, etc.

4.5 Chat

Para que funcione el chat tenemos dos Activitys que serán las encargadas, por un lado de crear las conversaciones, gestionarlas y por otro el de gestionar los mensajes para cada conversación.

4.5.1 MainActivity

En primer lugar tenemos el método “onCreate” que se llama cada vez que se crea la pantalla de Chats y donde se declaran todos los elementos que necesitaremos para la lista de conversaciones que verá cada usuario.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chat_main);

    //DECLARATION
    RelativeLayout goBackChatMain = (RelativeLayout)
findViewById(R.id.goBackChatMain);
    RelativeLayout emptyRelative = (RelativeLayout)
findViewById(R.id.empty);
    chatListView = (ListView) findViewById(R.id.chatListView);

    //IMPLEMENTATION
    chatListView.setEmptyView(emptyRelative);
    adapter = new ListChatMainAdapter(mContext, chatConversations);

    progressDialog = new ProgressDialog(ChatMainActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
```

```
    progressDialog.show();

    DownloadChats();

    goBackChatMain.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            finish();
        }
    });

    chatListView.setOnItemClickListener(this);
}
```

Código de la aplicación 28

Además de eso, también se encarga de descargar los chats, y cuando no existe ninguna conversación disponible para el usuario, llama a "chatListView.setEmptyView(emptyRelative)" que se encarga de poner en la pantalla el mensaje : "No has iniciado ninguna conversación".

A continuación tenemos el método "onItemClick" que es el adaptador de clicks de la lista.

```
public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {

    //SET NOTIFICATION GONE
    ChatMainObject chatSelection = chatConversations.get(position);
    chatSelection.setMessagesNotRead(false);
    adapter.notifyDataSetChanged();

    //OPEN CHAT
    Intent intent = new Intent(this, ChatConversationActivity.class);
    intent.putExtra("ID_INTERLOCUTOR",
chatSelection.getinterlocutorId());
    intent.putExtra("NAME_INTERLOCUTOR",
chatSelection.getinterlocutorName());
    intent.putExtra("ID_MARCADOR", chatSelection.getmarcadorId());
    intent.putExtra("NAME_MARCADOR", chatSelection.getmarcadorName());
    startActivityForResult(intent, REQUEST_CHAT_CONVERSATION);

}
```

Código de la aplicación 29

Cuando pulsas sobre un elemento de la lista de los Chats, se ejecuta, llama a la ventana de la conversación, enviando todos los datos que necesita (nombre, marcador, etc).

El método "DownloadChats" es llamado por el "onCreate" y como su nombre indica, se encarga de descargar los mensajes.

```
private void DownloadChats() {

        ParseQuery myQuery1 = new ParseQuery("ChatConversaciones");
        myQuery1.whereEqualTo("propietario", currentUser);

        ParseQuery myQuery2 = new ParseQuery("ChatConversaciones");
        myQuery1.whereEqualTo("interesado", currentUser);
```

```
        List<ParseQuery<ParseObject>> queries = new ArrayList<>();
        queries.add(myQuery1);
        queries.add(myQuery2);

        ParseQuery<ParseObject> mainQuery = ParseQuery.or(queries);

//
        mainQuery.orderByDescending("createdAt");
        mainQuery.include("marcador");
        mainQuery.include("interesado");
        mainQuery.include("propietario");
        mainQuery.findInBackground(new FindCallback<ParseObject>() {
            public void done(final List<ParseObject>
chatConversationsReceived, ParseException e) {
                if (e == null && !chatConversationsReceived.isEmpty())
{

                    numConversations =
chatConversationsReceived.size();

                    for (int i = 0; i <
chatConversationsReceived.size(); i++) {


if(chatConversationsReceived.get(i).getParseUser("propietario").getObj
ectId().equals(currentUser.getObjectId())) {
                            getHours(chatConversationsReceived.get(i),
true);
                        } else {
                            getHours(chatConversationsReceived.get(i),
false);
                        }
                    }

                } else if (e != null){
                    Log.e(TAG, "DownloadChats: " + e.getMessage());
                    progressDialog.dismiss();
                } else {
                    progressDialog.dismiss();
                }
            }
        });
    }
```

Código de la aplicación 30

Para ello, se hace una query que comprueba el currentUser (el usuario que ha iniciado sesión) de cada usuario y descarga todos los mensajes en los que aparezca, bien como emisor o como receptor del mensaje.

Posteriormente, tenemos el método "`getUltimoMensaje`" que se encarga de buscar el último mensaje de las conversaciones.

```
private void getUltimoMensaje(final ParseObject chatConversation,
final boolean esMiMarcador){

        ArrayList<ParseObject> whereClausure = new ArrayList<>();
```

```
        final ParseQuery<ParseObject> query =
ParseQuery.getQuery("ChatMensajes");
//        query.selectKeys(Arrays.asList("text", "timestamp",
"recipient.objectId", "read"));

        if(esMiMarcador) {

            whereClausure.add(currentUser);

whereClausure.add(chatConversation.getParseObject("interesado"));
            query.whereContainedIn("sender", whereClausure);

            whereClausure.clear();

whereClausure.add(chatConversation.getParseObject("interesado"));
            whereClausure.add(currentUser);
            query.whereContainedIn("recipient", whereClausure);

        } else {

            whereClausure.add(currentUser);

whereClausure.add(chatConversation.getParseObject("propietario"));
            query.whereContainedIn("sender", whereClausure);

            whereClausure.clear();

whereClausure.add(chatConversation.getParseObject("propietario"));
            whereClausure.add(currentUser);
            query.whereContainedIn("recipient", whereClausure);

        }

        query.whereEqualTo("marcador",
chatConversation.getParseObject("marcador"));
        query.orderByDescending("createdAt");

        query.getFirstInBackground(new GetCallback<ParseObject>() {
            @Override
            public void done(final ParseObject firstChatMessage,
ParseException e) {

                if (e == null && firstChatMessage != null) {


if(firstChatMessage.getParseObject("recipient").getObjectId().equals(c
urrentUser.getObjectId())  && !firstChatMessage.getBoolean("read")) {

                        if(esMiMarcador) {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("interesado").getObjectId(),

chatConversation.getParseObject("interesado").getString("username"),

chatConversation.getParseObject("interesado").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),

chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),
```

```
firstChatMessage.getString("texto"),

getFecha(firstChatMessage.getDate("timestamp")),
                                    true);
                        } else {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("propietario").getObjectId(),

chatConversation.getParseObject("propietario").getString("username"),

chatConversation.getParseObject("propietario").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),

chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),

firstChatMessage.getString("texto"),

getFecha(firstChatMessage.getDate("timestamp")),
                                    true);
                        }

                        chatConversations.add(chatObject);

                        if(chatConversations.size() ==
numConversations) {
                            orderConversations();
                        }
                    } else {

                        if(esMiMarcador) {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("interesado").getObjectId(),

chatConversation.getParseObject("interesado").getString("username"),

chatConversation.getParseObject("interesado").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),

chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),

firstChatMessage.getString("texto"),

getFecha(firstChatMessage.getDate("timestamp")),
                                    false);
                        } else {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("propietario").getObjectId(),

chatConversation.getParseObject("propietario").getString("username"),

chatConversation.getParseObject("propietario").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),
```

```
chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),

firstChatMessage.getString("texto"),

getFecha(firstChatMessage.getDate("timestamp")),
                                        false);
                            }

                            chatConversations.add(chatObject);

                            if(chatConversations.size() ==
numConversations) {
                                orderConversations();
                            }
                        }
                    } else {

                        if(esMiMarcador) {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("interesado").getObjectId(),

chatConversation.getParseObject("interesado").getString("username"),

chatConversation.getParseObject("interesado").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),

chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),
                                        "Chat vacío...",

getFecha(firstChatMessage.getDate("timestamp")),
                                        false);
                        } else {
                            chatObject = new ChatMainObject(

chatConversation.getParseObject("propietario").getObjectId(),

chatConversation.getParseObject("propietario").getString("username"),

chatConversation.getParseObject("propietario").getParseFile("Imagen"),

chatConversation.getParseObject("marcador").getObjectId(),

chatConversation.getParseObject("marcador").getString("Familia") + ",
" + chatConversation.getParseObject("marcador").getString("Especie"),
                                        "Chat vacío...",

getFecha(firstChatMessage.getDate("timestamp")),
                                        false);

                        }
                        chatConversations.add(chatObject);

                        if(chatConversations.size() == numConversations) {
                            orderConversations();
                        }
```

```
                }
            }
        });
    }
```

Código de la aplicación 31

Se trata de un método bastante extenso y complejo ya que se crea una query, que para cada conversación, localiza el último mensaje para ponerlo en la lista de chats de cada una de ellas, a modo de previsualización, y también gestiona el símbolo de mensajes sin leer, utilizando para ello el boolean "read".

Además de todo esto, también se encarga de ordenar las conversaciones según el último mensaje recibido en cada una de ellas.

Por último tenemos el método "onActivityResult" que gestiona la salida de un chat.

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

    if (requestCode == REQUEST_CHAT_CONVERSATION) {
        if (resultCode == RESULT_OK) {


chatConversations.get(data.getExtras().getInt("ADAPTER_POSITION")).set
HourMessage(data.getExtras().getString("HOUR_LAST_MESSAGE"));

chatConversations.get(data.getExtras().getInt("ADAPTER_POSITION")).set
MessageLast(data.getExtras().getString("LAST_MESSAGE"));
            adapter.notifyDataSetChanged();

        }
    }
}
```

Código de la aplicación 32

Cuando sales de una conversación y vuelves a la lista, es el método que se encarga de actualizar el último mensaje de cada una de ellas, que aparece visible por información.

4.5.2 ConversationActivity

Al igual que en el MainActivity, primero vamos a hablar del método "onCreate", que vemos a continuación.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_chat_conversation);

    //DECLARATION
    RelativeLayout relativeBackChat = (RelativeLayout)
```

```
findViewById(R.id.relativeBackChat);

    TextView tvNameChat = (TextView) findViewById(R.id.tvNameChat);
    TextView tvNameMarcador = (TextView)
findViewById(R.id.tvNameMarcador);

    lvChat = (ListView) findViewById(R.id.message_container);
    etMessage = (EditText) findViewById(R.id.text_send);

    RelativeLayout btSend = (RelativeLayout)
findViewById(R.id.btSend);
    final ImageView imageButtonSend = (ImageView)
findViewById(R.id.btn_send);

    //IMPLEMENTATION
    progressDialog = new
ProgressDialog(ChatConversationActivity.this);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage(getString(R.string.cargando));
    progressDialog.setCancelable(false);
    progressDialog.show();

    //Getting data from ChatMainActivity
    receivedData = getIntent();

    //SINCH
    bindService(new Intent(this, SinchService.class),
serviceConnection, BIND_AUTO_CREATE);
    checkSinchConnectionStatus();

    //Inicializar el adaptador con la fuente de datos
    adapter = new ListChatMessagesAdapter(this, chatMessagesList);
    DownloadChat();

    //Set name

tvNameChat.setText(receivedData.getStringExtra("NAME_INTERLOCUTOR"));

tvNameMarcador.setText(receivedData.getStringExtra("NAME_MARCADOR"));

    //Button listeners
    btSend.setOnClickListener(new SendButton());
    relativeBackChat.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {

            Intent intentReceived = getIntent();
            intentReceived.putExtra("ADAPTER_POSITION",
getIntent().getExtras().getInt("ADAPTER_POSITION"));
            intentReceived.putExtra("LAST_MESSAGE",
chatMessagesList.get(chatMessagesList.size() - 1).getString("texto"));
            intentReceived.putExtra("HOUR_LAST_MESSAGE",
getFecha(chatMessagesList.get(chatMessagesList.size() -
1).getDate("timestamp")));
            setResult(RESULT_OK, intentReceived);
            finish();

        }
    });

    TextWatcher textWatcherTitle = new TextWatcher() {
        public void afterTextChanged(Editable s) {
```

```
            //Check if there are a title
            if(etMessage.getText().length() > 0){

imageButtonSend.setColorFilter(getApplicationContext().getResources().
getColor(R.color.treeMapColor),
android.graphics.PorterDuff.Mode.SRC_IN);
            }else{

imageButtonSend.setColorFilter(getApplicationContext().getResources().
getColor(R.color.greyMidium),
android.graphics.PorterDuff.Mode.SRC_IN);
            }
        }
        public void beforeTextChanged(CharSequence s, int start, int
count, int after) { }
        public void onTextChanged(CharSequence s, int start, int
before, int count) {
        }
    };
    etMessage.addTextChangedListener(textWatcherTitle);
}
```

Código de la aplicación 33

De este método se pueden comentar varias cosas interesantes, como puede ser "receivedData", que es un intent que se encarga de recibir todos los datos que vienen del MainActivity (exactamente del método "onItemClick").

Por otro lado tenemos el "bindService", que activa Sinch (del que hablaremos a continuación), que hace un bind de la conversación para que reciba a tiempo real los mensajes, sin esto, el chat no funcionaría a tiempo real, por lo que funcionaría simplemente como un foro.

Después se declara un adapter que se relaciona con la lista "chatMessageList", que posteriormente el setAdapter se encargará de rellenar, añadiendo tantas filas como mensajes en la conversación.

Por último en este completo método, tenemos "textWatcherTitle" que hace que la flecha para enviar un mensaje se ponga en rojo cuando hay algo para enviar.

En esta Activity también tenemos otro método "DownloadChats", parecido al otro.

```
private void DownloadChat() {

        final ParseQuery<ParseObject> query =
ParseQuery.getQuery("ChatMensajes");

        ArrayList<ParseObject> whereClausure = new ArrayList<>();
        whereClausure.add(currentUser);
        whereClausure.add(ParseObject.createWithoutData("_User",
receivedData.getStringExtra("ID_INTERLOCUTOR")));
```

```
        query.whereContainedIn("sender", whereClausure);

        whereClausure.clear();
        whereClausure.add(ParseObject.createWithoutData("_User",
receivedData.getStringExtra("ID_INTERLOCUTOR")));
        whereClausure.add(currentUser);
        query.whereContainedIn("recipient", whereClausure);

        query.whereEqualTo("marcador",
ParseObject.createWithoutData("Marcador",
receivedData.getStringExtra("ID_MARCADOR")));
        query.orderByAscending("createdAt");

        query.findInBackground(new FindCallback<ParseObject>() {
            public void done(List<ParseObject> messagesReceived,
ParseException e) {
                if (e == null && chatMessagesList != null) {

                    for (int i = 0; i < messagesReceived.size(); i++)
{

                        chatMessagesList.add(messagesReceived.get(i));

                        if(!messagesReceived.get(i).getBoolean("read")
&&
messagesReceived.get(i).getParseObject("recipient").getObjectId().equa
ls(currentUser.getObjectId())) {
                            messagesReceived.get(i).put("read", true);

messagesReceived.get(i).saveInBackground();
                        }

                    }
                    lvChat.setAdapter(adapter);
                    progressDialog.dismiss();

                } else if (e != null){
                    Log.e(TAG, "dataReceived:Error: " +
e.getMessage());
                    progressDialog.dismiss();
                }
            }
        });
    }
```

Código de la aplicación 34

Donde se hace la query para descargar de la tabla ChatMensajes (de la base de datos) todos los mensajes que pertenecen a la conversación, generando el chatList y añadiéndolos al adaptador.

Posteriormente tenemos el método “onIncomingMessage” para los mensajes recibidos gracias a Sinch.

```
public void onIncomingMessage(MessageClient client, Message message) {

    Log.e(TAG, message.getTextBody());
```

```
    ParseObject chatObject = new ParseObject("ChatMensajes");
    chatObject.put("texto", message.getTextBody());
    chatObject.put("recipient", currentUser);
    chatObject.put("sender", ParseObject.createWithoutData("_User",
receivedData.getStringExtra("ID_INTERLOCUTOR")));
    chatObject.put("marcador",
ParseObject.createWithoutData("Marcador",
receivedData.getStringExtra("ID_MARCADOR")));
    chatObject.put("read", true);
    chatObject.put("timestamp", Calendar.getInstance().getTime());

    chatMessagesList.add(chatObject);
    adapter.notifyDataSetChanged();

}
```

Código de la aplicación 35

En este metodo se crea un objeto de Parse, cogiendo de Sinch el “message.getTextBody()” que es el contenido del mensaje en sí, y a partir de este, se genera un parseObject para añadirlo a la lista de los chats, y con “adapter.notifyDataSetChanged()” el adaptador coge lo ultimo que hay en la lista y lo muestra.

Por último, podemos hablar del método “sendButton”, que como su propio nombre indica, se encarga de gestionar la acción que provoca el botón de enviar un mensaje, creando para ello un parseObject con el mensaje y guardándolo en la base de datos.

```
private class SendButton implements View.OnClickListener{
    @Override
    public void onClick(View v) {

        final String text = etMessage.getText().toString();
        if (text.isEmpty())
            return;


messageService.sendMessage(receivedData.getStringExtra("ID_INTERLOCUTO
R"), text);

        //Creating ParseObject to save in DDBB.
        final ParseObject chatObject = new
ParseObject("ChatMensajes");
        chatObject.put("texto", text);
        chatObject.put("sender", currentUser);
        chatObject.put("read", false);
        chatObject.put("recipient",
ParseObject.createWithoutData("_User",
receivedData.getStringExtra("ID_INTERLOCUTOR")));
        chatObject.put("marcador",
ParseObject.createWithoutData("Marcador",
receivedData.getStringExtra("ID_MARCADOR")));
        chatObject.put("timestamp", Calendar.getInstance().getTime());
```

```
        chatObject.saveInBackground(new SaveCallback() {
            @Override
            public void done(ParseException e) {
                if(e == null){

                    chatMessagesList.add(chatObject);
                    adapter.notifyDataSetChanged();

                    ParseQuery query = ParseInstallation.getQuery();
                    query.whereEqualTo("user",
ParseObject.createWithoutData("_User",
receivedData.getStringExtra("ID_INTERLOCUTOR")));

                    ParsePush push = new ParsePush();
                    push.setQuery(query);

                    JSONObject data = new JSONObject();
                    try {
                        data.put("alert", currentUser.getUsername() +
":\n" + text);
                        data.put("title", "TreeMap");
                        data.put("badge", "Increment");
                        data.put("sound", "default");
                        data.put("type", "message");
                    } catch (JSONException je) {
                        je.printStackTrace();
                    }
                    push.setData(data);
                    push.sendInBackground();
                }
                else{
                    Log.e(TAG, e.getMessage());
                }
            }
        });
        etMessage.setText("");
    }
}
```

Código de la aplicación 36

Se puede decir que estos son los métodos que he creído más interesantes y/o importantes de estas Activitys.

4.5.3 Sinch

Para poder utilizar Sinch en el chat de nuestro proyecto, además de llamarlo desde la clase del chat es necesario crear un paquete con una clase de configuración para la autorización del servicio, al igual que para conectar con Parse.

Por tanto tenemos el paquete “sinchHelper”, y dentro la clase “SinchService” con varios métodos que vienen ya creados para diversas tareas, como para verificar el estado del servicio, si el cliente está iniciado, etc.

Entre ellos cabe destacar el método "startSinchClient" en el que configuramos el servicio, como hemos dicho antes.

```
public void startSinchClient(String username) {
    sinchClient = Sinch.getSinchClientBuilder()
            .context(this)
            .userId(username)
            .applicationKey(APP_KEY)
            .applicationSecret(APP_SECRET)
            .environmentHost(ENVIRONMENT)
            .build();

    //this client listener requires that you define
    //a few methods below
    sinchClient.addSinchClientListener(this);

    //messaging is "turned-on", but calling is not
    sinchClient.setSupportMessaging(true);
    sinchClient.setSupportActiveConnectionInBackground(true);

    sinchClient.checkManifest();
    sinchClient.start();
}
```

Código de la aplicación 37

Además de la configuración (APP_KEY y APP_SECRET) que se encuentra oculta por seguridad, se arranca Sinch, pero antes crea un "ClientListener".

4.6 Dando estilo a las pantallas con XML

En este apartado podemos ver el ejemplo de una pantalla de la aplicación, en este caso es la de perfil, que es bastante completa y tiene bastantes cosas interesantes, como puede ser la foto, los campos de texto o algún botón, así como varios elementos de organización del espacio.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".perfil.PerfilUsuarioActivity"
    android:background="@android:color/white">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="56dp"
        android:orientation="horizontal"
        android:gravity="center_vertical"
        >

        <RelativeLayout
```

```
        android:layout_width="56dp"
        android:layout_height="match_parent"
        android:id="@+id/btAtrasPerfil"
        android:gravity="center"
        >

        <ImageView
            android:layout_width="22dp"
            android:layout_height="22dp"
            android:background="@drawable/ic_atras"
            />

    </RelativeLayout>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/perfilUsuario"
        android:textSize="16sp"
        android:textStyle="bold"
        android:textColor="@color/treeMapBlack"
        android:layout_marginLeft="16dp"
        />

    <android.support.v4.widget.Space
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        />

    <RelativeLayout
        android:id="@+id/btEditarPerfil"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:gravity="center"
        >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Editar"
            android:textSize="15sp"
            android:textColor="@color/treeMapBlack" />
    </RelativeLayout>

    <RelativeLayout
        android:visibility="gone"
        android:id="@+id/btGuardarPerfil"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:padding="16dp"
        android:gravity="center"
        >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Save"
            android:textSize="15sp"
            android:textColor="@color/treeMapBlack"
```

```
            />

    </RelativeLayout>
</LinearLayout>

<View
    android:layout_width="match_parent"
    android:layout_height="0.5dp"
    android:background="@color/greyMidium"
    />

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true"
    >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="32dp"
        android:orientation="vertical"
        android:gravity="center_horizontal"
        >

        <LinearLayout
            android:id="@+id/btCambiarContraseña"
            android:layout_width="match_parent"
            android:layout_height="36dp"
            android:orientation="horizontal"
            android:gravity="center"
            >

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/modificarContraseña"
                android:textColor="@color/greyMidium"
                android:textStyle="bold"
                android:textSize="16sp"
                />

            <ImageView
                android:layout_width="12dp"
                android:layout_height="12dp"
                android:src="@drawable/ic_flecha"
                android:tint="@color/greyMidium"
                android:layout_marginLeft="6dp"
                />

        </LinearLayout>

        <tfm.treemap.es.treemap.entities.RoundedImageView
            android:id="@+id/imagenPerfil"
            android:layout_width="80dp"
            android:layout_height="80dp"
            android:layout_marginTop="24dp"
            android:scaleType="centerInside"
            />

        <LinearLayout
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:layout_marginTop="24dp"
>

<TextView
    android:id="@+id/tvNombreApellidos"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/nombreApellido"
    android:layout_marginTop="30dp"
    android:textColor="@color/greyMidium"
    android:textSize="14sp"
    />

<EditText
    android:id="@+id/etNombreApellidos"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/treeMapBlack"
    android:maxLines="1"
    android:background="@null"
    android:textSize="18sp"
    android:layout_marginTop="6dp"
    />

<View
    android:layout_width="match_parent"
    android:layout_height="0.5dp"
    android:background="@color/greyMidium"
    android:layout_marginTop="16dp"
    />

<TextView
    android:id="@+id/tvNombreUsuario"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/nombreUsuario"
    android:layout_marginTop="30dp"
    android:textColor="@color/greyMidium"
    android:textSize="14sp"
    />

<EditText
    android:id="@+id/etNombreUsuario"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/treeMapBlack"
    android:maxLines="1"
    android:background="@null"
    android:textSize="18sp"
    android:layout_marginTop="6dp"
    />

<View
    android:layout_width="match_parent"
    android:layout_height="0.5dp"
    android:background="@color/greyMidium"
    android:layout_marginTop="16dp"
    />
```

```
                <TextView
                    android:id="@+id/tvEmail"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/email"
                    android:textColor="@color/greyMidium"
                    android:textSize="14sp"
                    android:layout_marginTop="12dp"
                    />

                <EditText
                    android:id="@+id/etEmail"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:inputType="textEmailAddress"
                    android:maxLines="1"
                    android:background="@null"
                    android:textSize="18sp"
                    android:layout_marginTop="6dp"
                    android:textColor="@color/treeMapBlack"
                    />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="0.5dp"
                    android:background="@color/greyMidium"
                    android:layout_marginTop="16dp"
                    />

            </LinearLayout>
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

Código de la aplicación 38

En estas algo más de 200 líneas de código se define mediante código XML el diseño de una pantalla, que globalmente la separamos en “cajas” para organizar el espacio.

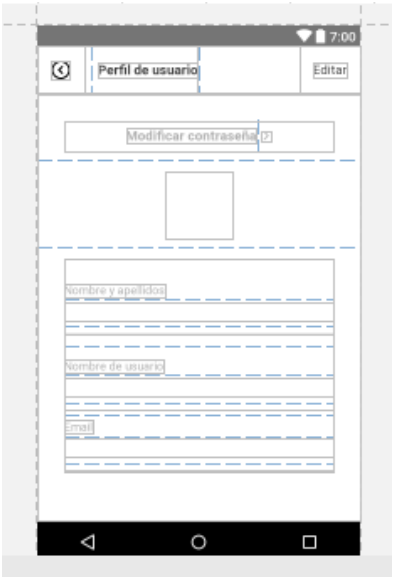

Figura 38: *Diseño pantalla “Perfil de usuario”*

En primer lugar tenemos los LinearLayout, que se trata de una especie de caja o contenedor lineal (uno a continuación del otro), que puede tener tanto orientación vertical como horizontal, que hace que se vayan añadiendo el contenido uno detrás de otro hacia abajo o de derecha a izquierda.

Otros elementos que se sitúan en zonas más especiales de la pantalla, como puede ser el botón hacia atrás o el icono de la cámara de fotos, se sitúan en un RelativeLayout, cuya principal característica es que los widgets que estén dentro de este contenedor basarán su posición en relación con los otros elementos, por ejemplo, podemos definir que el widget X quede debajo del widget Y, y que a su vez éste se alinee verticalmente con el widget Z.

El android:layout_width="match_parent" hace a su vez que el ancho del contenido se adapte al ancho de la pantalla, mientras que el android:layout_height="wrap_content" hace que se adapte la altura a los elementos que hay dentro del contenedor.

Por otra parte, los Text View nos permiten añadir texto sobre el layout, por ejemplo, en esta pantalla tenemos el título que se añade con un android:text="@string/perfilUsuario", siendo entre comillas el string que se mostrará.

Por otra parte, ScrollView permitirá que el usuario pueda desplazar con el dedo la interfaz creada (hacer scroll).

Por último, algunos elementos que no se encuentran en esta pantalla en especial pero si en el proyecto son android:inputType=“textPassword" que hace que no se muestre el texto plano sino los puntos de las contraseñas, android:hint=“@string/nombreApellido" que nos permite dentro de un editText ver el mensaje que aparece antes de pinchar y android:inputType=“textEmailAddress" hace que cambie el tipo de teclado para que aparezca la @, .com etc

5. RESULTADOS

5.1 Introducción

En este apartado se mostrarán y describirán las utilidades desarrolladas para la plataforma web, la función de las diferentes pestañas y una breve descripción de cada una, que además pueden servir de documentación para la plataforma, ya que se explica detalladamente, entre otras cosas, los pasos para conseguir registrar un árbol con éxito.

5.2 Ventana de bienvenida

En la ventana de bienvenida tendremos simplemente el logo de nuestra aplicación, que además de servir de bienvenida para que el usuario sea consciente de la aplicación que acaba de abrir, nos permite darle tiempo a la aplicación a conectarse con la base de datos.

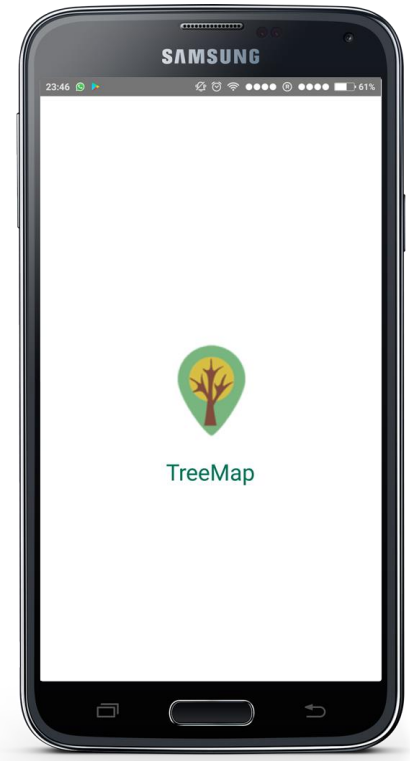

Figura 39: *Pantalla de bienvenida*

Vemos que se trata de un diseño sencillo, con el fondo blanco para hacerlo agradable a la vista, que tras algunos segundos desaparece para dejar paso a la siguiente pantalla.

5.3 Pantalla para iniciar sesión / registrarse

Si arrancamos la aplicación por primera vez o la última vez que lo hicimos cerramos sesión, después de la pantalla de bienvenida encontraremos la siguiente pantalla:

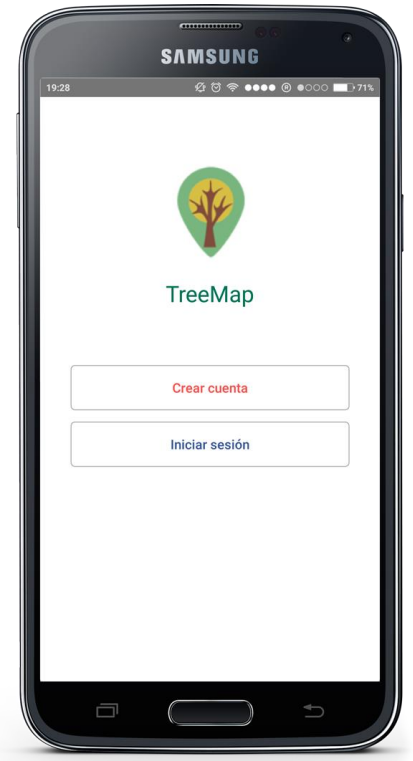

Figura 40: *Pantalla para iniciar sesión / registrarse*

Podemos ver que simplemente tenemos dos botones, con dos acciones distintas para elegir, por un lado crear cuenta, que nos llevará a la pantalla para registrarnos o bien iniciar sesión si ya tenemos una cuenta con la que acceder.

5.3.1 Crear cuenta

En esta ventana tenemos un sencillo formulario para crear una nueva cuenta, que se almacenará en la base de datos. Tenemos que ingresar el nombre y los apellidos, el nombre de usuario que será el que vean los demás usuarios cuando visualicen un árbol añadido por nosotros, el correo electrónico y la contraseña por duplicado para evitar errores.

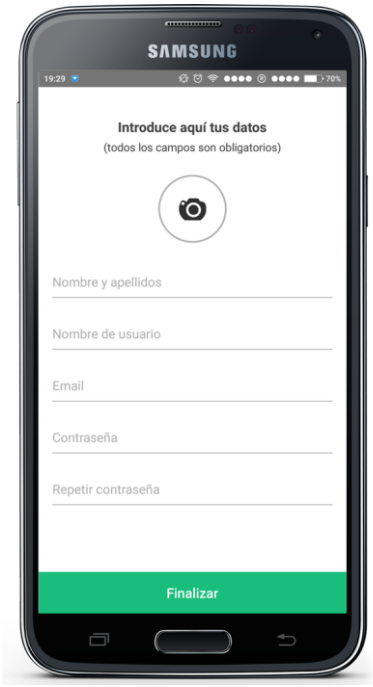

Figura 41: *Pantalla de registro*

Además de añadir los datos, en la pantalla se hace un control de los datos para evitar nombres demasiado cortos, contraseñas poco seguras, etc. Podemos ver un ejemplo de control en la imagen siguiente.

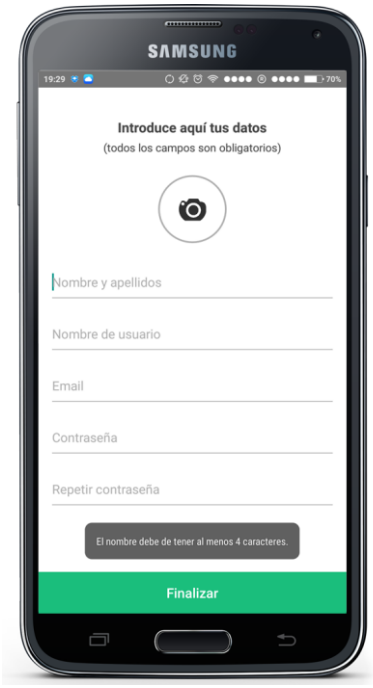

Figura 42: *Control sobre el nombre*

Además de rellenar todos los campos, que son obligatorios, podemos elegir de forma opcional si añadimos una foto nuestra, pulsando sobre el icono de la cámara de fotos aparecerá un menú que permite elegir si tomamos una foto en el momento, abriendo la cámara para ello, o si queremos elegir una foto de nuestra galería.

Figura 43 : *Añadir fotografía*

En la imagen siguiente podemos ver qué sucede cuando pulsamos sobre la opción de "Elegir fotografía desde la galería", vemos que nos da a elegir las diferentes formas de acceder a las fotografías del almacenamiento de nuestro teléfono, pudiendo seleccionar cualquiera de las opciones.

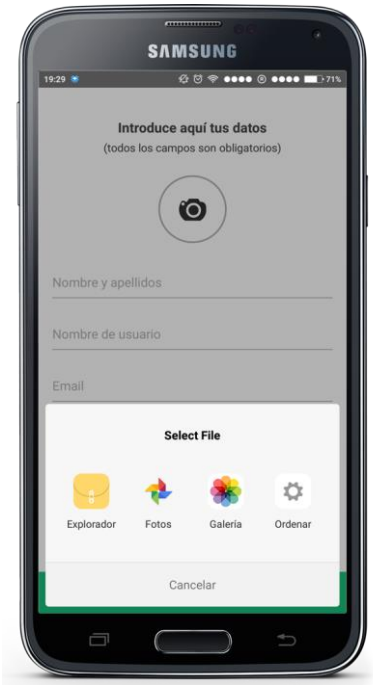

Figura 44: *Selección de galería*

Si por el contrario elegimos la opción de “Hacer foto” abrirá la cámara de fotos tal y como se puede ver en la foto, podremos hacer cualquier foto y seleccionarla pulsando sobre el botón que aparece en la parte inferior derecha.

Figura 45: *Cámara desde la aplicación*

5.3.2 Iniciar sesión

En la ventana de inicio de sesión vemos simplemente la existencia de dos campos a rellenar (el del nombre de usuario y la contraseña) y un botón para entrar en la aplicación de forma sencilla, como podemos ver en la siguiente imagen.

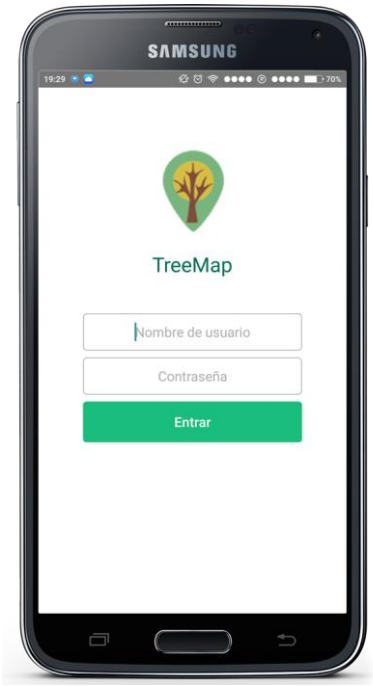

Figura 46: *Iniciar sesión*

5.4 Ventana de Inicio o "Mis árboles"

En la ventana de inicio, cuando hemos inciado sesión, tenemos por un lado en la zona central el índice de árboles que hemos insertado con nuestra cuenta, y si hay más árboles en la lista de los que caben, simplemente podremos deslizarnos haciendo "scroll" sobre la pantalla.

Figura 47: ***Pantalla principal***

Si pulsamos sobre cualquiera de los árboles de la lista, vemos que se abre la ventana de "Detalle" donde podremos ver en detalle los datos del árbol añadido.

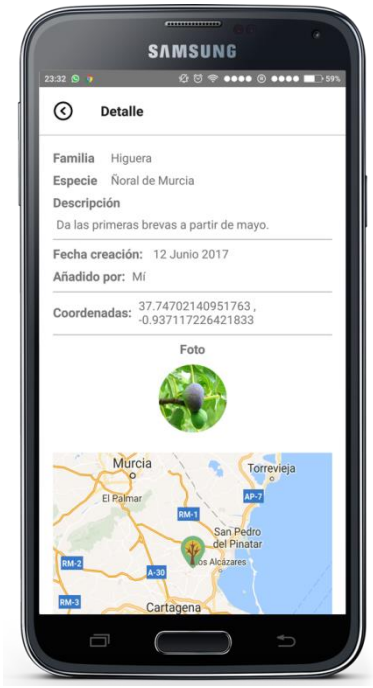

Figura 48: ***Ventana de detalle, parte 1***

Podemos ver que además de la familia y la especie del árbol, tenemos la descripción que haya podido añadir el usuario del mismo, así como la fecha de registro en la aplicación, las coordenadas exactas, una foto, que como veremos a continuación se puede ver en grande y por último la vista del árbol sobre el mapa de una forma más aislada.

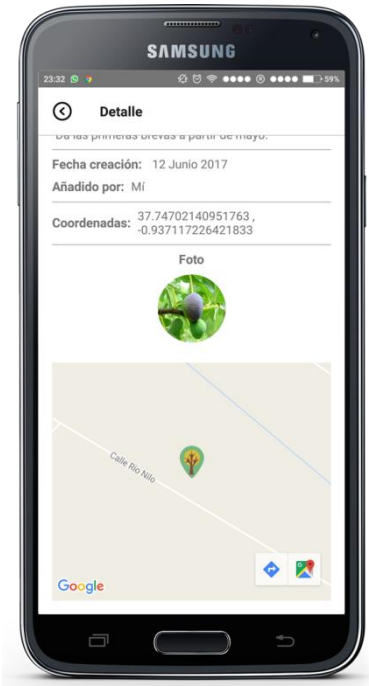

Figura 49: *Ventana de detalle, parte 2*

Como el contenido de la ventana ocupa más espacio que las dimensiones de la pantalla del móvil, se puede hacer un scroll para ver hasta abajo el mapa, sobre el cual se puede hacer un zoom como vemos en la imagen anterior y podemos pulsar sobre el botón inferior de la derecha para que nos los abra en Google Maps o nos de las indicaciones para llegar hasta el punto exacto.

Además de todo esto, si pulsamos sobre la miniatura de la foto, podremos verla a pantalla completa para apreciar al detalle lo que el usuario que ha añadido el árbol ha querido mostrar, para que cualquier interesado pueda ver el estado del mismo o el tamaño de sus frutos como es nuestro caso.

Figura 50: *Foto en ventana de detalle*

A esta ventana de detalle también se puede acceder, como veremos más tarde, desde la ventana del "Mapa".

En la lista de árboles, si realizamos una pulsación prolongada sobre cualquiera de ellos, podremos eliminar el árbol de una manera sencilla, pulsando sobre el botón de "Eliminar" de la pantalla nueva que aparece sobre la anterior.

Figura 51: *Eliminar un árbol*

Una vez seleccionado el botón, una nueva pantalla aparecerá para que confirmemos, si estamos seguros, el borrado del árbol, ya que esta operación no tiene vuelta atrás.

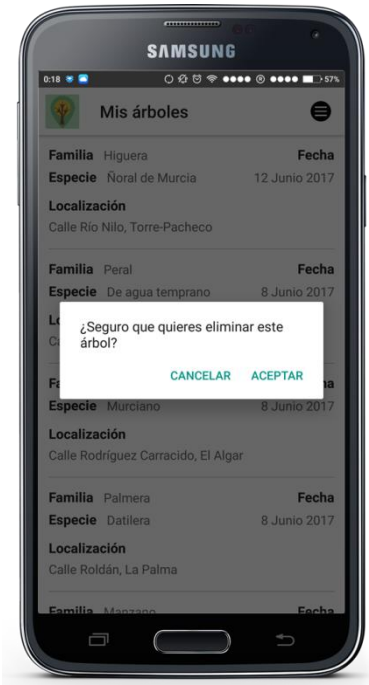

Figura 52: ***Confirmación requerida para eliminar***

Si volvemos de nuevo a la lista de árboles añadidos, en el lateral derecho de esta pantalla tenemos el menú, que se despliega o bien pulsando el botón superior derecho, como podemos ver en la imagen anterior, o bien deslizando el dedo de derecha a izquierda.

Podemos ver en la siguiente imagen que podemos acceder a las diferentes pantallas de la aplicación de una forma sencilla y también podemos ver una foto de nuestro perfil y nuestro nombre de usuario.

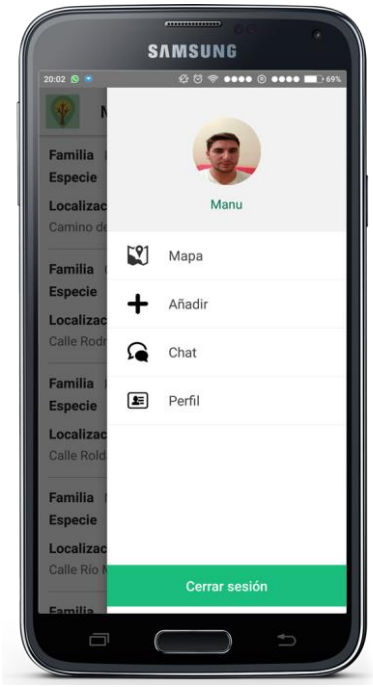

Figura 53: *Menú desplegado*

Y por supuesto, si pulsamos sobre la miniatura de la foto de perfil, nos abrirá la foto a tamaño completo.

Figura 54: *Foto de perfil*

5.5 Registro de árboles

Para registrar un árbol, simplemente tendremos que pulsar en el menú sobre la opción "Añadir" y se abrirá una nueva ventana con un formulario para registrar cualquier árbol.

Figura 55: *Ventana para añadir un árbol, parte 1*

Podemos ver que tenemos 3 campos a rellenar, dos de ellos son una lista desplegable con las opciones disponibles en la base de datos, siendo posible indicar que tenemos un árbol de una familia de las que están recogidas pero con una especie distinta, simplemente indicando en la especie “Otra” y añadiendo en la descripción el nombre de la especie de forma obligatoria para este caso.

Figura 56: *Ventana para añadir un árbol, parte 27*

Una vez rellenados los campos necesarios (familia y especie) y si queremos, añadido una descripción y una foto, podemos pasar al siguiente paso del registro de nuevos árboles, en este caso es un mapa, sobre el cual podremos hacer zoom, movernos y pulsar donde queramos añadirlo, además de buscar por ciudades e incluso calles gracias al servicio de Google.

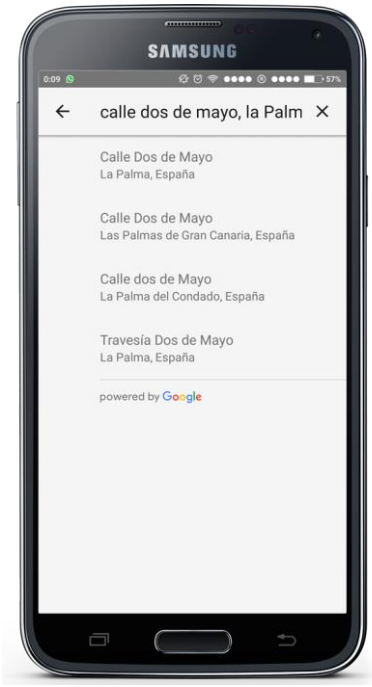

Figura 57: *Búsqueda de localizaciones*

Vemos que buscando la calle nos sugiere direcciones posibles, si la que buscamos es una de ellas, podemos pulsar sobre esa opción y el marcador del árbol se irá y se colocará en el lugar seleccionado, como podemos ver a continuación.

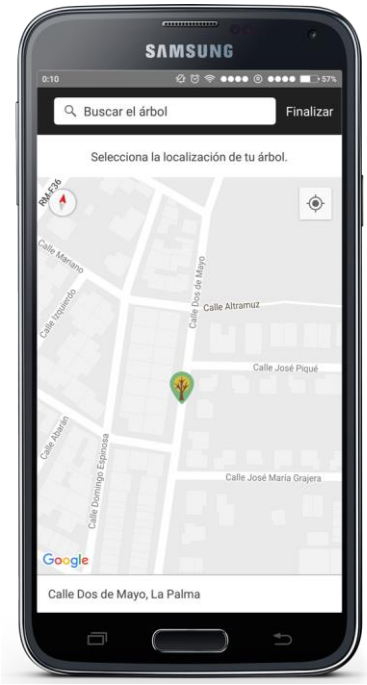

Figura 58: *Localización del árbol tras búsqueda*

Una vez seleccionado el lugar donde se encuentra nuestro árbol, podemos pulsar el botón superior derecho, “Finalizar”. Si todo ha ido bien, aparecerá un mensaje de éxito y nos rederigirá a la pantalla de “Árboles” por si queremos verificar que se ha añadido con éxito.

Figura 59: *Mensaje de confirmación de árbol añadido*

5.6 Visualización de los árboles registrados

Para visualizar los árboles registrados en el sistema, podremos acceder mediante la pestaña "Mapa" del menú.

Figura 60: *Visualización de los árboles añadidos por los usuarios*

Aunque cuando se abra la pantalla, sobre el mapa se vean pocos marcadores, estos se agrupan por zonas cuando el zoom no está lo suficientemente cerca, por lo que si vamos ampliando, veremos que existen multitud de ellos donde antes sólo aparecía uno.

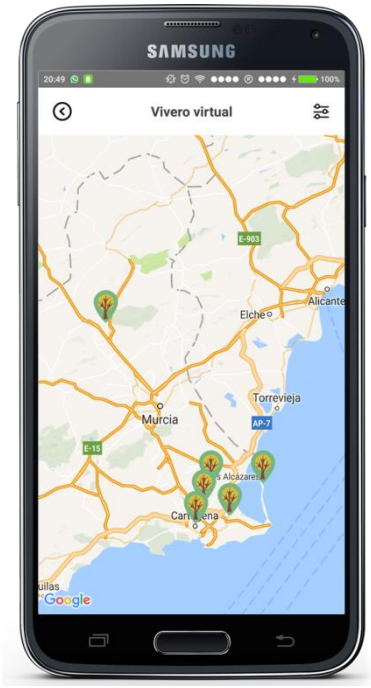

Figura 61: ***Zoom sobre el mapa***

Además, podemos filtar sobre el mapa los árboles que queremos que aparezcan, por si estamos interesados simplemente en una familia o una especie concreta. Para ello, pulsamos sobre el botón que aparece en la parte superior derecha y nos abrirá una nueva ventana en la que podemos seleccionar por la familia a buscar.

Figura 62: *Filtro del mapa*

Una vez seleccionada la familia, podemos pulsar el botón “Aplicar filtro” directamente o desplegar la lista de especies de esa familia y elegir una.

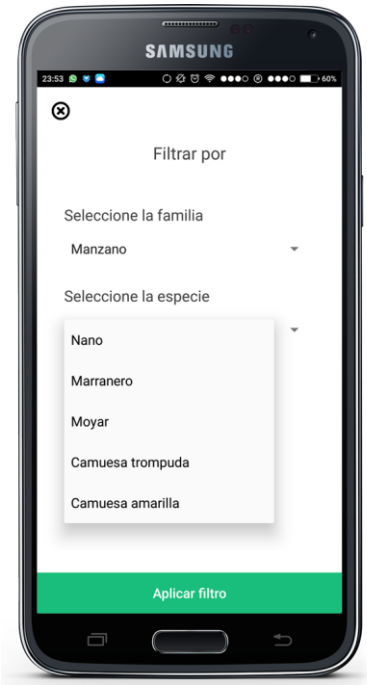

Figura 63: *Filtro por especie*

Una vez elegido el filtro, lo aplicamos y vemos como efectivamente, sobre el mapa sólo aparecen estos árboles (en este caso, solamente dos en el sureste peninsular).

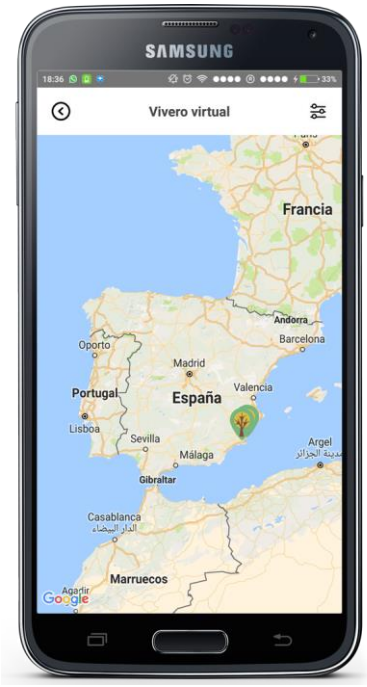

Figura 64: *Resultado de filtro por especies*

Además de realizar el filtrado de árboles, también podemos pulsar sobre cualquiera de los marcadores y aparecerá una ventana sobre el marcador del mapa con una serie de detalles, que podemos ver en la imagen siguiente.

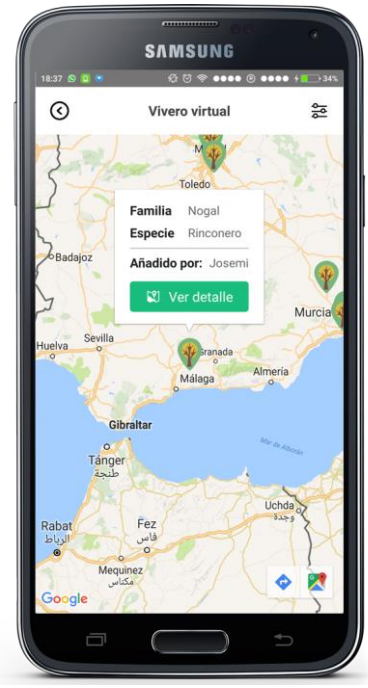

Figura 65: *Ventana al pulsar sobre un árbol*

Si pulsamos sobre el botón ver detalle, veremos la vista de detalle, al igual que la veíamos de nuestros propios árboles desde la ventana principal, sólo que además, como el árbol no es nuestro, tendremos un botón "Contactar" para iniciar una conversación, que veremos a continuación.

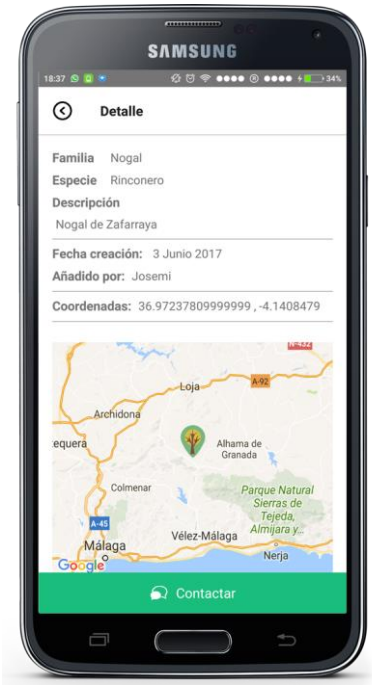

Figura 66: *Vista de detalle*

5.7 Chat

En la ventana del chat nos encontramos con la lista de conversaciones abiertas, tanto las que hemos creado nosotros desde la pantalla vista en el apartado anterior, como las que otra persona, interesada en nuestro árbol, ha creado para contactar con nosotros.

5.7.1 Lista de conversaciones

Vemos que las conversaciones se ordenan según la fecha y la hora del último mensaje, y desde la vista en la que tenemos la lista de conversaciones, podemos ver a modo de información el último mensaje que se ha escrito en cada conversación y el árbol en el que está interesado el creador de la conversación.

Figura 67: ***Lista de conversaciones***

En la lista de chats podemos ver un punto rojo para las conversaciones con mensajes sin leer.

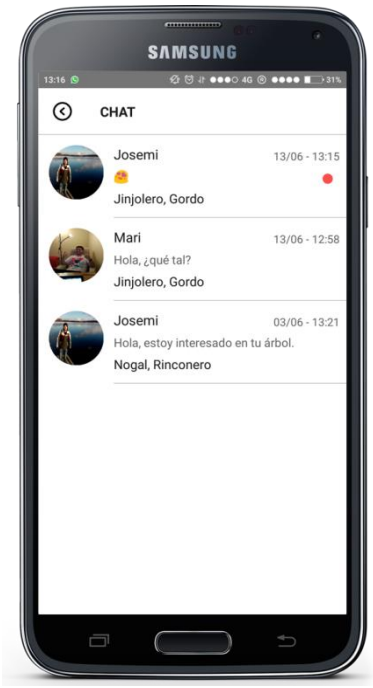

Figura 68: ***Aviso de mensaje sin leer***

Además del punto rojo, la aplicación cuenta con el sistema de notificaciones gracias al cual podemos ver el mensaje que nos han enviado y pulsar para abrir la aplicación.

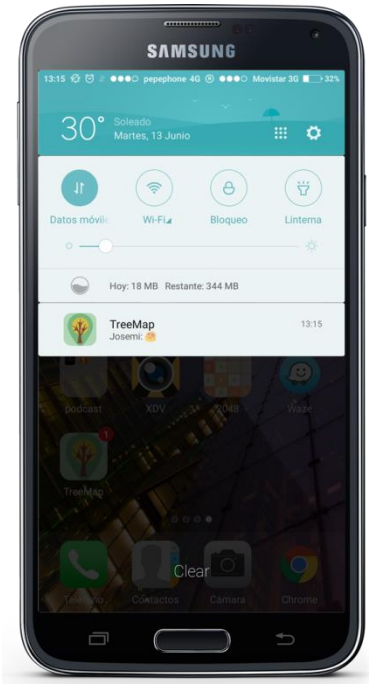

Figura 69: *Notificación de TreeMapMb*

5.7.2 Conversación

Una vez dentro de la conversación podemos ver los mensajes en el orden inverso de llegada, los más nuevos se sitúan en la parte más baja de la conversación.

Figura 70: *Conversación, parte 1*

Podemos ver también que tenemos un sistema de check (verificación de la llegada del mensaje al receptor) similar al utilizado por WhatsApp.

Figura 71: ***Conversación, parte 2***

Cuando tenemos un mensaje escrito, el botón de enviar se pone de color rojo, indicando que hay texto disponible para enviar a la conversación.

Figura 72: ***Botón enviar de color rojo***

Por último, podemos ver la conversación desde el punto de vista del otro interlocutor.

Figura 73: ***Conversación vista por el otro interlocutor***

5.8 Perfil

En la ventana de perfil, podemos ver nuestros datos de una forma muy sencilla.

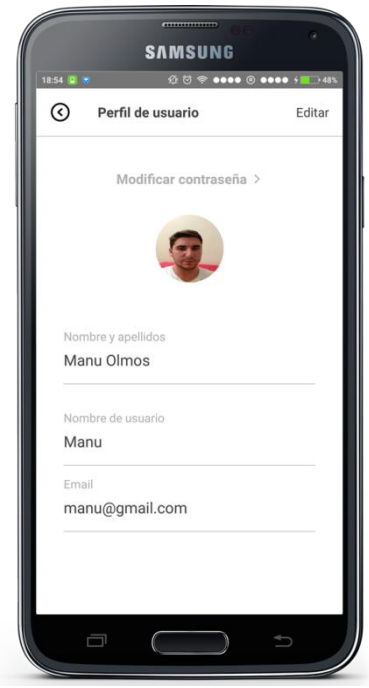

Figura 74: *Pantalla de perfil, parte 1*

También podemos editarlos simplemente pulsando el botón superior derecho “Editar”, que permite modificar todos los campos, incluido el de la fotografía.

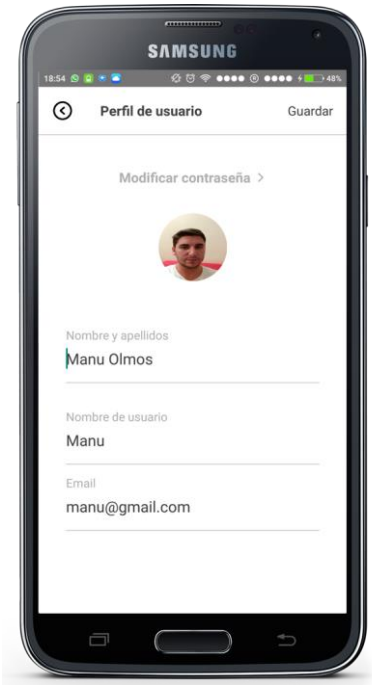

Figura 75: *Pantalla de perfil, parte 2*

Una vez modificados los campos que el usuario quiera, se debe pulsar sobre el botón guardar, que nos dirigirá a la ventana principal y mostrará un mensaje de éxito.

Figura 76: *Mensaje de éxito al guardar perfil*

Por otro lado tenemos el botón de “Modificar contraseña” que se encuentra sobre la fotografía, que abre una nueva ventana (en modo pop-up) para introducir nuestra nueva contraseña dos veces.

Figura 77: *Pantalla de cambio de contraseña*

Como en todas las ventanas en las que el usuario debe insertar datos, existen una serie de controles que se aseguran que lo añadido por el usuario sea coherente. En este caso, se comprueba que la contraseña nueva, añadida por duplicado, coincida para poder guardarla. Si no es así, aparecerá un mensaje informando del error.

Figura 78: *Control sobre el cambio de contraseña*

6. CONCLUSIONES Y TRABAJO FUTURO

Este apartado recoge las ideas para una continuación de este trabajo, desarrollo de la plataforma y apertura al público, además de las conclusiones obtenidas tras realizar este proyecto.

6.1 Conclusiones y líneas futuras

Para la realización de este Trabajo Fin de Máster se ha realizado un extenso estudio de las tecnologías móviles en el que se ha adquirido un amplio conocimiento de la misma además del aprendizaje sobre cómo desarrollar aplicaciones básicas para Android, siendo un campo que desconocía por completo hasta la realización de este proyecto.

En el proyecto se desarrolla el enorme potencial de este tipo de programación en el entorno de la catalogación de variedades hortofruticolas autóctonas, demostrando que la tecnología puede ser un elemento clave para mejorar la naturaleza, siendo una forma útil para poner en contacto a usuarios de todas partes, con intereses comunes, a través de una herramienta sencilla e intuitiva.

Aunque las características esbozadas hasta el momento dejan abierta la posibilidad de expandir el sistema en infinitas direcciones, a continuación se listan una serie de posibles mejoras a nivel operativo para expandir el modelo de negocio aún más:

- Extenderla a otras plataformas, con la creación de una APP para iOS que se conecte de forma remota a la misma base de datos y ofrezca las mismas características que brinda la versión para Android.

- Continuación del desarrollo de la aplicación Android añadiendo mejoras para hacer que la experiencia del usuario sea aún mejor, con funcionalidades como notificaciones de que se han añadido árboles en los que estamos interesados, sistema de reputación para los usuarios más activos, etc.

- Mejora en el rendimiento de la APP, mejorando el tiempo de carga y de respuesta, con accesos a base de datos más rápidos y mayor velocidad al inciar la aplicación, así como una mayor optimización de las fotografías añadidas por los usuarios.

- Añadir la posibilidad de administrar la aplicación creando un menú diferente habilitado únicamente para las cuentas seleccionadas.

- Mejoras diversas para el Chat, como aviso de lectura de un mensaje por parte del receptor (como el doble check azul en Whatsapp), posibilidad de enviar fotos o vídeos, etc.

- Subir la aplicación a Play Store, la tienda de aplicaciones oficial de Android para que cualquier persona del mundo pueda descargarla y utilizarla.

Además del desarrollo del código, también se podría hacer crecer la aplicación de otras formas, como abrirla al público general, anunciando la página en alguna red social, tipo Facebook o Twitter, que son expertos en anuncios dirigidos al público del sector gracias a su capacidad de manejar millones de datos (BigData), en foros especializados donde permitan el anuncio de otras páginas y con la creación de algún anuncio para webs del sector de la agricultura, para dar a conocer el proyecto y conseguir que la gente se registre y añada árboles al proyecto.

7. ANEXO: PREPARACIÓN DEL ENTORNO DE DESARROLLO DE PROGRAMACIÓN

7.1 Configuración de Google Maps en Android

Antes de empezar a utilizar esta API en nuestras aplicaciones será necesario realizar algunos preparativos, y es que para hacer uso de los servicios de Google Maps es necesario que previamente generemos una Clave de API (o API key) asociada a nuestra aplicación. Éste es un proceso sencillo y se realiza accediendo a la Consola de Desarrolladores de Google.

Una vez hemos accedido, tendremos que crear un nuevo proyecto desde la lista desplegable que aparece en la parte superior derecha y seleccionando la opción “Crear proyecto…”.

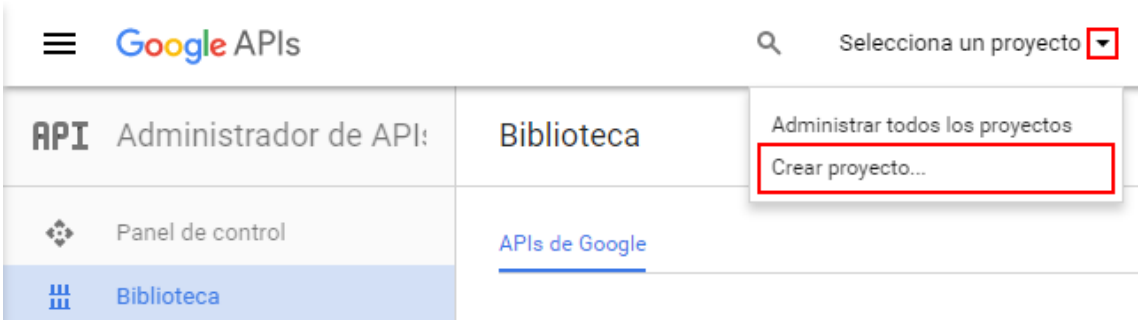

Figura 79: *Anexo 1*

Aparecerá entonces una ventana que nos solicitará el nombre del proyecto. Introducimos algún nombre descriptivo, se generará automáticamente un ID único (que podemos editar aunque no es necesario), y aceptamos pulsando “Crear”.

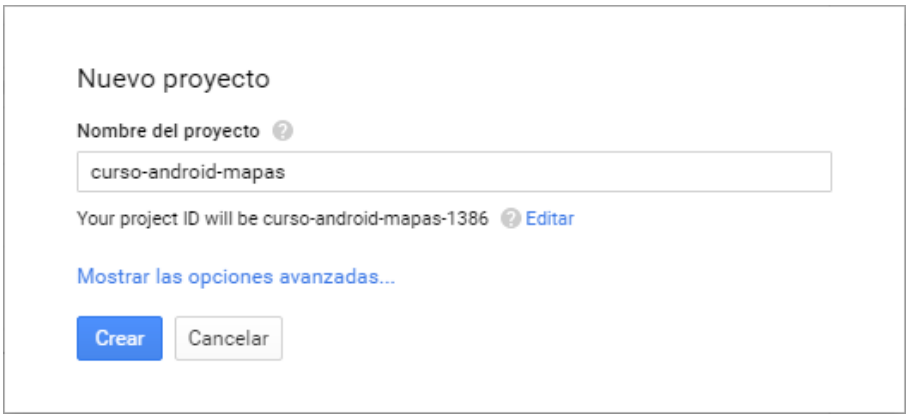

Figura 80: *Anexo 2*

Una vez creado el proyecto llegamos a una página donde se nos permite seleccionar las APIs de Google que vamos a utilizar (como podéis ver la lista es bastante extensa). En nuestro caso particular vamos a seleccionar “Google Maps Android API“.

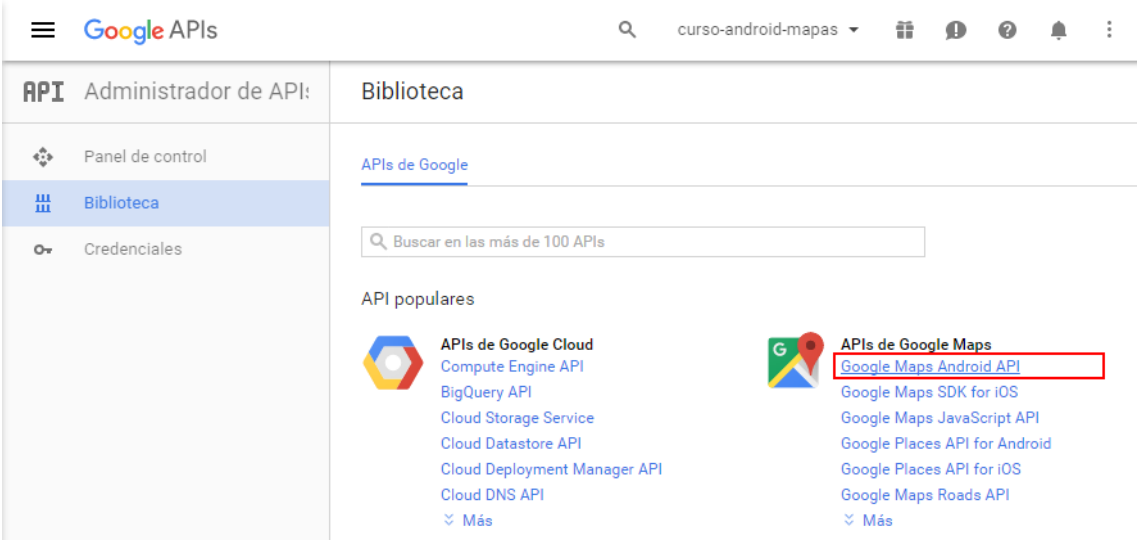

Figura 81: *Anexo 3*

Aparecerá entonces una ventana informativa con una breve descripción de la API y una advertencia indicando que se necesitará una clave de API para poder utilizarla. Por el momento vamos a activar la API haciendo click sobre la opción “HABILITAR” que aparece en la parte superior.

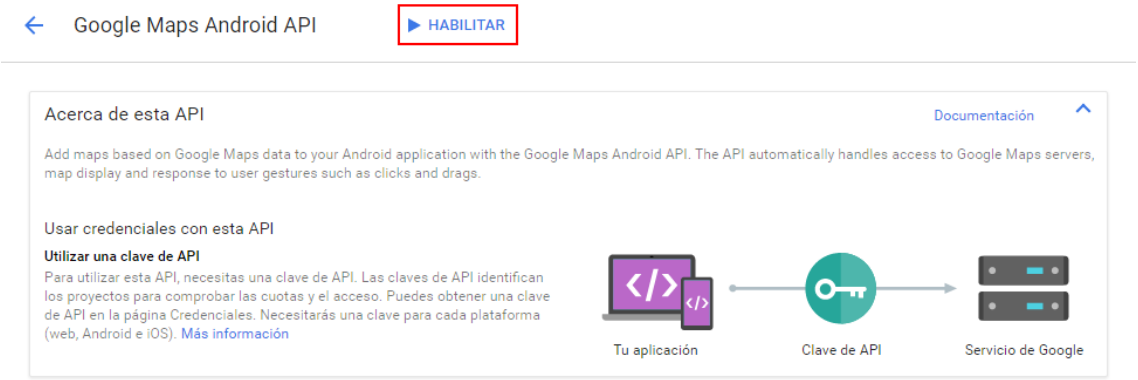

Figura 82: *Anexo 4*

Una vez activada nos vuelve a aparecer la advertencia sobre la necesidad de obtener credenciales para el uso de la API por lo que, ahora sí, tendremos que iniciar el proceso de obtención de la clave. Pulsaremos para ello sobre el botón “Ir a las credenciales“.

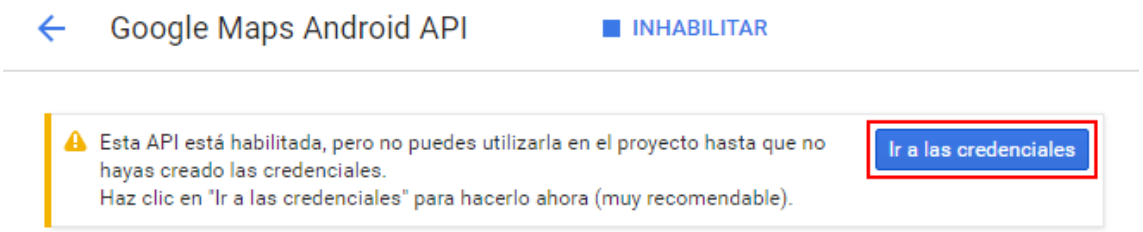

Figura 83: *Anexo 5*

Esto iniciará un pequeño asistente. En el primer paso nos volverán a preguntar qué API vamos a utilizar (aunque debería aparecer seleccionada por defecto la de Google Maps para Android), y desde dónde vamos a utilizarla. Indicamos "Android" y pulsamos el botón "¿Qué tipo de credenciales necesito?".

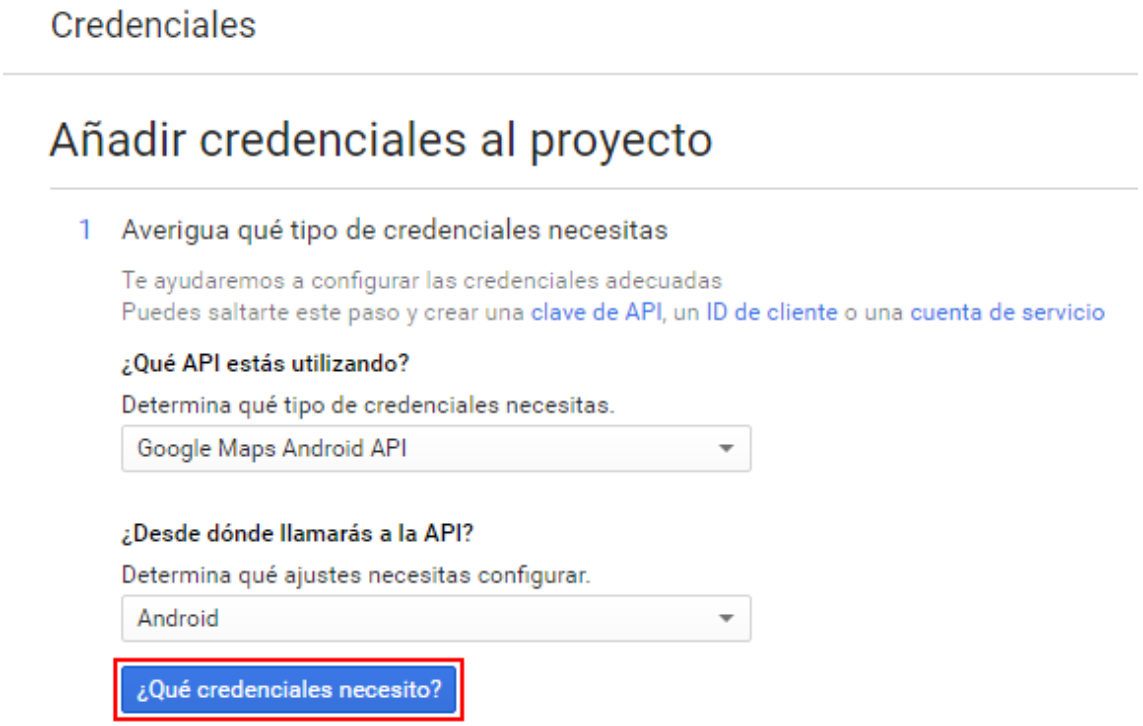

Figura 84: *Anexo 6*

En el segundo paso tendremos que poner un nombre descriptivo a la clave de API que se va a generar, no es demasiado relevante, por lo que podemos dejar el que nos proponen por defecto. También en este paso se nos da la posibilidad de poder restringir el uso de este proyecto a determinadas aplicaciones concretas. Como es una práctica bastante recomendable vamos a ver cómo hacerlo. Pulsaremos sobre el botón "+ Añadir nombre de paquete y huella digital" y veremos que se solicitan dos datos:

Nombre de paquete

Huella digital de certificado SHA-1

El primero de ellos es simplemente el paquete java principal que utilizaremos en nuestra aplicación. Lo indicaremos al crear nuestro proyecto en Android Studio (o si ya tenemos una aplicación creada podemos encontrarlo en el fichero AndroidManifest.xml, en el atributo package del elemento principal).

El segundo de los datos requiere más explicación. Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Este proceso de firma es uno de los pasos que tenemos que hacer siempre antes de distribuir públicamente una aplicación.

Adicionalmente, durante el desarrollo de la misma, para realizar las pruebas y la depuración del código, aunque no seamos conscientes de ello también estamos firmado la aplicación con un "certificado de pruebas". Pues bien, para obtener la huella digital del certificado con el que estamos firmando la aplicación podemos utilizar la utilidad keytool que se proporciona en el SDK de Java.

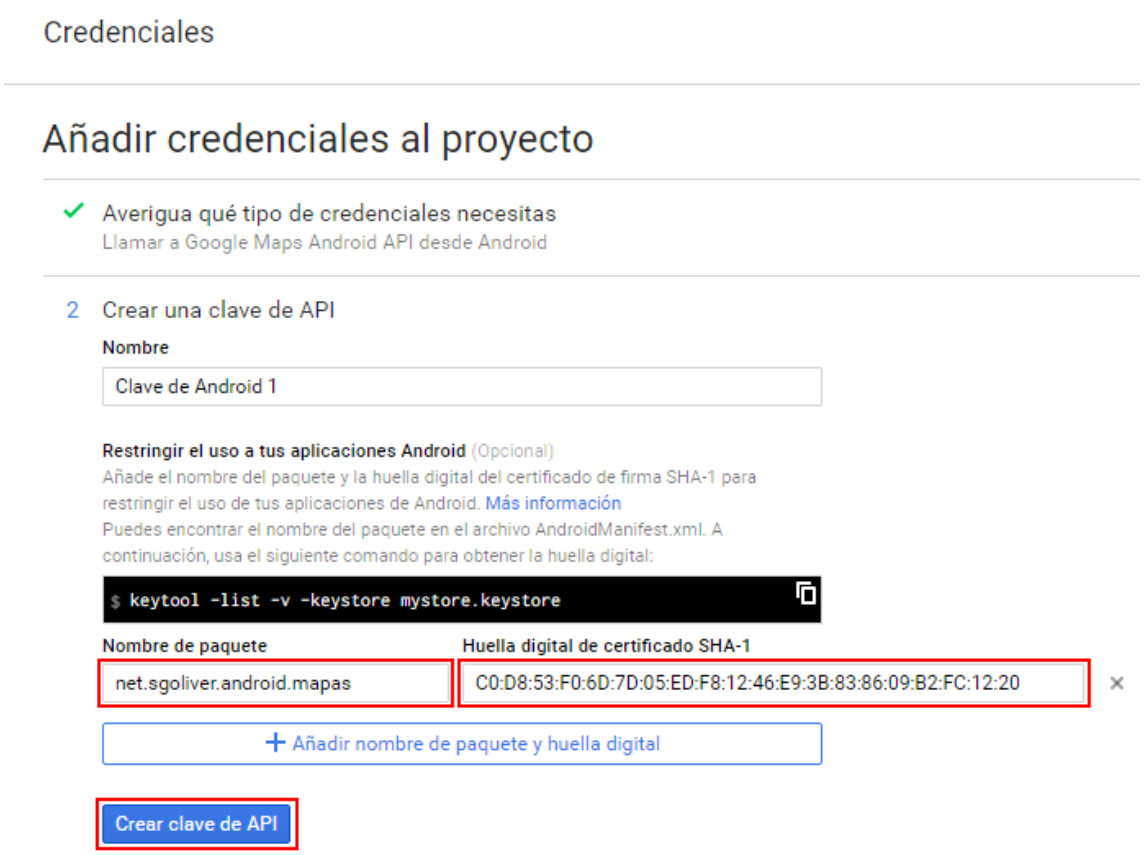

Figura 85: *Anexo 7*

Hecho esto, obtendremos por fin nuestra clave de API. La copiamos en lugar seguro (más tarde nos hará falta para nuestra aplicación Android), y pulsamos el botón “Listo” para finalizar.

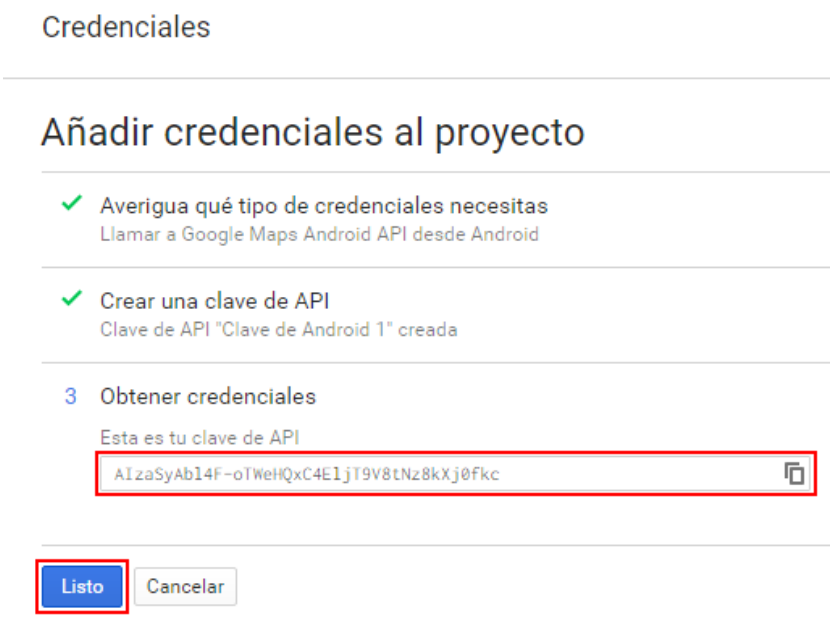

Figura 86: *Anexo 8*

Con la clave de API ya sólo nos quedaría añadirla a nuestro código para poder disfrutar de Google Maps en nuestra aplicación. En el fichero AndroidManifest.xml, dentro del elemento <application> tendremos que añadir un nuevo elemento <meta-data> en el que se especificará la versión de los Google Play Services utilizada para compilar la aplicación.

Inmediatamente después añadiremos otro nuevo <meta-data> donde indicaremos la clave de API para Google Maps que hemos obtenido a través de la consola de desarrolladores.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyAbl4F-oTWeHQxC4EljT9V8tNz8kXj0fkc"/>
```

Figura 87: *Anexo 9*

Ya tendríamos nuestra clave, que permitirá que funcione Google Maps en nuestra aplicación. [23]

Figura 88: *Anexo 10*

7.2 Primeros pasos en Android Studio

Una vez que conoces un poco Java y XML puedes ver de qué son capaces a la hora de empezar a desarrollar una aplicación Android. Para empezar lo primero que tendremos que hacer es instalar el entorno de desarrollo.

Una vez descargado, instalado y configurado correctamente, el siguiente paso es el de crear un proyecto. Para crear el proyecto ejecutaremos Android Studio y dejaremos que cargue la configuración inicial, acto seguido nos aparecerá una ventana en la que podremos elegir entre diferentes opciones, pero nosotros seleccionamos Start a new Android Studio proyect.

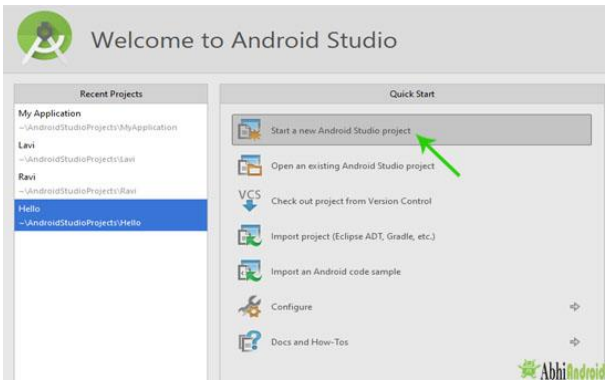

Figura 89: *Anexo 11*

Ahora, nos encontramos con una ventana en la que nos preguntará por el nombre de la aplicación, el dominio y la carpeta en la que guardará el proyecto.

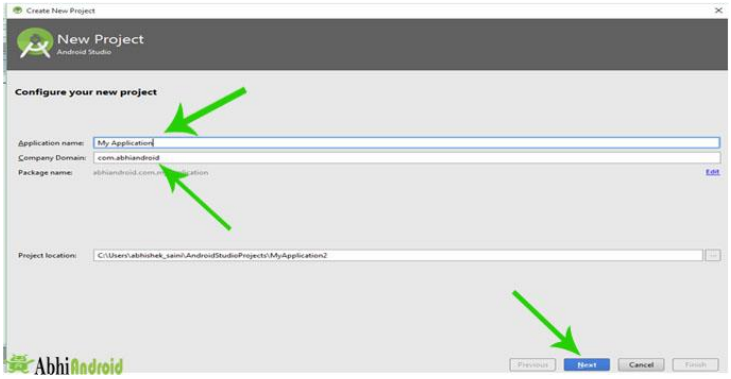

Figura 90: *Anexo 12*

Al pulsar Next podremos elegir si será una aplicación para móviles y tablets, para Android Wear, Android TV, Android Auto o para las Google Glass.

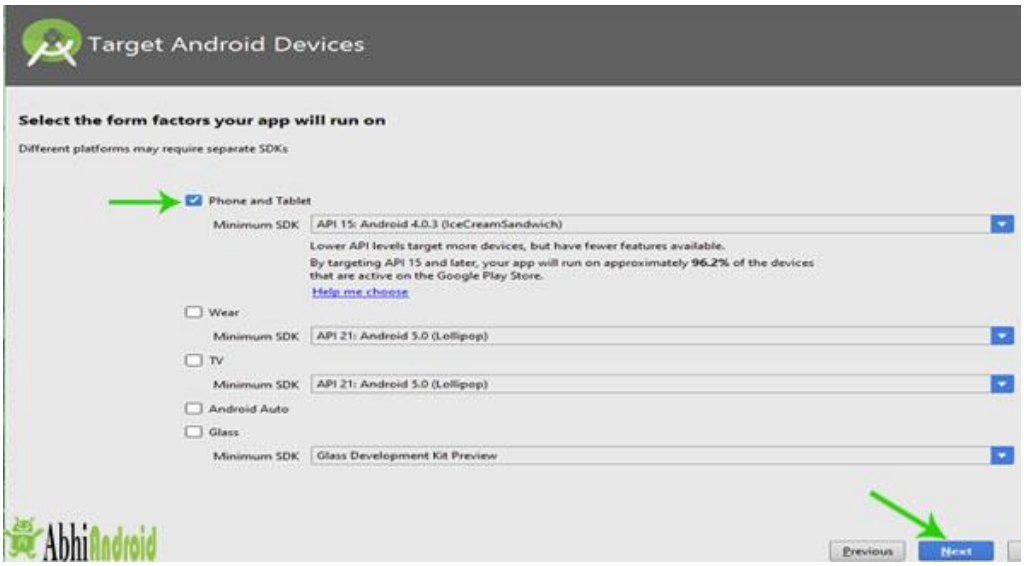

Figura 91: *Anexo 13*

A continuación veremos una ventana en la que nos dará a elegir entre una serie de diseños basados en Material Desing. Una vez seleccionado, la plataforma empezará a preparar todos los archivos necesarios para que nos sea más sencillo a la hora de empezar a programar nuestra nueva aplicación.

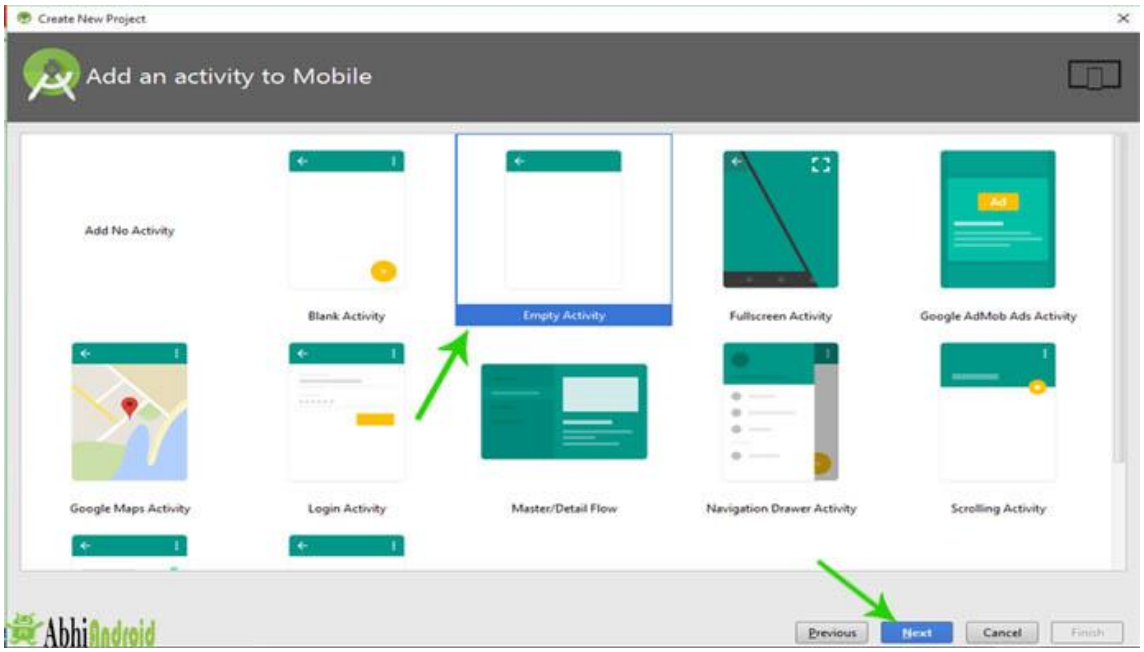

Figura 92: *Anexo 14*

Cuando finalice el proceso, estaremos en el editor de código de la aplicación, por lo que vamos a enseñaros donde están los archivos Java y XML para que empecéis a practicar con ellos.

Podremos crear una nueva "Activity" de una forma sencilla para empezar a trabajar.

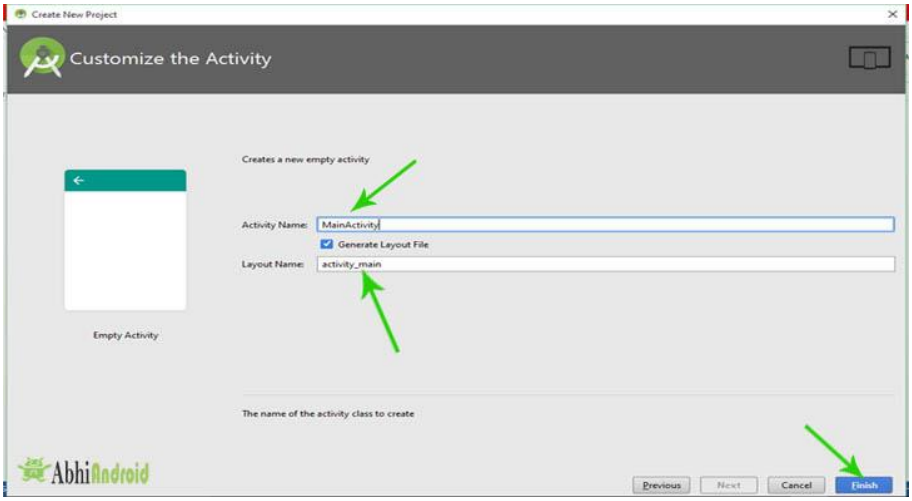

Figura 93: *Anexo 15*

¿Dónde puedo encontrar los archivos Java y XML en Android Studio?

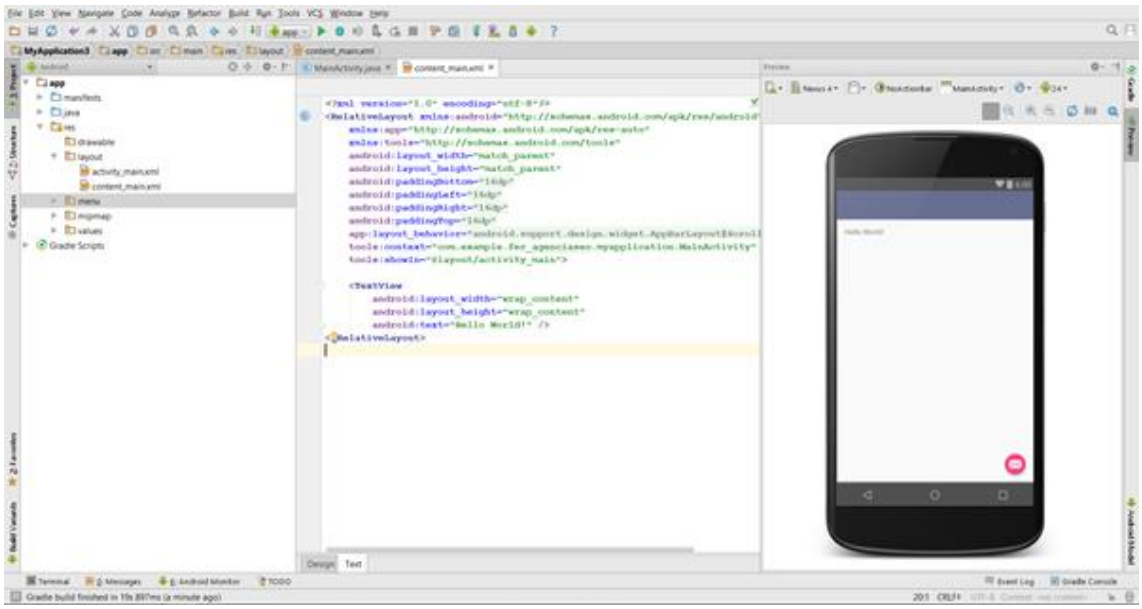

Figura 94: *Anexo 16*

Lo primero, en la carpeta Java, vamos a encontrar obviamente todos los recursos y archivos de Java que vayamos creando para nuestra aplicación.

Ahora bien, en las carpetas Values y XML encontraremos los archivos XML que corresponderán al diseño que queramos darle a nuestra aplicación.

Como podemos ver, Java y XML son dos lenguajes imprescindibles para desarrollar aplicaciones en Android. Por eso es necesario saber cómo utilizarlas para poder empezar a desarrollar tus ideas. [24]

8. REFERENCIAS BIBLIOGRÁFICAS

[1] «Tecnología Android» 2012. [En línea]. Available: http://tecnologiasandroid.blogspot.com.es/2012/05/definicion-android.html.

[2] J. P. Porto, «definicion.de» 2015. [En línea]. Available: http://definicion.de/android/ .

[3] «conceptodefinicion.de» 2011. [En línea]. Available: http://conceptodefinicion.de/android/.

[4] «Academia Android,» 2014. [En línea]. Available: https://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/.

[5] «Android Developers» 2017. [En línea]. Available: https://developer.android.com/studio/intro/index.html?hl=es-419 .

[6] «Definicion.de» 2013. [En línea]. Available: http://definicion.de/java/.

[7] «Personales.UPV» [En línea]. Available: http://personales.upv.es/rmartin/cursoJava/Java/Introduccion/PrincipalesCaracteristicas.htm.

[8] «Definicion.de» 2013. [En línea]. Available: http://definicion.de/xml/.

[9] J. Cadillo, «Conocimiento y Sistemas» 2010. [En línea]. Available: https://conocimientoysistemas.wordpress.com/tag/caracteristicas-xml/.

[10] «FUNDAMENTOSDEXML» 2012. [En línea]. Available: https://fundamentosdexml.wordpress.com/2012/04/19/ventajas-y-desventajas-de-xml-2/.

[11] «Miguel Diaz Rubio» 2013. [En línea]. Available: http://www.migueldiazrubio.com/desarrollo-ios-parse-i-introduccion-e-instalacion/.

[12] «API, Ticbeat» 2015. [En línea]. Available: http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/.

[13] «GoogleMaps, Google» 2015. [En línea]. Available: https://www.google.es/intx/es-419/work/mapsearth/products/mapsapi.html.

[14] «Google Maps Api -Introducción» [En línea]. Available: https://developers.google.com/maps/documentation/android-api/intro?hl=es-419.

[15] «GoogleMaps, MaestrosDelWeb» 2015. [En línea]. Available: http://www.maestrosdelweb.com/google-maps-api-v3-introduccion-y-primeros-pasos/.

[16] «Google Maps API - Documentación» 2017. [En línea]. Available: https://developers.google.com/maps/documentation/android-api/map?hl=es-419.

[17] «RECUPERANDO VARIEDADES ANTIGUAS DE MANZANA,ANSE» 2015. [En línea]. Available: http://www.asociacionanse.org/recuperando-variedades-antiguas-de-manzana/20080723.

[18] A. p. l. C. d. l. H. d. Murcia, «Huermur - Variedades tradicionales de la Huerta de Murcia» [En línea]. Available: https://es.scribd.com/doc/41175258/Variedades-tradicionales-de-la-Huerta-de-Murcia.

[19] D. Ruiz, «Fruta de Hueso» 2016. [En línea]. Available: http://frutadehueso.com/documentos/david-ruiz.pdf.

[20] «CEBAS-CSIC» 2010. [En línea]. Available: http://www.cebas.csic.es/documentos/Fichas%20variedades%20almendro/Triptico%20Marta.pdf.

[21] «Clima de España, Wikipedia» 2015. [En línea]. Available: https://es.wikipedia.org/wiki/Clima_de_Espa%C3%B1a.

[22] «Selección en variedades hortícolas tradicionales, Interempresas.net» 2005. [En línea]. Available: http://www.interempresas.net/Horticola/Articulos/76166-Seleccion-en-variedades-horticolas-tradicionales.html.

[23] «SGOLIVER.NET» 2016. [En línea]. Available: http://www.sgoliver.net/blog/mapas-en-android-google-maps-android-api-1/.

[24] F. Pérez, «Yeeply.com» 2016. [En línea]. Available: https://www.yeeply.com/blog/lenguajes-basicos-desarrollador-android/.