UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Spot&Joy: Reproductor de listas de Spotify contextual y pervasivo

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Spot&Joy: Reproductor de listas de Spotify contextual y pervasivo

Autor: José Joaquín Abril Torrejón

Tutor: Juan Manuel Murillo Rodríguez

Co-Tutor/es: José Manuel García Alonso

ÍNDICE GENERAL DE CONTENIDOS

1. INTRODUCCIÓN 13
2. OBJETIVOS 15
3. ANTECEDENTES / ESTADO DEL ARTE 16
3.1. Internet de las Cosas 16
3.1.1. Concepto 16
3.1.2. Aplicaciones en la actualidad 18
3.1.3. Investigación 19
3.2. Otras aplicaciones sociales 20
4. MÉTODOLOGÍA 22
4.1. Fases de realización 22
4.2. Recursos disponibles 25
4.2.1. PC Portátil Asus GL552VW-DM151 25
4.2.2. Smartphones 26
4.2.2.1. Huawei Y635-L01 26
4.2.2.2. Motorola Moto E2 4G LTE (dos dispositivos) 27
5. IMPLEMENTACIÓN Y DESARROLLO 28
5.1. Análisis 28
5.1.1. Requisitos 29
5.1.1.1. Requisitos funcionales 29
5.1.1.2. Requisitos no funcionales 30
5.1.2. Casos de uso 31
5.1.3. Prototipo y componentes del proyecto 35
5.1.3.1. Construcción del prototipo 35
5.1.3.2. Identificación de componentes de la aplicación 40
5.1.3.3. Diagrama de componentes 41
5.1.4. Herramientas de desarrollo 44

5.1.4.1. Android Studio 44

5.1.4.1.1. Seleccionando versiones 44

5.1.5. Estudio de plataformas 46

5.1.5.1. Google Sign-In 46

5.1.5.1.1. Obtención de archivos necesarios 47

5.1.5.1.2. Integración en el proyecto 50

5.1.5.2. Plataforma Spotify 51

5.1.5.2.1. Spotify Android SDK 51

5.1.5.2.2. Las librerías 51

5.1.5.2.2.1. Inconvenientes de las librerías 52

5.1.5.2.2.2 Configuración de la plataforma 52

5.1.5.2.2.3. Obtención de archivos necesarios 54

5.1.5.2.2.4. Integración en el proyecto 55

5.1.5.2.3. Wrapper Web API 56

5.1.5.2.3.1. ¿Por qué el Wrapper? 56

5.1.5.2.3.2. ¿Dónde obtener el Wrapper? 57

5.1.5.2.3.3. Integración en el proyecto 57

5.1.5.3. Plataforma nimBees 59

5.1.5.3.1. ¿Qué es nimBees? 59

5.1.5.3.2. ¿Cómo obtener nimBees? 60

5.1.5.3.2.1. Integración en el proyecto 60

5.1.3.3.2.2. Configuración de Google Cloud 60

5.1.3.3.2.3. Configuración del portal nimBees 64

5.1.3.3.2.4. Configuración de nimBees en el proyecto 65

5.1.4. Otras librerías 66

5.1.4.1. Picasso 66

5.1.4.2. Guava 66

5.1.5. Conclusiones y complicaciones durante la fase de estudio 67

5.2. Diseño 67

5.2.1. Arquitectura 68

5.2.1.1. Administrador 69

5.2.1.2. Usuarios comunes o estándar 73

5.2.2 Vistas de la aplicación 76

5.2.3. Colores y estilos 78

5.2.4. Listas personalizadas 81

5.3. Implementación 83

5.3.1. Activities 83

5.3.1.1. Sign-In 83

5.3.1.2. Devices 84

5.3.1.2.1. SignOut 84

5.3.1.2.2. Buscar Dispositivos 84

5.3.1.2.2.1. Solicitud de acceso a dispositivo 85

5.3.1.2.4 Crear una sesión de reproducción 85

5.3.1.3. MainUser 86

5.3.1.3.1. Orden de ejecución 87

5.3.2. Fragments 89

5.3.2.1. Mis Playlists 89

5.3.2.2. Reproductor 91

5.3.2.3. Buscador 92

5.3.2.4. Lista de Canciones 95

5.3.2.5. Usuarios 97

5.3.3. Clases implementadas 98

5.3.3.1. Wrapper de Spotify 98

5.3.3.2. UserDevice 99

5.3.3.3. TrackSelect 99
5.3.3.4. Compatibility 99
5.3.3.5. SystemTrack 99
5.3.3.6. PlayerTrack 100
5.3.4. Servicios 100
5.3.4.1. PlayerView Service 102
5.3.4.2. Users Service 102
5.3.4.3. Player Service 103
5.3.4.3.1. Lista de reproducción 105
5.3.4.3.2. SpotifyPlayer 106
5.3.4.3.2.1. Cambio de canción 107
5.3.4.3.2.2. La última canción 107
5.3.4.3.3. Enviar y reproducir (Mezclador Armónico) 108
5.3.4.3.3.1. Tabla de Compatibilidades 112
5.3.4.3.3.2. Algoritmo de inserción 115
5.3.5. Gestor de Credenciales 116
5.3.6. Adapters 116
5.3.7. Comunicaciones 117
5.3.7.1. Mensaje 117
5.3.7.2. Tipos de Mensaje 118
5.3.8. BroadcastReceiver 123
5.3.9. Manifest 124
5.4. Pruebas 125
6. RESULTADOS Y DISCUSIÓN 126
7. CONCLUSIONES 128

ÍNDICE DE TABLAS

1. Colores de la aplicación y uso 78
2. Objetos de modelo del Wrapper utilizados 98
3. Claves Cameloth Wheel 110
4. Correspondencia de clases tonales 111
5. Tipos de Mensajes 119

ÍNDICE DE FIGURAS

1. **Escenario de Spot&Joy** 14
2. **Captura de pantalla de FLO Music** 20
3. **Captura de pantalla de Outloud** 21
4. **Estrategia de desarrollo** 24
5. **Actores** 31
6. **Diagrama de Casos de Uso (1)** 32
7. **Diagrama de Casos de Uso (2)** 33
8. **Diagrama de Casos de Uso (3)** 34
9. **Mockup (Inicio y registro)** 37
10. **Mockup (Lista de dispositivos)** 37
11. **Mockup (Gestión del administrador)** 38
12. **Mockup (Vista de reproducción del administrador)** 38
13. **Mockup (Vista de reproducción de usuario estándar)** 39
14. **Diagrama de componentes** 42
15. **Gráfica de uso de APIs de Android** 45
16. **Google Sign In (Creando una aplicación)** 47
17. **Huella digital del proyecto Android** 48
18. **Generar archivo google-services** 49
19. **Añadiendo archivo google-services** 50
20. **My Applications (Spotify)** 53
21. **Configurando la aplicación** 53
22. **Google Cloud (Inicio)** 61
23. **Google Cloud (APIs disponibles)** 62
24. **Google Cloud (Creando una nueva clave de API)** 62
25. **Google Cloud (Restrición del proyecto vía SHA-1)** 63
26. **Google Cloud (Propiedades del proyecto)** 63
27. **Configurando nuevo proyecto en nimBees** 64
28. **Propiedades de nimBees en la aplicación** 65
29. **Arquitectura general de la aplicación** 68
30. **Arquitectura software (Administrador)** 70
31. **Arquitectura software (Usuario estándar)** 74
32. **Vistas activities** 76
33. **Vistas de fragments (1)** 77
34. **Vistas de fragments (2)** 77
35. **Icono del usuario por defecto** 79
36. **Icono de canción por defecto** 80
37. **Icono de dispositivo móvil** 80
38. **Logo de la aplicación** 80
39. **Icono de la aplicación** 81
40. **Elemento de una lista personalizado (layout.xml)** 82
41. **Variables Activity Devices** 85
42. **NavigationView (Implementación)** 88
43. **Carga del fragment Player** 88
44. **Obtención de Playlists** 90
45. **Fragment Buscador (SearchPager)** 93
46. **Fragment Buscador (Obtener Datos)** 94
47. **Carga de resultados *doInBackground(…)*** 95
48. **Ejemplo de Servicio de la aplicación (Usuarios)** 101

49. Gestión del cambio de canción 107
50. Gestión de fin de canción 107
51. Añadir canción al reproductor 108
52. Cameloth Wheel 109
53. Tabla de correspondencia de claves 112
54. Correspondencias para la Tabla de Compatibilidades 113
55. *NotificationManager* de la aplicación 121
56. Ejemplo de *BroadcastReceiver* para comunicaciones 122
57. Ejemplo de BroadcastReceiver en *fragment* Reproductor 123
58. Documento *Manifest.xml* 124
59. Pantalla Inicial 131
60. Seleccionar correo 131
61. Lista de Dispositivos (sin resultados) 132
62. Lista de Dispositivos (con resultados) 132
63. Credenciales de Spotify 133
64. Credenciales de Spotify incorrectas 133
65. Reproductor admin 134
66. Pantalla Principal (admin) 134
67. Mis playlists 135
68. Buscador 135
69. Lista de canciones 136
70. Reproductor (una canción) 137
71. Reproductor (varias canciones) 137
72. Salir (admin) 138
73. Pantalla Principal (estándar) 139
74. Vista del Reproductor (estándar) 139

RESUMEN

En la actualidad, Internet se utiliza en la mayor parte del mundo, así como cada vez es mayor el número de dispositivos que se conectan a la red para compartir información.

Con la aparición del concepto conocido como *Internet de las Cosas (Internet of Things, IoT)*, cada vez son más los dispositivos que, además de estar conectados a la red, se encuentran interconectados entre sí.

Este concepto revolucionario provoca que día tras día aumente el número de plataformas *IoT*, las cuales interactúan con otros dispositivos con el fin de monitorizar la información obtenida por los mismos. Dentro de la amplia gama de dispositivos que, a día de hoy, interactúan con las plataformas *IoT* nos encontramos con SmartTVs o altavoces. Estos a su vez, interactúan con otros tipos de aplicaciones, como pueden ser las *streaming*. Un claro ejemplo cotidiano puede ser: YouTube, Netflix, Spotify...

Sin embargo, en la mayoría de las ocasiones, este tipo de aplicaciones suele presentar algunas carencias. En concreto, los dispositivos *IoT* en general y las plataformas de *streaming* en particular, no están preparadas para ser utilizadas de forma simultánea por más de un usuario.

El aumento del número de dispositivos conectados de distintos usuarios, provoca que cada vez se deban producir más interacciones sociales, como podría ser un grupo de amigos que quieran reproducir música en una fiesta, donde todos quieran que sus canciones favoritas sean escuchadas.

En el presente documento, se plantea este enfoque orientado a una de las plataformas de *streaming* más importantes, *Spotify*. Procediendo a resolver el problema de la interacción de usuarios, permitiéndoles crear una lista de reproducción en común, en base a las preferencias de cada uno.

Como solución se plantea el desarrollo de una aplicación realizada para sistemas operativos *Android*, que permite a usuarios acceder a sus *playlists* o buscar otras canciones de interés, para que éstas puedan ser compartidas y reproducidas de forma conjunta en un determinado contexto social.

Agradecimientos…

Probablemente la sección de este documento que debería ser más larga, pues aquí no sólo se recoge el trabajo y el esfuerzo diario realizado por un alumno. Son muchas las cosas que debo agradecer y muchas las personas a las que estar agradecido. Se habla del esfuerzo del alumno, pero no puedo entregar este trabajo sin mencionar a los verdaderos protagonistas de esta historia: José y Carolina, papá y mamá, por vuestro enorme sacrificio, por vuestra dedicación y paciencia, por vuestros valores, gracias.

A mi hermana, a mi tía y a mi cuñado, por estar ahí siempre que se les necesitaba, gracias.

Al amor de mi vida, por estar ahí siempre, apoyándome y animándome día tras día, ayudándome a avanzar, a dar el siguiente paso, gracias.

A los otros dos protagonistas directores de este trabajo, Juan Manuel Murillo Rodríguez y José Manuel García Alonso, por saber guiarme durante todo este camino y apoyarme, gracias.

Por todas esas caras que, a lo largo de mi tiempo en la universidad, me hacían reír cada mañana por muy temprano que fuese, gracias.

Y en general, a todo el profesorado encargado de transmitirnos el concepto de ingeniería informática mediante sus asignaturas, gracias.

1. INTRODUCCIÓN

En el presente punto, se pretende introducir al lector del documento dentro del problema al que éste proyecto se enfrente, así como explicar la solución propuesta.

A lo largo de los diferentes puntos, se le ofrece al lector un enfoque distinto de solución, partiendo de rasgos más generales, hasta los detalles más técnicos.

A pocos años de haber comenzado el siglo XXI, nos encontramos ante una sociedad completamente digitalizada, algo que era impensable no hace demasiados años. Las tecnologías no dejan de evolucionar, al igual que la conciencia de la sociedad respecto a las mismas.

Disponemos de dispositivos de todos los tamaños con distintas funcionalidades, abordando un abanico de posibilidades gigantes que no cesa de crecer, y siempre guiados bajo un mismo propósito general, que no es otro que el de facilitarnos las tareas. Teniendo en cuenta el crecimiento de Internet y su integración en los distintos sistemas, se puede decir que el ser humano vive rodeado por los mismos, y, a su vez, vive en torno a Internet.

Esto ha provocado una evolución en lo referente a dispositivos informáticos, viéndose los mismos obligados a tener que recabar información de forma constante sobre su entorno u otros dispositivos, incluso tener que intercambiar datos entre ellos. Algo que, a día de hoy, es conocido como *"Internet de las Cosas"*. Este nuevo concepto para el desarrollo de aplicaciones ha llegado a ser denominado como "La siguiente Revolución Industrial" [1]. Cada vez son más los campos de ámbitos muy distintos los que se "alían" a esta nueva vertiente: tiendas convencionales, grandes almacenes, hoteles, incluso las propias redes sociales. En resumen, las distintas aplicaciones utilizadas por las personas, se encargan de recoger información, y esta a su vez es compartida con otros dispositivos.

Un claro ejemplo de una plataforma *IoT*, podría ser el control de un termostato de una vivienda, el cual posee un conjunto de temperaturas para determinados tiempos e introducidas por un usuario. ¿Pero qué ocurre cuando el sistema debe establecer sus condiciones a las preferencias de más de un usuario?, ¿y si cada miembro de la vivienda requiere de unas preferencias completamente distintas (hora de salir a trabajar o al colegio, hora de llegada, temperatura ideal…)?

El sistema podría reajustarse manualmente cada vez que fuese necesario, pero, ¿por qué no conseguir que el sistema permitiese la interacción de un grupo de usuarios en base a sus preferencias?

El número de usuarios crece a medida que lo hacen las tecnologías, por lo que cada vez se requiere de un número de interacciones sociales mayor entre ellos.

Este es un claro ejemplo del problema existente en la actualidad en la mayoría de las plataformas *IoT*. Se puede decir que todavía existe una barrera bastante grande entre dispositivos y usuarios.

Dentro de las plataformas *streaming*, siendo ejecutadas en dispositivos *IoT (smartphones, altavoces, smarTV)*, existe el mismo problema. ¿Qué mecanismos pueden proporcionar para que un grupo de usuarios pueda realizar una reproducción social? ¿Qué pueden ver un grupo de usuarios de Netflix cuando las preferencias las realiza a nivel de usuario?

En el ámbito musical dentro de estas plataformas, podemos encontrar *Spotify*. Siendo esta la más extendida mundialmente con más de 140 millones de usuarios. Y, sin embargo, no ofrecen mecanismos para que un grupo de usuarios pueda reproducir sus listas de reproducción de forma conjunta.

Aquí es donde entra en juego Spot&Joy, una aplicación desarrollada para sistemas operativos Android que permite a un grupo de usuarios reproducir música de forma conjunta en base a sus preferencias (listas de reproducción favoritas de *Spotify*, canciones buscadas en la plataforma…)

La aplicación permite efectuar la reproducción en un dispositivo concreto, permitiendo al resto de usuarios poder enviar canciones al mismo. A continuación, en la Figura 1, se muestra un escenario mostrando las interacciones de los usuarios.

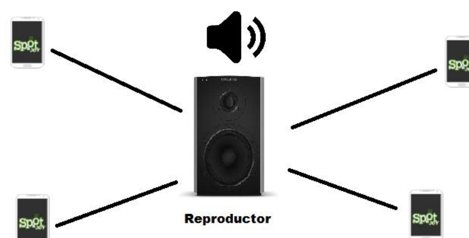

Figura 1. Escenario de Spot&Joy

2. OBJETIVOS

Se conoce el problema, se ha presentado una posible solución. A continuación, se listan los objetivos que se pretenden conseguir en este proyecto con el fin de dar una solución a la reproducción conjunta de un grupo de usuarios de *Spotify*.

Para ello, tal y como se ha mencionado en la introducción del presente documento, se pretende desarrollar una aplicación *Android* (orientada a dispositivos móviles) donde se debe permitir:

- **Crear listas de reproducción de forma conjunta**: los usuarios de la aplicación podrán crear una única lista de reproducción en base a las preferencias de todos ellos.
- **Reproducir música de forma conjunta**: la lista de reproducción conjunta definida en el punto anterior, podrá ser escuchada en un determinado dispositivo que hará de reproductor.
- **Obtener *playlists* de *Spotify***: Un usuario podrá buscar en todo momento las *playlists* o listas de reproducción que éste se encuentre siguiendo en *Spotify*. Pudiendo añadir las canciones de las mismas a la lista de reproducción conjunta.
- **Buscar contenido**: Además de obtener sus *playlists*, el usuario podrá buscar contenido en la plataforma Spotify bajo un criterio de búsqueda (tipo de contenido: Álbum, Playlist o Canción) y añadir los resultados a la lista de reproducción conjunta.
- **Visualizar estado actual de la reproducción**: todos los usuarios trabajando en el mismo contexto, podrán ver en todo momento el estado de la reproducción en su propio dispositivo: qué canción están siendo reproducida y cuáles sonarán a continuación.
- **Mezclador armónico**: las canciones a reproducir se intentarán reordenar en cuanto a su posicionamiento en la lista de reproducción conjunta se refiere intentando conseguir una reproducción lo más armónica posible.

3. ANTECEDENTES / ESTADO DEL ARTE

Analizados los objetivos generales que se pretenden conseguir, se pretende explicar a una profundidad mayor qué es la *Internet de las Cosas (Internet of Things, IoT*), cuándo surge y ejemplos reales de aplicaciones del mismo. También se hace mención que el presente proyecto se encuentra dentro de una línea de investigación, explicando de forma breve en qué consiste.

Por otro lado, se analizarán plataformas comerciales de propósitos similares a la aplicación propuesta en este documento, mostrando sus similitudes y sus diferencias.

3.1. Internet de las Cosas

3.1.1. Concepto

El término *Internet of Things* fue propuesto en el año 2009 por Kevin Ashton, ingeniero y empresario creador del centro de investigación Auto-ID de la *Universidad dde Massachusetts (MIT)* asegurando que este concepto podría cambiar al mundo al igual que Internet lo hizo anteriormente.

Teniendo en cuenta la gran cantidad de objetos o cosas que a día de hoy pueden ser conectados a Internet, se deduce que Internet como tal, establece un número de comunicaciones con cosas mayor el número de personas que se comunican. Según especialistas, se espera que en el año 2020 haya más de 20000 millones de dispositivos conectados a la red y que un 75% de los vehículos se encuentren conectados a Internet, o que en el año 2025 se generen ingresos superiores a los 3 billones de dólares gracias al *IoT*.

El objetivo principal que pretende conseguir *Internet of Things*, es el de conectar todos los objetos que nos rodean, siendo capaces de recoger información sobre ellos, procesarla y compartirla.

Algunos ejemplos podrían ser: Un sistema que permita recoger la información de unas zapatillas deportivas de un deportista, que permita procesar la información y permitir crear un programa de seguimiento. Una ciudad que pueda regular los semáforos de las zonas más concurridas de forma automática, o informar de rutas secundarias en caso de accidentes o incidencias…

Las aplicaciones son prácticamente infinitas, simplemente recogiendo datos de forma constante, consiguiendo información en tiempo real tanto para usuarios, como para otros dispositivos (digitalización del mundo físico)

Sin embargo, también se debe recalcar que, en base a las barreras tecnológicas actuales, el uso de toda esta información disponible se encuentra más que limitado (cuello de botella) aunque, como se verá a continuación, cada vez es mayor el número de aplicaciones de este concepto en el mundo real.

Nada puede tener una cantidad tan elevada de puntos a favor sin la aparición de ciertas contraposiciones. Por ejemplo, el ataque que sufrió Estados Unidos el pasado año, causando que gran parte del país quedase sin internet. Este suceso se acusa a la *Internet de las Cosas* (conjunto de dispositivos interconectados a Internet permitiendo incluso la interconexión entre ellos) Un determinado dispositivo conectado a la *Internet de las Cosas* puede no poseer los **controles de seguridad suficientes** por lo que son más fáciles de infectar que otros dispositivos. Durante el año transcurrido, éste número de ataques ha crecido hasta un 3%

En conclusión, este concepto aplicado a las tecnologías tiene un enorme futuro por delante, del que apenas hemos comenzado a rasgar sobre la punta de un enorme iceberg. La barrera existente entre dispositivos y usuarios se encuentra ensanchada precisamente a enfocar las aplicaciones *IoT* a tomar preferencias a nivel de usuario, descartando las más que posibles interacciones sociales que día a día éste podría llevar a cabo.

3.1.2. Aplicaciones en la actualidad

En el siguiente apartado, se intenta abarcar de forma breve algunos ejemplos de aplicaciones de *Internet of Things* en el mundo actual, intentando abarcar ámbitos muy distintos con el fin de mostrar cómo de grande es el alcance de las mismas a día de hoy.

- **Ciudades inteligentes**: monitorear las plazas de aparcamiento de la ciudad, las vibraciones de materiales en edificios, en puentes, en monumentos históricos… Evaluación de la energía radiada por estaciones base o routers Wi-Fi, el seguimiento en tiempo real tanto de vehículos como de peatones para optimizar rutas de conducción, etc.

- **Entornos inteligentes**: detección de incendios forestales monitorizando los gases de combustión y condiciones del fuego, control de las emisiones CO2, control de lugares para detección de temblores…

- **Seguridad y emergencias**: control de acceso en áreas restringidas y detención de intrusos, medición de la radiación de centrales nucleares para generar alertas de fuga o evaluación de niveles de gas y fugas en entornos industriales.

- **Agricultura inteligente:** mejorar la calidad del vino vigilando la humedad del suelo entre otras propiedades de los viñedos para así controlar el nivel de azúcar en las uvas y la salud de la propia vid. Control de condiciones climáticas para maximizar la producción de verduras y frutas o pronosticar formación de hielo, lluvia, sequías…

- **Hogar inteligente (domótica)**: gestión y optimización del uso de energía y agua, así como recomendar consejos sobre cómo ahorrar en base a dicha gestión, encendido o apagado de electrodomésticos de forma remota…

- **<u>Salud inteligente</u>**: detección de caídas, principalmente orientado a personas mayores o discapacitadas que vivan solas, control de condiciones de frigoríficos médicos (almacenamiento de vacunas, medicamentos, incluso de órganos).

Concluyendo este punto, se pueden encontrar innumerables usos en la actualidad para la *Internet de las Cosas* aun considerando que la barrera establecida sigue siendo bastante grande respecto a la inimaginable cantidad de aplicaciones que podrían llevarse a cabo. Poco a poco, el mundo continúa su digitalización.

3.1.3. Investigación

No se puede terminar de explicar las características de las plataformas *IoT* sin mencionar que el presente proyecto se enmarca dentro de una línea de investigación que, entre otra gente, los directores del proyecto han llevado a cabo. En él se plantea la necesidad de conseguir que los sistemas *IoT* permitan facilidades para conseguir que los usuarios puedan trabajar de forma conjunta permitiendo así una interacción social.

En la propuesta definen el modelo computacional *Contexto-Situacional* [17] como una posible solución, el cual debería hacer que los sistemas reajustasen sus comportamientos de forma automática, en base a las distintas preferencias de un determinado grupo de usuarios que se encuentren comunicados entre sí bajo un mismo contexto.

3.2. Otras aplicaciones sociales

Actualmente en el mercado se pueden encontrar diferentes aplicaciones que permitan la interacción entre usuarios para definir una *playlist* común en base a sus canciones o *playlist*. A continuación, se presentan algunas de ellas de forma breve con el fin de analizar sus funcionamientos.

FLO Music

Aplicación móvil disponible para sistemas operativos iOS que permite definir listas de reproducción sociales con canciones disponibles en las plataformas *Spotify* y *SoundCloud*. Permite guardar temporalmente las canciones permitiendo la opción de reproducción sin Wi-Fi. La lista de reproducción podrá ser gestionada y creada únicamente por un dispositivo que hará de *host*, teniendo éste el control total sobre la reproducción.

Esta aplicación difiere respecto a la explicada en el presente documento en aspectos como puede ser la política de reproducción donde, inserta las canciones en la lista de forma continua según van llegando, a diferencia de *Spot&Joy* que buscará gestionar la lista para conseguir una reproducción lo más armónica posible.

A continuación, puede observarse una captura de pantalla de la misma en funcionamiento en la Figura 2:

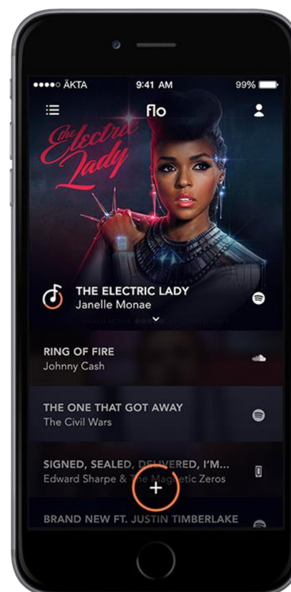

Figura 2. Captura de pantalla de FLO Music [18]

Outloud

Esta aplicación se encuentra disponible tanto para sistemas operativos *Android* como *iOS*. Permite añadir canciones a una playlist común. Las canciones se insertarán en la lista según vayan llegando y, su orden podrá ser alterado mediante un sistema de votación tal que aquella canción con más votos, será la siguiente en reproducir.

Aunque en lo referente al objetivo principal, esta aplicación marca el mismo objetivo principal que la presente aplicación, su política de reproducción es completamente diferente.

El curso actual de la reproducción es mostrado desde un portal web, que muestra toda la actividad de la lista de reproducción: ver próximas canciones, últimas añadidas, mejores DJs de la fiesta (usuarios cuyas canciones introducidas han sido las más votadas) …

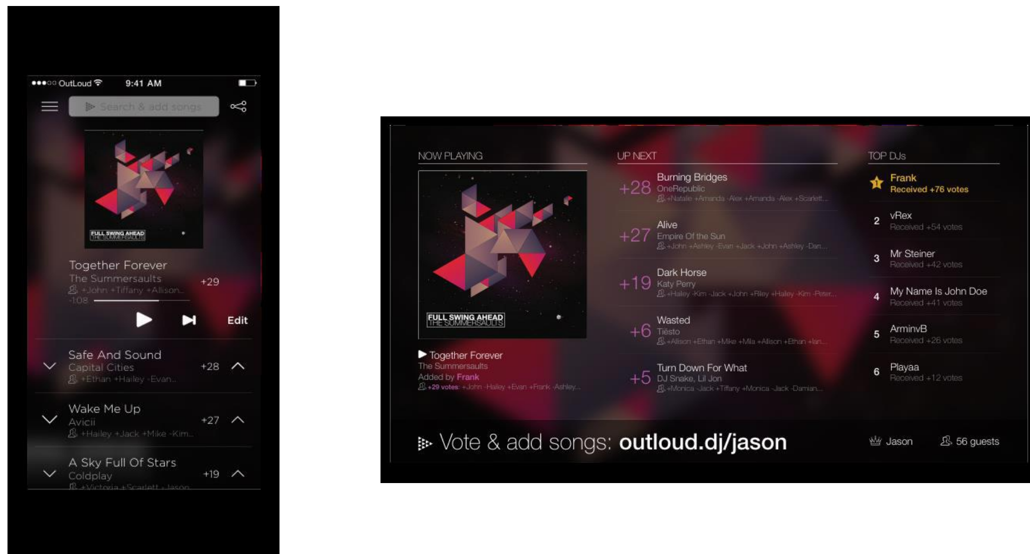

Figura 3. Captura de pantalla de Outloud [19]

4. MÉTODOLOGÍA

Una vez explicado el problema, fijados los objetivos para abarcar la solución, en el presente apartado se pretende abordar de forma genérica y breve las distintas etapas que han tenido presencia durante toda la gestación del proyecto, así como conocer los recursos que han sido utilizados.

4.1. Fases de realización

A continuación, se pretende explicar cómo se ha abordado la solución del problema. Nos encontramos ante un problema cuya solución consiste en un desarrollo *software*, por lo que se ha optado dividir el proceso en una serie de etapas utilizando como guía las fases de desarrollo *software* estudiadas a lo largo de la titulación: *análisis, diseño, implementación, pruebas y documentació*n.

Las etapas permiten abordar la solución del problema desde rasgos más generales, hasta llegar a los engranajes más específicos.

A continuación, se describe un breve resumen de cada una de las diferentes fases. Posteriormente, a cada una de ellas se les dedicará un apartado completo de la documentación.

- **Análisis**: Definir una lista de objetivos a conseguir y en base a ellos, fijar un conjunto de requisitos que detallen qué tareas debe realizar la aplicación y cómo deberá hacerlo. En base a dichos requisitos, definir casos de uso que muestren las tareas necesarias para conseguir los objetivos. Posteriormente se realizan unos prototipos de maquetas que muestren una posible navegabilidad de la plataforma. Se debe realizar un estudio del mercado actual en lo referente a aplicaciones similares. A raíz de los objetivos prefijados a cumplir, estudiar las tecnologías candidatas que ayuden a cumplir con los mismos (desarrollo de Spotify en Android, plataformas que permitan algún tipo de comunicación entre dispositivos…) Una vez conocidas, se deben seleccionar las tecnologías candidatas que ayuden a cumplir con los mismos (desarrollo de *Spotify* en Android, plataformas que permitan algún tipo de comunicación entre dispositivos…)

- **Diseño y Arquitectura:** en base al diagrama de componentes y a los prototipos maqueta, se procede a realizar un diseño de la arquitectura de la aplicación que pueda cumplir con los objetivos. Definiendo aspectos como el diseño de las pantallas, la interfaz, servicios o comunicaciones. Se pretende dotar a nuestra aplicación de un esqueleto que le sirva de arquitectura para posteriormente proceder a su realización.

- **Implementación:** desarrollo de la aplicación Android utilizando el lenguaje de programación Java y apoyándose en el diseño y la arquitectura previamente seleccionada.

- **Pruebas:** permiten verificar que todos y cada uno de los requisitos necesarios cumplen y garantizan su correcto funcionamiento en base a las especificaciones.

- **Documentación:** Tras la finalización y validación de todas las fases anteriores, se procede a documentar tanto de forma interna a nivel del código desarrollado, como de forma externa, tal y como es reflejado en este documento. Se pretende dar constancia por escrito del proceso llevado a cabo para realizar éste trabajo.

Con esto no se pretende decir que el desarrollo de la aplicación haya sido desarrollar cada fase en su totalidad de forma seguida.

A medida que las distintas fases eran abordadas, siempre aparecían nuevas necesidades, nuevos requisitos u objetivos, descartar otros… El proyecto evolucionaba (aparecen nuevos requisitos no especificados, hay que marcar nuevos objetivos, necesidad de integrar una nueva vista o incluso una nueva plataforma…), requería grandes cambios y para ello, se debían retomar etapas anteriores. Para ello, se utiliza la retroalimentación como estrategia a seguir durante las cuatro primeras fases que se mencionan en este apartado

A continuación, se muestra la **evolución** del desarrollo de la aplicación, abordando desde la idea en papel hasta su implementación final para dispositivos Android en la Figura 4.

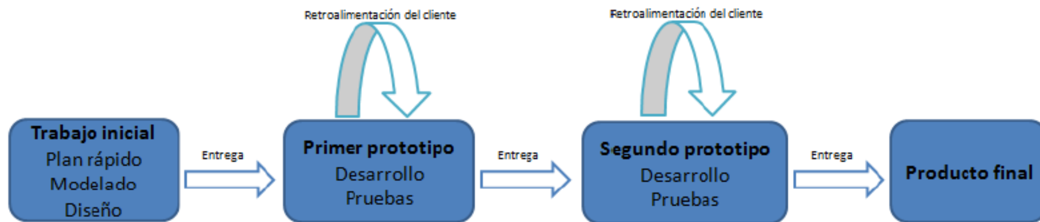

Figura 4. Estrategia de desarrollo

Para concluir este punto, se procede a exponer los temas abordados en cada una de las etapas identificadas en la Figura 4.

- **Trabajo inicial**: se plantea el problema y se busca una solución posible. Se definen objetivos, una posible maqueta inicial que permita definir una serie de vistas que permitan realizar la aplicación. Se selecciona ***Android***, como sistema operativo. Se establece el *smartphone* como el dispositivo al que se orientará la solución, así como los dispositivos *raspberrys* como dispositivos de reproducción.
- **Primer prototipo:** cambia el concepto de la aplicación, descartando *raspberry* como dispositivo reproductor y orientando todas las funcionalidades para *smartphones*. Se definen todas aquellas funcionalidades que permitan efectuar tareas **monousuario**: capacidad de reproducir música, conectar con *Spotify*, cargar listas del usuario y reproducirlas… pero no permite una reproducción conjunta, tampoco se considera añadir un buscador…

- **Segundo prototipo:** se incorporan las tareas **multiusuario**, aquellas que proporcionan mecanismos de envío de canciones entre dispositivos. Se desarrolla la gestión de usuarios y es llevada a cabo por el usuario administrador de la sesión. Se incorpora un elemento buscador de canciones a añadir o enviar al reproductor. El cual se limita a encolar las canciones según van siendo seleccionadas. **Se permite definir al fin una lista de reproducción social** con las canciones que los usuarios van enviando.

- **Producto final:** Se reajusta el diseño de las vistas, y la política de reproducción que permite encolar las canciones en la lista de una determinada manera, cambia. Se define un mezclador que intente encolar las canciones en la lista según su armonía, intentando crear una reproducción armónica dentro de lo posible. Finalmente se "limpia" el código y se documenta.

4.2. Recursos disponibles

Se describen a continuación los recursos hardware que han sido utilizado a lo largo de todas las fases de desarrollo de la aplicación, indicando sus características.

4.2.1. PC Portátil Asus GL552VW-DM151

Utilizado durante todo el desarrollo del proyecto, desde la recolección de información, hasta llevar a cabo la implementación de la aplicación o escribir el presente documento.

Sus características físicas son las siguientes:

- Procesador: Intel Core i7-6700HQ
- Memoria RAM: 16GB (8GB*2) DDR4 2133MHz
- Almacenamiento: 128GB SSD SATA3+1TB 7200rpm SATA
- Gráficos: NVIDIA GeForce GTX9600M 4GB GDDR5 VRAM
- Sistema operativo: Windows 10 Home (64 bits)

4.2.2. Smartphones

Han servido para poder ir realizando las pruebas software de la aplicación, probando distintos escenarios que permitan verificar el correcto funcionamiento de la aplicación. Este punto se retoma de forma detallada en el apartado *5.4. Pruebas*.

Los dispositivos móviles utilizados han sido los siguientes:

4.2.2.1. Huawei Y635-L01

Smartphone principal utilizado a la hora de desarrollar la aplicación. Ha participado durante todo el proceso de implementación de la aplicación.

Sus características físicas son las siguientes:

- CPU: Procesador Qualcomm Snapdragon 1,2GHz
- Memoria RAM: 1GB
- Almacenamiento: 8GB
- Dimensión de pantalla: 5 pulgadas
- Resolución de pantalla: 854x480 píxeles
- Versión de Android: 4.4.4 (Kit-Kat)

4.2.2.2. Motorola Moto E2 4G LTE (dos dispositivos)

Dos dispositivos móviles con las mismas características y modelo, proporcionados por el tutor de este proyecto con la finalidad de probar las funcionalidades multiusuario de la aplicación, como puede ser la petición de conexión a una sesión o el envío de canciones a la misma.

Sus características físicas son las siguientes:

- CPU: Procesador Qualcomm Snapdragon 1,2GHz
- Memoria RAM: 1GB
- Almacenamiento: 8GB
- Dimensión de pantalla: 4.5 pulgadas
- Resolución de pantalla: 960x540 píxeles
- Versión de Android: 4.4.4 (Kit-Kat)

5. IMPLEMENTACIÓN Y DESARROLLO

Una vez explicado en escala general el proceso llevado a cabo para la realización del presente proyecto, se pretende explicar, una a una, todas las fases mencionadas en el apartado anterior, a una escala mayor, desglosando cada una de ellas en una serie de puntos e introduciendo así al lector en todo el proceso de desarrollo de una forma más profunda. A partir de este momento, cada punto de este apartado hará referencia al proceso completo efectuado para una determinada etapa.

5.1. Análisis

En el siguiente apartado se pretende explicar todo el desarrollo de la fase de análisis utilizado en el presente proyecto. Abarcando desde la fijación de los objetivos, hasta el estudio y la selección de tecnologías que ayuden a llevar a cabo la realización de la aplicación. Se establece como medio donde aplicar la solución el dispositivo móvil *smartphone* ya que es de los más comunes y el dispositivo *raspberry* como reproductor de canciones.

Una vez conocidos los propósitos, se establecen los casos de uso que serán necesarios para conocer qué tareas deberá realizar el sistema de cara a cumplir con los requerimientos especificados

También se ha abordado una pequeña parte de diseño, referente al desarrollo de una maqueta no funcional que sirva de guía a la hora de plantear qué podrá hacer y que no la aplicación, con el fin de identificar los requisitos (tanto funcionales como de sistema)

Los requisitos a su vez, ayudan a definir una serie de casos de uso que muestren las tareas que el usuario deberá realizar para cada funcionalidad.

Se define una serie de componentes, así como la interactuación necesaria entre ellos que cumpla con los requisitos y objetivos identificados. Se realiza pues, un diagrama de componentes.

Como se mencionaba en el punto *4. Metodología*, la retroalimentación produce modificaciones en algunos aspectos, como ha podido ser el hecho de orientar la aplicación en su totalidad a *smartphones*, evitando así la complejidad de tener que desarrollar dos aplicaciones distintas para dos sistemas distintos.

A continuación, se pretende introducir al lector más allá, abordando cada uno de los puntos más importantes que se mencionan en esta fase de *Análisis* para abordarlos con una profundidad mayor.

5.1.1. Requisitos

Se debe especificar el comportamiento que se espera de un determinado proyecto *software*. En base al estudio de aplicaciones similares, se establece una serie de requisitos que se consideran **indispensables** para el proyecto. Para continuar, se enumeran y se describen de forma breve los requisitos establecidos para el diseño y desarrollo de la aplicación.

5.1.1.1. Requisitos funcionales

Describen las interacciones de los usuarios con el sistema.

RF1: Crear una reproducción conjunta

Un usuario podrá crear una lista reproducción compuesta por las canciones proporcionadas por otros usuarios o él mismo.

RF2: Reproducir canciones

El usuario que posea la lista de reproducción conjunta, podrá escuchar las canciones en su dispositivo.

RF3: Cargar playlists de Spotify

Cualquier usuario podrá tener acceso a las listas que sigue en la plataforma Spotify.

RF4: Buscar una canción

Todos los usuarios podrán buscar canciones en la plataforma Spotify.

RF5: Enviar canciones

Cada usuario podrá enviar canciones a la lista de reproducción conjunta.

RF6: Identificar reproducciones

Un usuario podrá buscar dispositivos que estén reproduciendo listas conjuntas en su entorno próximo.

RF7: Ver estado de la reproducción

Ver qué canción está en reproducción y cuáles son las próximas a estarlo (si las hay)

RF8: Identificación

El usuario accederá a la aplicación mediante su cuenta Gmail.

RF9: Identificación de Spotify

El usuario accederá a las funcionalidades de *Spotify* cuando se haya identificado en la plataforma.

RF10: Salir de la sesión

Cada usuario podrá salir de una sesión de reproducción conjunta en cualquier momento.

5.1.1.2. Requisitos no funcionales

Requisitos complementarios o atributos de calidad. Especifican ciertos criterios que juzgan determinados comportamientos del sistema.

RNF1: Concurrencia

1. Gestión adecuada del tráfico de envío de mensajes.
2. Se debe garantizar que todos los dispositivos conectados a la sesión puedan enviar canciones al reproductor de forma continua.
3. El administrador debe poder recibir todas las canciones enviadas por todos los usuarios en cualquier punto de la sesión.

RNF2: Accesibilidad y usabilidad

1. La interfaz gráfica debe ser sencilla, elegante e intuitiva.
2. Debe suponer un esfuerzo **mínimo** y ningún tipo de impedimento por parte del usuario que haga uso de la aplicación.

RNF3: Escalabilidad

1. La aplicación no debe mermar sus capacidades ante un número elevado de usuarios.

RNF4: Seguridad

1. La información referente a usuarios nunca será almacenada por los administradores. Sus datos perdurarán en la sesión el tiempo que éstos se encuentren conectados.
2. Solamente habrá que *loguearse* como usuario de *Spotify* una vez.
3. El administrador podrá expulsar a cualquier usuario en cualquier momento, evitando así la presencia de intrusos (si se utilizase la aplicación en un ambiente público y abierto)

5.1.2. Casos de uso

A continuación, se especifica una serie de casos de uso que permitan resolver los distintos objetivos en base a los requisitos especificados en el punto anterior.

Para ello, se definen dos roles, que definen a los tipos de actores que podrán aparecer durante un momento u otro en la aplicación.

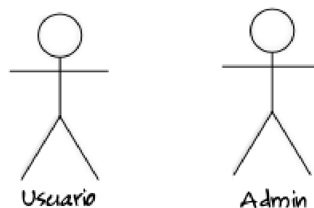

Figura 5. Actores

Primeramente, se expone el caso general. Mostrando las tareas que todos los usuarios tienen a su disposición cuando ejecutan la aplicación. El rol *Usuario*, en este caso, se utiliza para identificar a **cualquier usuario** de la aplicación.

A continuación, se muestra el diagrama de casos de uso correspondiente en la figura 6:

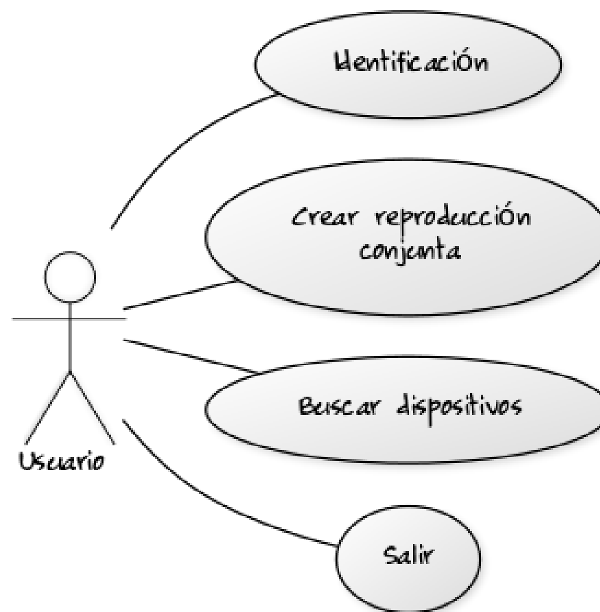

Figura 6. Diagrama de Casos de Uso (1)

Una vez el usuario se haya identificado, podrá crear una sesión para reproducción compartida, buscar dispositivos de otros usuarios que ya estén llevando a cabo una reproducción o salir de la aplicación.

Salir: simplemente, el usuario identificado dejará de la aplicación.

Crear una reproducción conjunta: permite al usuario iniciar una sesión de reproducción. Se asigna al usuario el rol **Administrador**.

Buscar dispositivos: permite al usuario buscar dispositivos que hayan iniciado una sesión de reproducción disponible en sus cercanías. Una vez se hayan encontrado resultados, el usuario podrá elegir a cuál entrar. El usuario mantiene su rol **Usuario**.

A partir de este punto, se desglosan dos funcionalidades distintas dependiendo del rol seleccionado.

Casos de uso - Administrador

Se especifican los casos de uso disponibles para un usuario Administrador.

- Mostrar playlists: el administrador tendrá acceso directo a las listas de reproducción que siga en Spotify.
- Buscar contenido: el administrador tendrá acceso a un buscador de contenido de *Spotify*.
- Ver reproducción: podrá tener acceso a ver el estado actual de la reproducción.
- Enviar a reproducir: el administrador podrá seleccionar una playlist de su lista y enviar las canciones a reproducir. Por otro lado, en base a los resultados encontrados de buscar contenido en Spotify, podrá seleccionar los mismos para que sean reproducidos.
- Gestionar usuarios: el administrador tendrá la opción de gestionar los usuarios conectados a su sesión, permitiendo seleccionar uno de ellos y proceder a eliminarlo de la misma.
- Cerrar sesión: el administrador tiene acceso a poder cerrar su sesión de reproducción.

A continuación, se muestra en la Figura 7, el diagrama de casos de uso del Administrador.

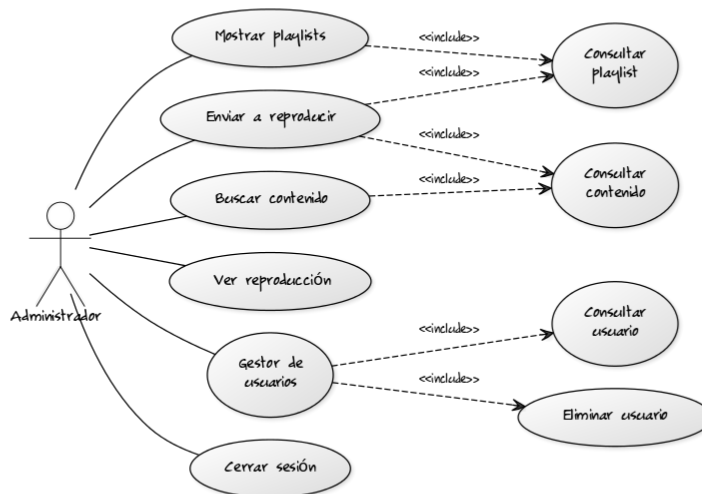

Figura 7. Diagrama de Casos de Uso (2)

Casos de uso – Usuario

Se especifican los casos de uso disponibles para un **usuario estándar**. Aunque en general el proceso a seguir para realizar las distintas funcionalidades es muy similar a los casos definidos para el rol *Administrador*, sí que existen ciertas diferencias: el usuario de una sesión no tendrá acceso a un gestor de usuarios (tarea **exclusiva** del **administrador**) y, por otro lado, **no podrá cerrar nunca la sesión**, sino que simplemente, saldrá de ella. La sesión continuará hasta que así lo quiera el **administrador**.

Por otro lado, las canciones deseadas serán enviadas al reproductor del usuario administrador, el cual notificará los cambios en todos los dispositivos conectados a su reproducción posteriormente (los cambios de la lista de reproducción actual).

A continuación, se muestra en la Figura 8, el diagrama de casos de uso para usuarios estándares.

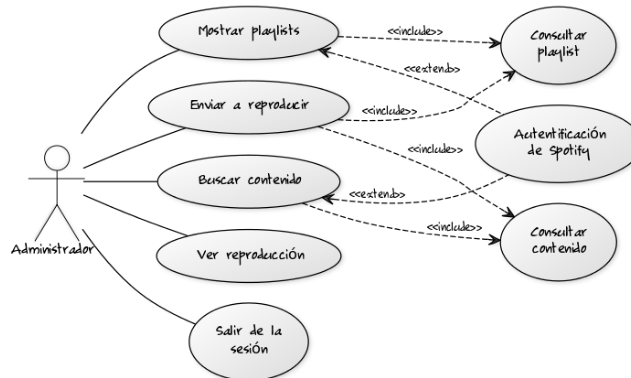

Figura 8. Diagrama de Casos de Uso (3)

5.1.3. Prototipo y componentes del proyecto

Una vez especificados todos aquellos requisitos necesarios para la realización de este proyecto, se procede a trasladar las funcionalidades a una maqueta prototipo cuyo diseño de interfaz permita cumplir los requisitos propuestos en el apartado anterior.

Partiendo de la base de que el desarrollo de este proyecto ha sido mediante retroalimentación en cada una de sus fases, a continuación, se muestra el diseño que inicialmente se consideró en base a unos requisitos que fueron evolucionando.

5.1.3.1. Construcción del prototipo

La idea del prototipo es poder fijar los requisitos definidos en un posible ejemplo, mostrando a su vez cómo estarían todas las operaciones que serán implementadas posteriormente conectadas entre sí y cómo podría ser posible la navegación entre un posible conjunto de vistas.

Partiendo de la idea previamente descrita, se pretende comenzar a idear una aplicación que cumpla con tal objetivo. Esto supone la aparición de otras muchas ideas que permitan enriquecer a la misma, o que algunas de éstas evolucionen o cambien en base a cómo ha ido evolucionando el proyecto durante su desarrollo.

Se llega a la conclusión de que el sistema debe de estar compuesto por dos partes bien diferenciadas: una administradora y otra estándar.

Teniendo en cuenta que el reproductor debe ser un determinado usuario, dicho usuario deberá poseer privilegios sobre su dispositivo móvil (será el que reproduzca la música) o incluso sobre el resto de usuarios de la sesión.

La primera versión de la lista de objetivos a conseguir por la aplicación sería la siguiente:

- Autentificación de usuarios de la aplicación
- Formulario para registros de usuario (**descartado**)
- Mostrar qué canción suena y cuáles vendrán a continuación.
- Posibilidad de pausar o saltar canción (**descartado**) (sólo para el administrador)
- Desconexión de la sesión

Algunos ejemplos sobre este aspecto pueden ser los siguientes:

- Contemplados inicialmente (**descartados** posteriormente)
    - Formulario de registro para usuarios
    - Gestión y definición de otras políticas de reproducción
- No contemplados inicialmente (incorporados posteriormente)
    - Buscador de canciones en base a una categoría (playlist, álbum, canción…)
    - Acceso Login vía Spotify

Como herramienta de ayuda para esta primera fase, se ha hecho uso de la plataforma **NinjaMock**, que permite crear *wireframes* para móviles.

A continuación, se procede a mostrar las imágenes referentes al primer prototipo de la aplicación, ya que fue el que sirvió de guía para en fases posteriores definir las vistas reales de la misma.

En la figura 9, se muestra el diseño prototipo del sistema de autentificación, que a su vez hará de pantalla principal de la aplicación. A su lado, se encuentra el diseño de la pantalla de registro de usuarios.

Figura 9. Mockup (Inicio y registro)

En la figura 10, puede verse una pantalla donde se podrán buscar “Salas de reproducción” (reproducciones en curso por parte de algunos dispositivos)

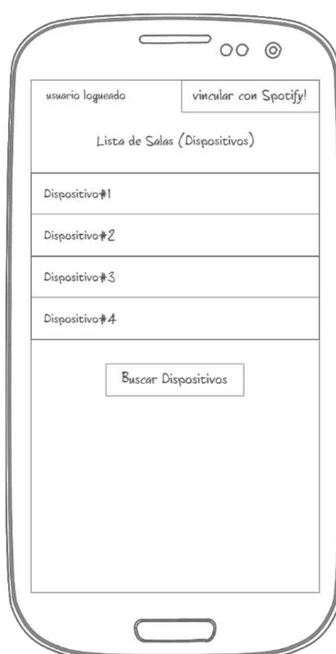

Figura 10. Mockup (Lista de dispositivos)

Cuando un dispositivo móvil accediese a una sala de reproducción por primera vez, éste pasaría a ser el administrador de la sala y a seleccionar una determinada política de reproducción, tal y como se ve en la figura 11.

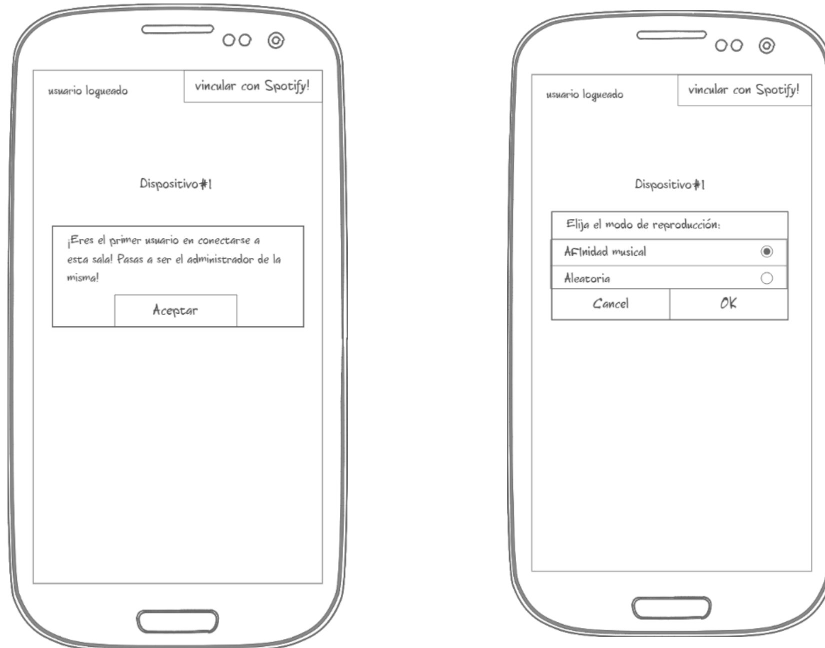

Figura 11. Mockup (Gestión del administrador)

En la figura 12, se puede apreciar cómo sería la vista de la reproducción desde un dispositivo móvil administrador.

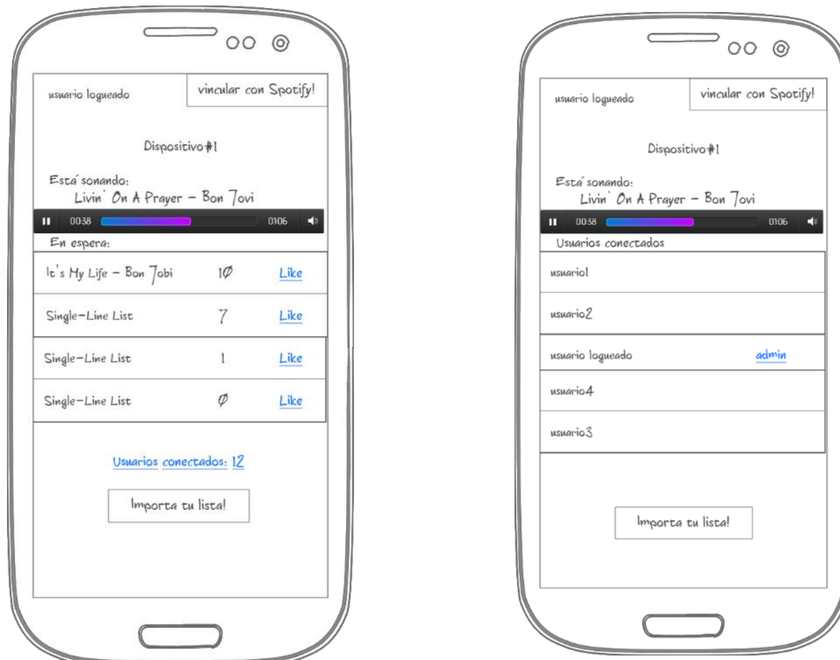

Figura 12. Mockup (Vista de reproducción del administrador)

En la figura 13, se puede apreciar cómo sería la vista de la reproducción desde un dispositivo móvil estándar

Figura 13. Mockup (Vista de reproducción de usuario estándar)

5.1.3.2. Identificación de componentes de la aplicación

A raíz del prototipo inicial, se pretende identificar qué componentes software son necesarios para poder llevar a cabo todos los objetivos propuestos. Esto supone un primer acercamiento a lo que viene a ser el futuro desarrollo de software, y de las decisiones tomadas en esta fase, dependerá que fases posteriores de la aplicación se hayan llevado a cabo de una determinada manera.

- Gestor de usuarios: gestor para los usuarios de la plataforma, tiene control sobre su autentificación.
- Gestor administrador: gestiona la sesión que haga de administrador, garantizando el control de todos los usuarios vinculados a la sesión, o la interacción del sistema administrador con la lista de reproducción.
- Gestor de Spotify: encargado de gestionar las canciones, la obtención de las mismas en base a un determinado identificador y la inserción de ellas en el reproductor. También se encarga del control de credenciales necesarias para tener acceso a nuestra cuenta de Spotify.
- Gestor de búsquedas: permite llevar a cabo la búsqueda de canciones, álbumes o listas de reproducciones en la plataforma Spotify.
- Gestor de reproducción: encargado de gestionar la lista de reproducción, permitiendo la incursión de las mismas siguiendo la política marcada por el algoritmo de mezcla o su actualización cada vez que termine la reproducción de cada canción.
- Gestor de conexiones: se ocupa de llevar a cabo el control de las conexiones que permite la interacción multiusuario de la aplicación, desde la búsqueda de dispositivos, solicitudes de acceso o desconexión a otros, o el envío de canciones.

5.1.3.3. Diagrama de componentes

Los requisitos han marcado las **funcionalidades** de la aplicación a conseguir. El prototipo y los casos de uso, una posible navegabilidad y el orden de ejecución que podría utilizarse.

En este apartado se pretende mostrar qué componentes serían necesarios a nivel de *software* para poder definir la arquitectura de la presente aplicación. Gracias a esto, podemos tener una idea inicial referente a información que será necesaria para nuestro sistema para cumplir con los objetivos propuestos, como saber qué identificadores serán necesarios o qué componentes interactuarán entre sí para cumplir con un determinado requisito.

A continuación, se muestra el diagrama de componentes de Spot&Joy (Figura 14).

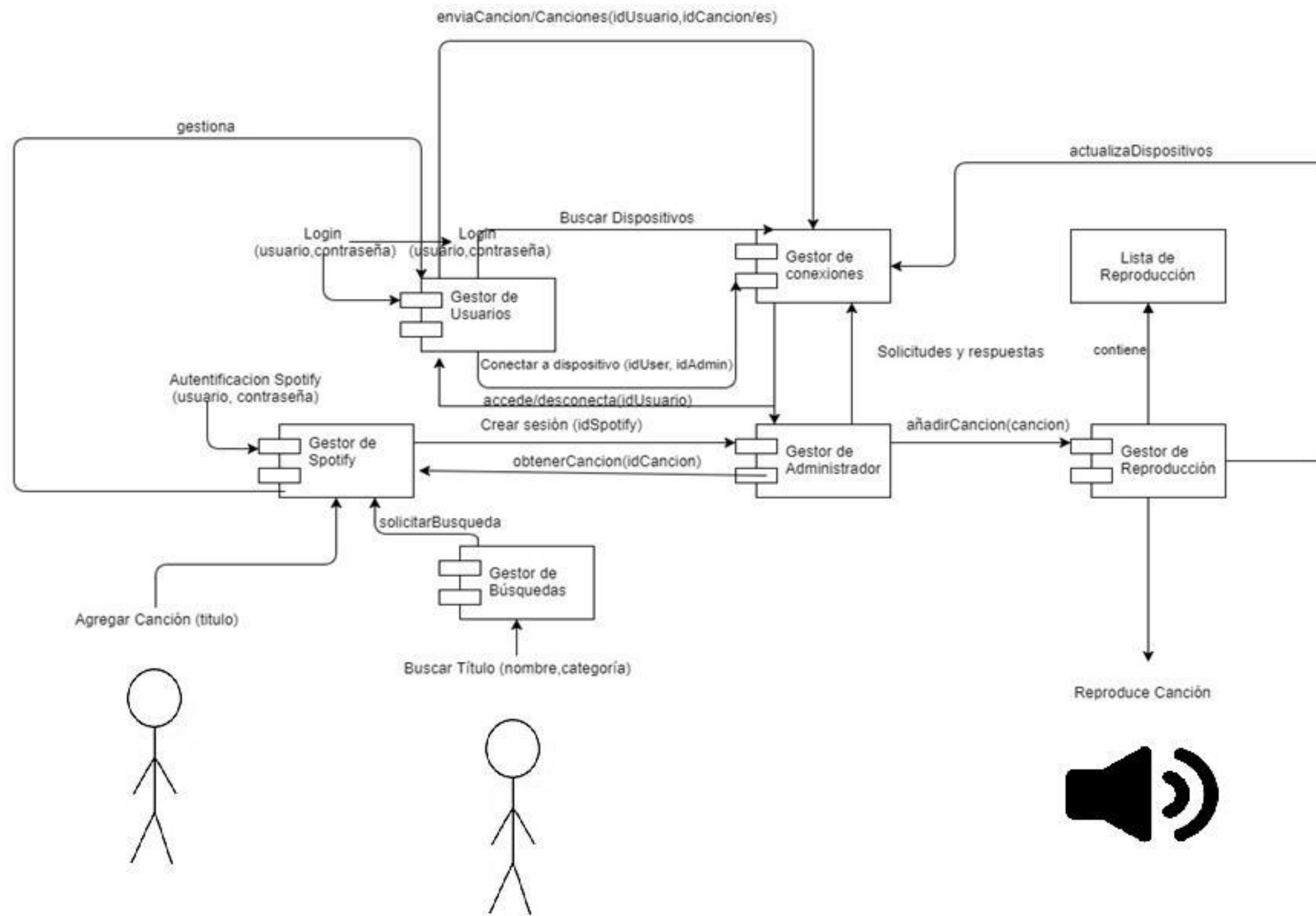

Figura 14. Diagrama de componentes

Como se menciona anteriormente, y concebido el diagrama, se comienza a pensar en el desarrollo de los elementos con los que interactuarán los componentes, es decir, qué parámetros mínimos serán necesarios para poder llevar a cabo la interconexión propuesta.

Entre las conclusiones obtenidas, destacan las conexiones entre los dispositivos que, aun siendo muy genéricas en esta etapa del desarrollo, se empieza a enfocar a un tipo de comunicación donde peticiones y respuestas marcarán las pautas. Todo lo referente a las mismas es tratado en puntos posteriores.

En lo que se refiere al envío de canciones, se llega a la conclusión de la necesidad de trabajar con identificadores únicos para las mismas, y que esta sea la información a enviar (que no sea necesario enviar la canción completa al dispositivo reproductor), y que el sistema pueda obtener una canción en base a dicho identificador. Las comunicaciones deben permitir el envío tanto de una canción, como de un grupo de canciones (sus identificadores).

Se llega a la conclusión de que el sistema tendrá dos servicios de autentificación disponible, uno para la propia aplicación, y otro para Spotify. Se recuerda que uno de los objetivos a conseguir no es otro que el de que cualquier usuario de la aplicación pueda acceder a una determinada sesión y poder consultar qué está siendo reproducido, incluso qué se reproducirá próximamente en caso de haber más canciones en la lista de reproducción. El acceso a Spotify debe permitir habilitar el resto de funcionalidades a un usuario estándar (búsqueda de canciones, consulta de listas o envío de canciones) o poder crear una determinada sesión (como se verá posteriormente en la fase de estudio, la cuenta de dicho usuario en la plataforma debe ser Premium)

5.1.4. Herramientas de desarrollo

En este apartado, se explicará qué herramientas de desarrollo software han sido empleadas para llevar a cabo el desarrollo del proyecto.

5.1.4.1. Android Studio

Concretamente se ha hecho uso de Android Studio, que desde 2014 se trata del IDE (Integrated Development Eviroment o Entorno de Desarrollo Integrado) para Android, reemplazando a Eclipse, que hasta la fecha se trataba del IDE oficial, e incorporando nuevas características que éste no poseía, como pueden ser:

- Construcción de proyectos usando la herramienta Gradle, que permite automatizar la construcción de los proyectos.
- Previsualización de layout para varios tipos de dispositivos de forma simultánea.
- Facilidades para pruebas de testeo basadas en JUnit.
- Importación de ejemplos con código o proyectos completos desde la plataforma **GitHub**.

Android Studio también permite crear dispositivos virtuales que emulen a dispositivos móviles, incluso personalizar las características y dimensiones del mismo. Este tipo de utilidades **no han sido empleadas para el desarrollo de este proyecto**. El desarrollo del mismo ha sido **siempre** probado en dispositivos móviles reales.

5.1.4.1.1. Seleccionando versiones

Android Studio nos puede beneficiar en el sentido de proporcionar información actualizada (dependerá lógicamente de si tenemos actualizada la propia herramienta o no) respecto al porcentaje de dispositivos existentes por versión.

Continuando en la misma pantalla será la de seleccionar la opción *Help Me Choose*, y aparecerá un gráfico similar al de la figura 15.

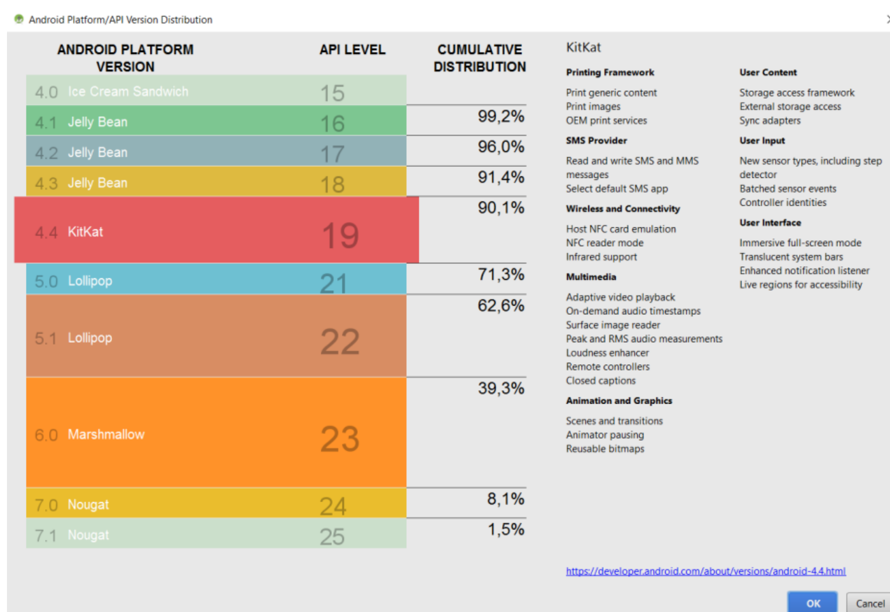

Figura 15. Gráfico de uso de APIs de Android

Tal y como se aprecia en el gráfico, seleccionando la API 19 como mínima, se conseguiría englobar a un 90.1% de todos los usuarios y teniendo en cuenta que el resto de versiones anteriores a la misma sólo englobarían a un porcentaje del 9.8% (siendo este porcentaje menor cada año hasta el punto de que llegue a ser 0), se han descartado versiones actuales.

Llegados a este punto, se estima que la *SDK mínima* para este proyecto será la API 19.

Concretamente, el proyecto ha sido desarrollado bajo la **API versión 25** y con una API mínima versión 19.

5.1.5. Estudio de plataformas

Se conoce qué entorno se va a utilizar, se posee la herramienta, se posee un proyecto base al que se le van a agregar los distintos componentes definidos en apartados anteriores, implementándolos de forma precisa para que cumplan con sus objetivos y especificaciones predefinidas. Pero antes de todo eso, se debe conocer qué elementos software se van a utilizar como base para implementar dichos componentes.

Por ejemplo, se sabe que las listas de reproducción serán cargadas desde Spotify, o que las canciones podrán ser enviadas entre móviles. Se debe comenzar un proceso de investigación para conocer qué librerías, APIs y plataformas serán necesarias y que ayuden a definir los componentes.

5.1.5.1. Google Sign-In

Aunque no esté relacionada con los ejemplos anteriores, sí que puede ser beneficioso para la aplicación el utilizar las herramientas ofrecidas por Google y que nos permiten definir un sistema de autentificación basado en Gmail.

¿Por qué esto en lugar de definir un sistema de registro de usuarios y de autentificación propio para la aplicación? Pues básicamente debido a que se intenta aprovechar una de las cualidades de cada dispositivo móvil que posea sistema operativo Android, el cual siempre debe llevar asociada una cuenta Gmail y por tanto, nuestro sistema de *logueo* para la aplicación pedirá seleccionar con qué cuenta se quiere acceder (en caso de tener más de una registrada para el dispositivo)

5.1.5.1.1. Obtención de archivos necesarios

Se accede a la página web de Google Developers, concretamente a la siguiente dirección: https://developers.google.com/identity/sign-in/android/start-integrating

En dicha dirección, lo primero que se encuentra es una lista de prerrequisitos a cumplir a la hora de poder configurar nuestro proyecto para que de soporte a esta funcionalidad de Google para nuestro proyecto, donde se destaca que la versión mínima del proyecto Android sea la 2.3 (Gingerbread) o que se tenga descargado Google Play Services SDK.

Lo siguiente es conseguir un archivo de configuración para nuestro proyecto. Para ello se selecciona la opción disponible, la cual viene disponible después de los prerrequisitos, ***Get a Configuration File***.

Se accede a una pantalla de configuración, donde se debe insertar qué nombre vamos a dar a nuestro proyecto dentro de la plataforma **Firebase** (ofrece múltiples alternativas como se verá en punto posteriores), y su respectivo paquete Gmail, tal y como puede verse en la figura 16.

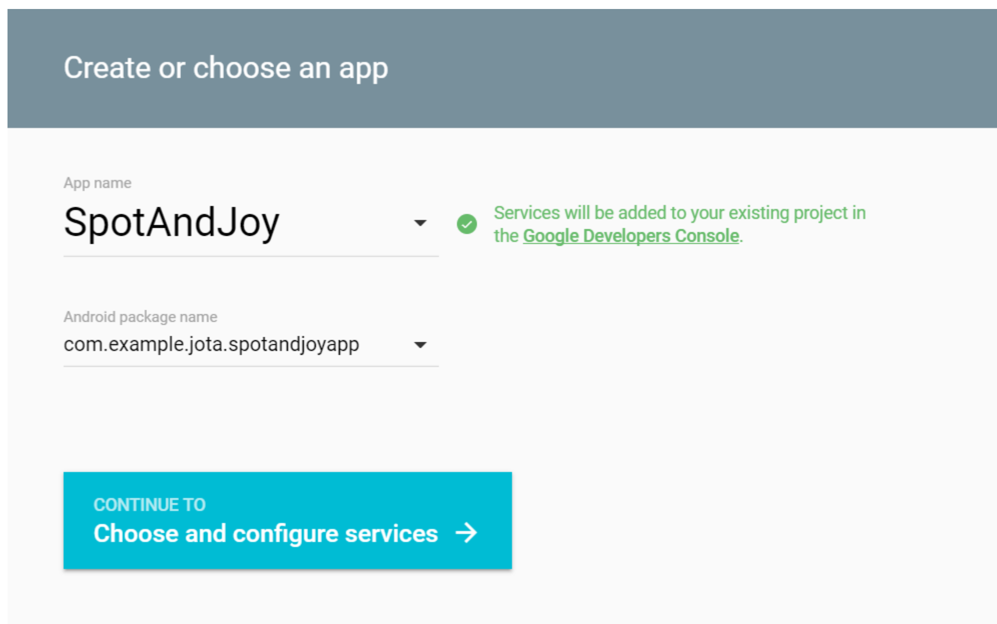

Figura 16. Google Sign In (Creando una aplicación)

Después de rellenar dichos campos, seleccionamos ***Continue to Choose and configure services***.

En la siguiente pantalla se debe seleccionar aquellos servicios que se quieren obtener de Google. Para este caso, se han seleccionado ***Cloud Messaging*** (se explicará en apartados posteriores el porqué) y ***Google Sign-In***.

Para poder generar un archivo de configuración, se debe introducir el certificado **SHA-1** (Secure Hash Algorithm o Algoritmo de Hash Seguro) de nuestro proyecto (se trata de su huella digital).

Para conocer el SHA-1 de nuestro proyecto basta con seleccionar la pestaña Gradle disponible en la parte superior derecha de la pantalla de trabajo de Android Sudio, seleccionar el módulo **":app"** y ahora, en el sistema de recursos disponibles ir a *Tasks* > *android* y ejecutar la opción **signinReport**.

Se puede ver que, de forma automática, el sistema ha lanzado un mensaje que muestra en la parte de consola de Android Studio, donde se muestra una determinada información referente al proyecto, incluyendo el SHA-1. A continuación, se muestra en la figura 17, el resultado de obtención del SHA-1 del presente proyecto.

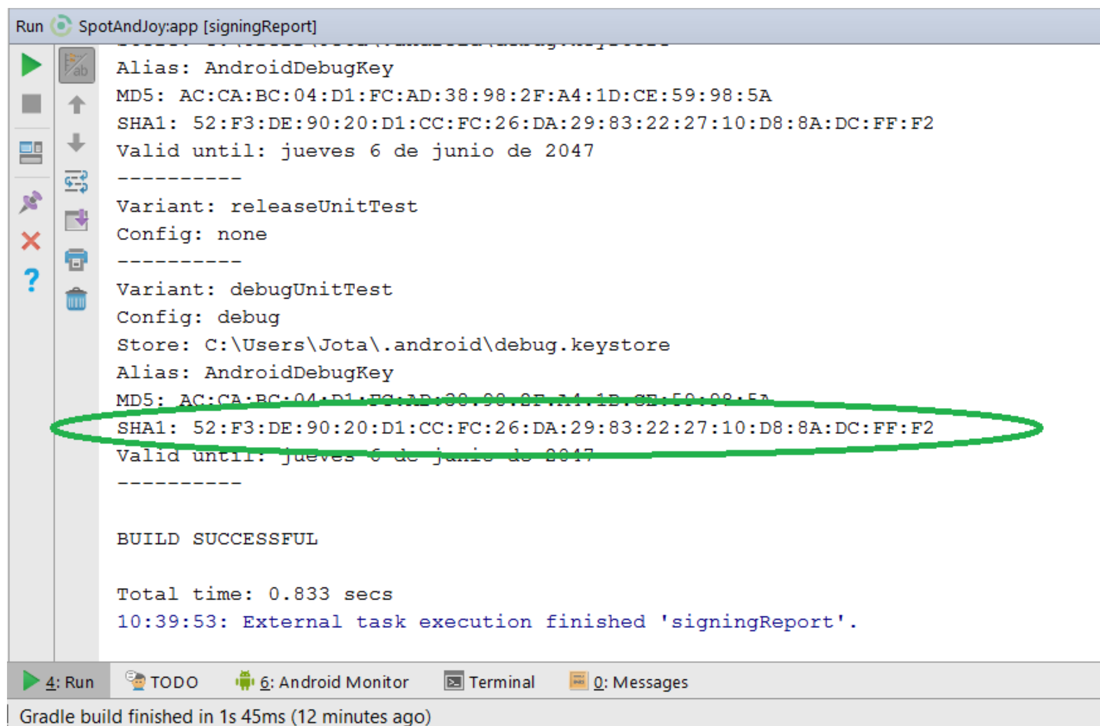

Figura 17. Huella digital del proyecto Android

Se procede a copiar el SHA-1 en el respectivo campo del formulario y se selecciona la opción **Enable**. Si la configuración introducida ha sido válida, el botón habrá cambiado mostrando ahora el mensaje, **Generate to Generate Configuration File**, tal y como se muestra en la figura 18.

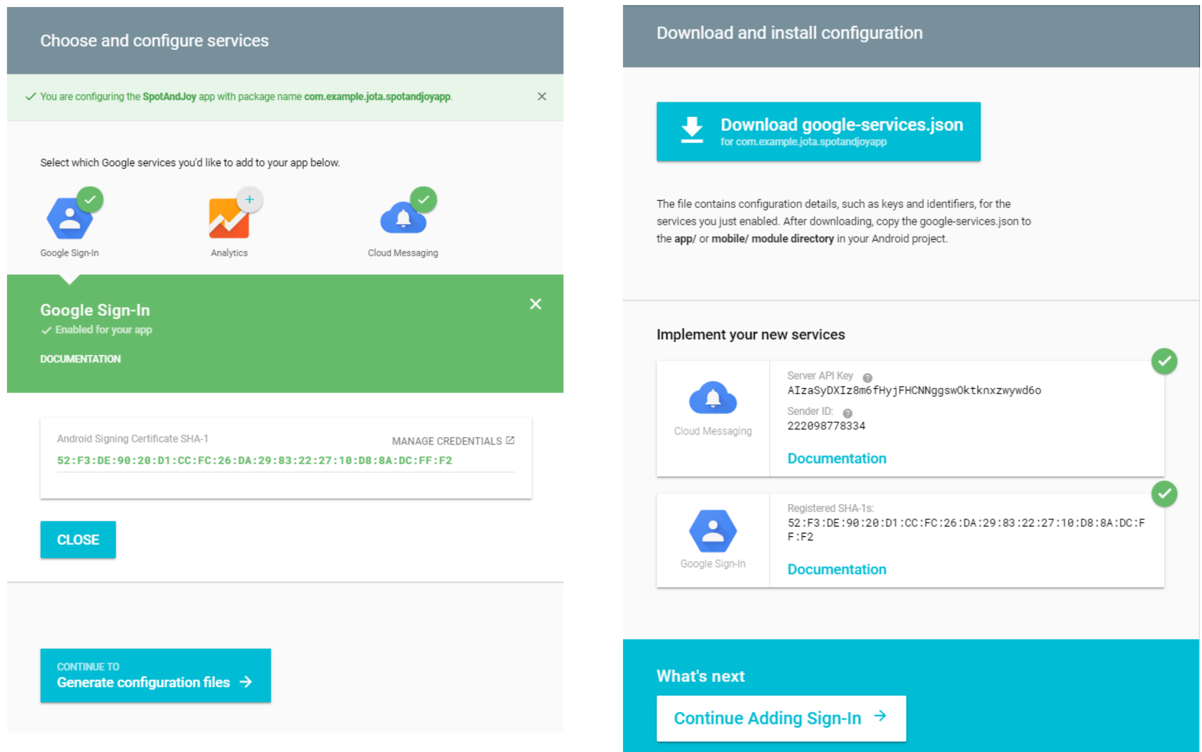

Figura 18. Generar archivo google-services

Se selecciona esta opción y ahora, nos aparece la opción de descargar un archivo JSON llamado *google-services.json*, el cual posee la configuración necesaria, acorde a todos los parámetros que se han ido introduciendo a lo largo del proceso de configuración del mismo, para poder hacer uso en nuestro proyecto de **Google Sign-In**.

5.1.5.1.2. Integración en el proyecto

Para incorporar este archivo en el proyecto, basta con copiar éste archivo dentro del directorio app de nuestro proyecto. Tal y como se muestra en la figura 19.

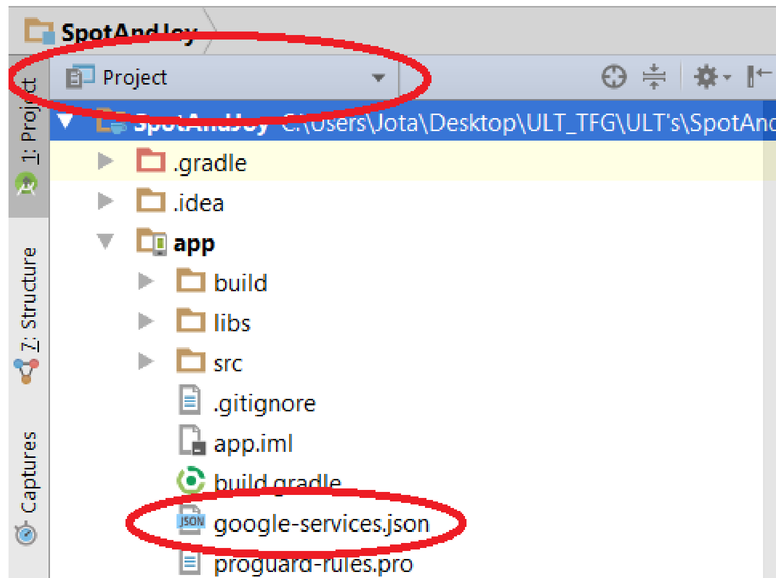

Figura 19. Añadiendo archivo google services

Para terminar la correcta configuración del plugin dentro del proyecto, basta con añadir las siguientes dependencias en el **gradle** del mismo.

- Dentro del módulo app, se debe añadir:

```
apply plugin: 'com.google.gms.google-services'
```

Y la siguiente dependencia:

```
compile 'com.google.android.gms:play-services-auth:10.2.0'
```

- Dentro de Project, se debe añadir la siguiente dependencia:

```
classpath 'com.google.gms:google-services:3.0.0'
```

Con este último paso se da por finalizado el proceso de integración de Google Sign-In en el proyecto. Los servicios ofrecidos por el mismo se pueden ver reflejados en puntos posteriores de esta documentación, donde se explica cómo y dónde ha sido utilizado en nuestro proyecto.

5.1.5.2. Plataforma Spotify

5.1.5.2.1. Spotify Android SDK

El siguiente punto a tratar es el de la incursión de Spotify dentro de nuestra plataforma. Para ello, se accedió a la plataforma web de la propia aplicación para analizar qué puede ofrecer Spotify a desarrolladores de software Android.

El portal es accesible desde la dirección: https://developer.spotify.com/ Donde se puede observar que la plataforma musical ofrece facilidades para desarrolladores tanto de Android como de iOS, y una API Web.

Para nuestro proyecto, se accederá a la sección *Spotify Android SDK*, donde la primera información que nos ofrece respecto al SDK es que permite añadir de manera sencilla funcionalidades de Spotify a un proyecto Android. También nos advierte que, actualmente el estado de dichas funcionalidades se encuentra en fase **beta**, aunque esto no es del todo así, puesto que parte de la misma ya ha entrado en fase **alpha**, tal y como se podrá ver a continuación.

5.1.5.2.2. Las librerías

Las funcionalidades vienen recogidas en dos librerías, las cuales llevan asociada la extensión de archivos para Android AAR:

- Spotify Authentication Library: ofrece la posibilidad de loguearnos en la plataforma Spotify, obtener el *token* de autenticación, el cual permite a nuestra aplicación poder utilizar el resto de servicios de Spotify. Actualmente ha pasado a fase de alfa dentro de la web de desarrolladores, y dicha versión ha sido integrada en este proyecto a raíz de su aparición (**Android Auth 1.0**)
- Spotify Player Library: contiene clases para el *streaming* y reproducción de sonido. Se encarga de la negociación con los servicios backend (**Versión 24-noconnect-2.20b**).

5.1.5.2.2.1. Inconvenientes de las librerías

Aquí es donde entra en juego el mayor inconveniente de Spotify a la hora de su desarrollo en Android, si se desea llevar a cabo la reproducción de canciones completas, es necesario poseer una cuenta ***Premium***. Actualmente, su versión beta permite reproducir una muestra de buena parte de las canciones, por lo tanto, en este punto se desestima la opción de que un usuario ***No Premium*** permita crear una sesión de reproducción, ya que lo que se pretende en este punto, es que todas las canciones puedan ser compartidas y reproducidas en su totalidad.

Otro inconveniente de cara a la librería *Player* es el hecho de estar orientada a no conexión y, por tanto, se dedica a descargar en memoria las canciones que en algún determinado punto forman parte del reproductor.

5.1.5.2.2.2 Configuración de la plataforma

Un punto importante a tener en cuenta, es que estas librerías están orientadas para trabajar con una API mínima de nivel 15 (versión 4.0.3 – Ice Cream Sandwich)

Otro inconveniente del cual nos advierte la plataforma, es que actualmente ninguna de las librerías que ofrecen trabaja directamente con metadatos de la plataforma. ¿Qué supone esto? Pues básicamente que como desarrollador y únicamente con Android SDK integrado en nuestro sistema, no se tiene acceso a la información que ofrece Spotify, y, por consiguiente, no se tiene forma de obtener las canciones.

Para poder tener acceso a dicha información, se puede utilizar su API Web, pero se pospone dicho aspecto y no será retomado hasta la finalización de éste.

Partiendo de la anterior premisa, se procede a crear dentro de la plataforma de Spotify Developers un nuevo proyecto. Para ello, se debe acceder a la siguiente dirección: https://developer.spotify.com/my-applications/ Lógicamente, se debe estar logueado con nuestras credenciales para poder acceder a este apartado de la plataforma. A continuación, se muestra el proceso en la figura 20.

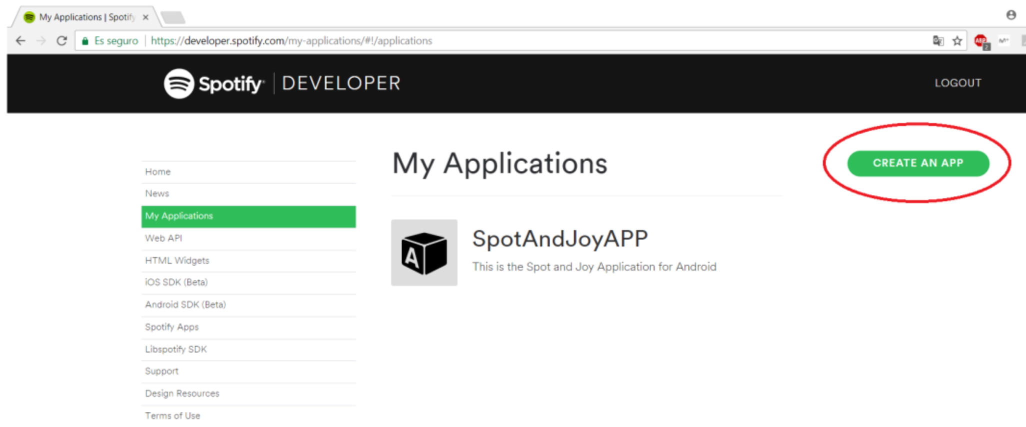

Figura 20. My Applications (Spotify)

Tal y como se aprecia en la imagen anterior, existe un proyecto llamado SpotAndJoyApp, el cual contiene la configuración registrada del presente proyecto. Para crear otro nuevo, basta con pulsar donde dice **Create an App**, y seremos redirigidos a una ventana como la que puede verse en la figura 21.

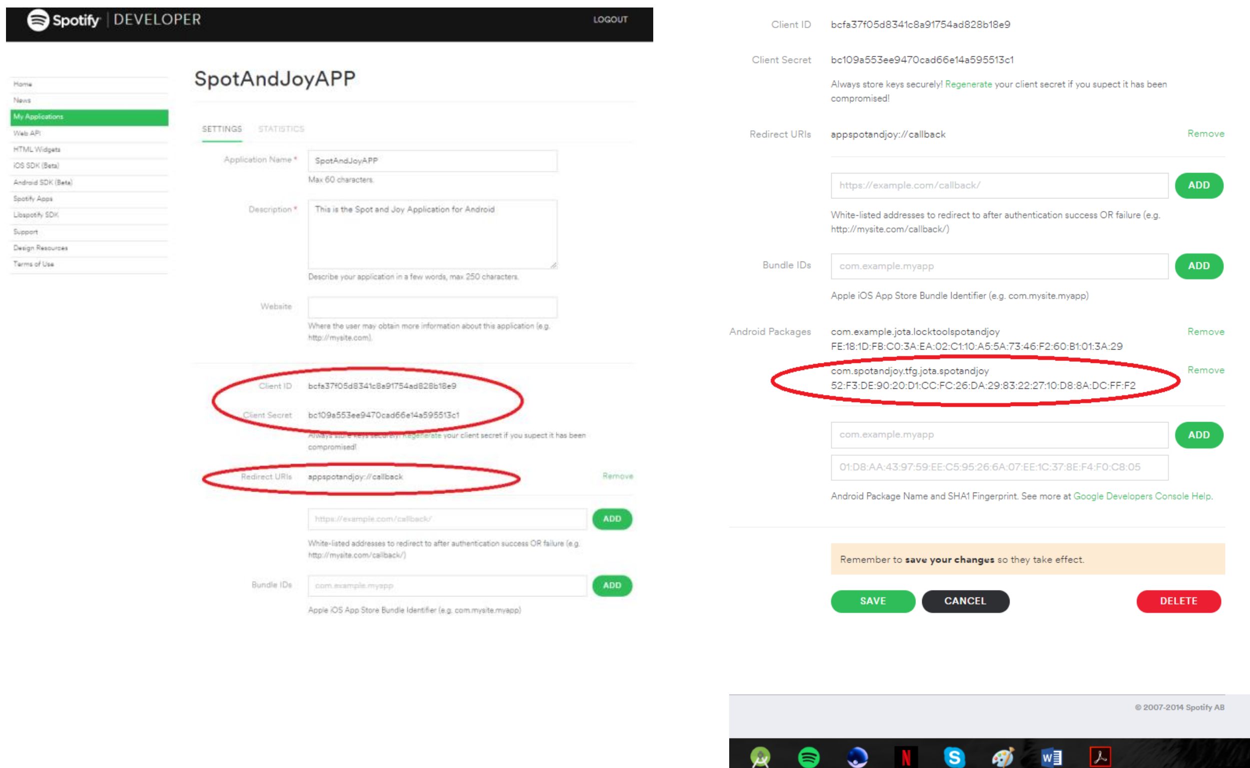

Figura 21. Configurando la aplicación

En la primera parte de las dos anteriores imágenes, se puede que se trata de un formulario de configuración para nuestro proyecto, dentro de Spotify, por lo que se debe indicar con qué nombre se va a registrar dentro de la plataforma y dar una breve descripción de la aplicación.

Se puede apreciar también que la plataforma nos ofrece unos campos en el formulario que vienen completados y no pueden ser cambiados: *Client ID y Client Secret*. Ambos serán necesarios para nuestro proyecto como se verá más adelante. Se debe insertar también una dirección de redirección para las URL.

La segunda pantalla básicamente continúa con el formulario el cuál no se veía adecuadamente en una única captura, y nos pide rellenar un último apartado, donde se pide el nombre con la dirección del paquete Android del proyecto, y el código SHA-1 (ya se vio cómo se puede obtener éste en la sección **(4.4.1.1 – Obtención de archivos necesarios**, referente a Google Sign-In)

Al pulsar aceptar, los cambios se verán reflejados de forma automática. La aplicación ha sido creada por el sistema.

5.1.5.2.2.3. Obtención de archivos necesarios

Ya se tiene la configuración del proyecto dentro de la plataforma, ahora se procede a descargar los archivos de las librerías de Spotify.

- Spotify Authentication Library: Puede obtenerse a través de la siguiente dirección: https://github.com/spotify/android-auth/releases
- Spotify Player Library: Se obtiene accediendo a la siguiente dirección: https://github.com/spotify/android-sdk (concretamente, el archivo *spotify-player-24-noconnect-2.20b.aar*)

5.1.5.2.2.4. Integración en el proyecto

- **Spotify Player**

Para integrar la librería en nuestro proyecto, simplemente se debe copiar el archivo de la misma dentro de la directiva *app/libs* de nuestro proyecto.

A continuación, se accede al módulo "**:app**" de nuestro gradle, y se deben añadir la siguiente información tanto en *repository* como en *dependencies*.

```
compile 'com.spotify.sdk:spotify-player-24-noconnect-2.20b@aar'
```

- **Spotify Authentication**

Inicialmente su integración era igual de sencilla que la librería Player, pero al pasar de versión *beta* a *alpha*, la gestión de la misma efectuó cambios. Aunque la utilización de la misma en este proyecto ha sido tal y como se hacía en su versión anterior, sí que se debe cambiar la integración de la nueva al sistema.

Básicamente se procede a simplificar su integración para poder ser usada por nuestra aplicación. Para ello, se extrae el archivo **.zip** descargado en el punto anterior referente a esta librería y se ejecuta el archivo llamado *gradlew.bat* que no deja de ser un wrapper que permite registrar la librería para poder hacer uso de la misma sin necesidad de integrar ningún archivo como en el caso anterior.

Se debe añadir la siguiente dependencia en el módulo**: app**

```
compile 'com.spotify.android:auth:1.0.0-alpha'
```

En el archivo *Manifest* de nuestro proyecto, se añade el permiso para que la aplicación pueda tener acceso a Internet y acceso a los sistemas de localización:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Llegados a este punto, se posee completamente integrado Spotify SDK a nuestro proyecto. En puntos posteriores se retoma este punto donde se muestra cómo han interactuado las clases y métodos ofrecidos por ambas librerías en este proyecto.

5.1.5.2.3. Wrapper Web API

El siguiente punto retoma el tema mencionado en el apartado anterior referente a los metadatos de Spotify y el principal problema que se tiene sobre los mismos.

Hasta el momento se ha encontrado solución para la autenticación en la plataforma y la reproducción de canciones, pero, ¿cómo podrán ser buscadas? Pues una solución que Spotify ofrece pare solventar este inconveniente es el de hacer uso a su Web API, para poder realizar peticiones GET que nos permita obtener dicha información.

Esto supone al desarrollador tener que implementar distintos modelos para clases donde almacenar la información ofrecida por dicha API.

5.1.5.2.3.1. ¿Por qué el Wrapper?

Una forma de ahorrar tiempo en todo esto y de encontrar un modelo ya definido, es hacer uso del **Wrapper** o "envoltorio" de la API que se ofrece también en su página web, el cual ha sido desarrollado por el equipo Spotify, y ofrece a los desarrolladores que quieran hacer uso del mismo para obtener cualquier tipo de metadatos, el poder obtenerlo de manera sencilla bajo dos principios, incluso define métodos *callbacks* para llevar a cabo la obtención de los mismos.

¿Qué nos permite obtener dicho envoltorio? Pues básicamente cualquier información referente a canciones, álbumes, playlists, autores…

Otro punto muy importante es que para poder hacer uso del **Wrapper**, se debe pasar el *token de autenticación* de Spotify, lo cual nos permite incluso obtener la lista de playlists que el usuario autenticado tenga a su disposición.

También permite la obtención de elementos del modelo (Canciones, álbumes o playlists) en base a una determinada *query* para búsquedas, o, poder obtener un elemento concreto en base a un identificador dado.

Y es por todo éste conjunto de funcionalidades que, unidos a las de Spotify SDK, hacen que se permita como desarrollador abarcar buena parte de la lista de los objetivos propuestos a conseguir para este proyecto.

5.1.5.2.3.2. ¿Dónde obtener el Wrapper?

Para poder obtener el **Wrapper**, se debe acceder a la siguiente dirección: https://github.com/kaaes/spotify-web-api-android donde se puede obtener un ejemplo funcional con el mismo integrado. Este ejemplo ha servido de guía a la hora de definir varias funcionalidades de Android, todo sea dicho, aunque éste punto se verá en la gestión de la vista de Búsquedas. Además de ayudar a comprender cómo funciona el propio **Wrapper**, qué es cada elemento objeto prediseñado, y qué llamadas a la **Web API** se están llevando a cabo.

5.1.5.2.3.3. Integración en el proyecto

Una vez descargado el ejemplo funcional, probado y comprendido en su totalidad, se decide llevar a cabo su integración (del **Wrapper**) en el proyecto. Para ello, se deben tener abiertos en Android Studios tanto el ejemplo seleccionado, como el proyecto donde se está trabajando.

Primeramente, se debe definir un nuevo módulo para nuestro proyecto. Para ello, se hace click en el despliegue de componentes del proyecto (parte izquierda de la pantalla de desarrollador, y se hace click con el botón derecho, teniendo seleccionada la raíz **app**. A continuación, se marca la opción *New > Module*

Android Studio ofrece su catálogo de tipos de módulos disponibles, se selecciona **Android Library** y se selecciona *Next*. Aquí se debe asignar un nombre a la librería (el nombre del módulo cambiará acorde al nombre de librería que se le quiera dar) y se debe asignar una *SDK* mínima (se recomienda dejar la misma que tiene el proyecto. En este caso, nivel API 19)

Al seleccionar *Finish*, el módulo aparece como creado en la vista de navegación de elementos de la izquierda del proyecto.

A continuación, se debe acceder al proyecto ejemplo que se descargó previamente, y se deben copiar **todos** los elementos disponibles en el respectivo módulo, dentro de la carpeta *java*, y ser pegados en la respectiva carpeta del módulo de nuestro proyecto.

Este paso es un tanto delicado, puesto que hemos copiado archivos de un proyecto que no es el nuestro, lógicamente tendrá asignados unos nombres de paquetes que probablemente poco tendrán que ver con los aquí elegidos. Por lo tanto, se procede a hacer uso de la utilidad *Find/Replace* (Ctrl+Shift+R) Es un proceso

donde hay que tener mucho cuidado, pero rápido y efectivo. Se debe introducir qué texto se quiere buscar, que para este caso es la dirección del paquete que desea ser cambiada (menos la extensión del elemento propiamente dicho) y por qué texto se desea modificar cada coincidencia, y es aquí donde se debe introducir nuestra dirección del paquete. Se selecciona la opción *Replace All*.

Ya para terminar, basta con configurar las respectivas dependencias dentro del gradle, donde se puede apreciar que ahora se posee un elemento más, y no es otro que el del módulo que acabamos de crear. Pues es en este donde primeramente se accede, y se debe añadir la siguiente dependencia:

```
compile('com.squareup.retrofit:retrofit:1.9.0')
```

Para terminar, se accede al módulo :app del proyecto y se añade la siguiente dependencia:

```
compile project(':spotifywebapi')
```

Con este último paso se da por finalizada la integración del Wrapper en el proyecto. La utilización del mismo, así como explicación de su modelo, será visto en puntos posteriores, concretamente en la fase de diseño.

Se da por sentado que en el documento *Manifest* se encuentra el permiso para que la aplicación pueda tener acceso a Internet.

5.1.5.3. Plataforma nimBees

Otro objetivo importante a conseguir en este proyecto, es el de poder comunicar varios dispositivos móviles con la finalidad de que se puedan mandar canciones a determinados usuarios, permitiendo así una aplicación *multiusuario*, donde un usuario pueda conectarse a otro que haga de reproductor, y a su vez el que haga de reproductor, administre a los usuarios de la sesión.

5.1.5.3.1. ¿Qué es nimBees?

Aquí es donde entra en juego nimBees, una potente API de notificaciones PUSH que permite enviar notificaciones personalizadas entre clientes mediante Internet o Bluetooth. Es una forma más que puede permitir conseguir los objetivos previamente propuestos. La plataforma se encuentra disponible tanto para sistemas Android, como para iOS.

Características Principales

- Segmentación de usuarios: enviar notificaciones PUSH sólo a los usuarios adecuados y enseñándoles la información relevante acorde al contexto.
- Filtros avanzados: envía notificaciones basadas en cómo, dónde o cuándo los usuarios utilizan sus dispositivos.
- Privacidad: la información de los usuarios no tiene porqué ser almacenada en servidores de terceros.
- Gestión de beacons.

Otro aspecto positivo es que la plataforma, aunque ofrezca servicios de pago, también garantiza el envío de un determinado número de mensajes (10000 al mes) de forma gratuita, éste es otro motivo que, como desarrolladores, podamos optar a esta API, o al menos probarla.

Se pretende definir un conjunto de mensajes haciendo uso de las notificaciones para que sean enviados como notificaciones y gestionados de forma adecuada por los distintos dispositivos móviles que hagan uso de este proyecto.

5.1.5.3.2. ¿Cómo obtener nimBees?

Una vez explicado de forma breve qué puede ofrecer nimBees para el desarrollo de este proyecto, se accede a su portal web, donde se procede a llevar a cabo un registro gratuito para la plataforma.

5.1.5.3.2.1. Integración en el proyecto

Para obtener el archivo de librería de la plataforma, basta con acceder a su portal web, registrarse e ir a la sección de descargas:

https://api.nimbees.com/downloads

La versión utilizada para la realización de este proyecto ha sido la 1.5.1. Una vez descargada, y al igual que se hizo durante la integración del *Wrapper* de la Spotify Web API (*véase 5.1.5.2.3, Wrapper Web API)*, se define un nuevo módulo en Android, y se selecciona la opción que dice *Import .JAR/AAR Package*. Ahora únicamente se añade la dependencia de la nueva plataforma en el módulo: app

```
compile project(':nimBees-platform')
```

Llegados a este punto, se recuerda que durante la integración de la plataforma Sign In ofrecida por Google, a la hora de definir el proyecto, se habilitó el servicio *Google Cloud Messaging*. Pues esto se debe a que el mismo será necesario a la hora de incorporar nimBees en la aplicación.

5.1.3.3.2.2. Configuración de Google Cloud

Se debe acceder a la plataforma online Google Cloud para habilitar (si no lo están una serie de servicios) Para ello se debe acceder con la cuenta de correo Gmail con la que se realizó previamente la configuración de Google Sign In (aunque este paso no es del todo necesario, se puede definir un nuevo proyecto totalmente distinto en cualquier otra cuenta y habilitar allí los servicios)

https://console.cloud.google.com

Se debe seleccionar el proyecto en cuestión (en caso de utilizar el mismo) y seleccionamos la sección *APIs & services*, tal y como se puede ver en la figura 22.

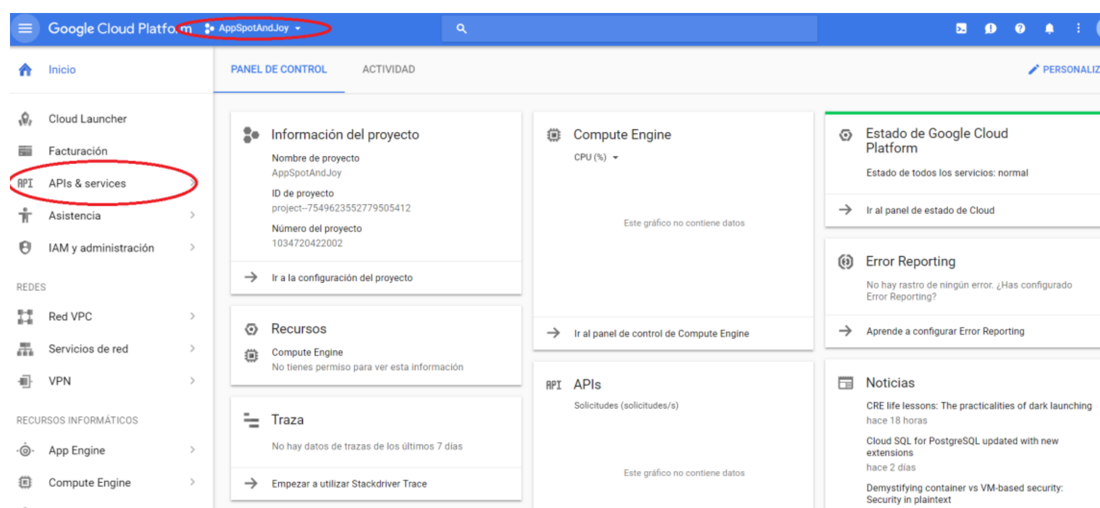

Figura 22. Google Cloud (Inicio)

En la siguiente vista aparecen los servicios habilitados en este proyecto. Para configurar correctamente la plataforma, se deben habilitar las siguientes APIs y servicios:

- BigQuery API
- Google Cloud Messaging
- Google Cloud SQL
- Google Cloud Storage
- Google Cloud Storage JSON API

Si alguno de los anteriormente listados no apareciese, basta con seleccionar la sección ***Habilitar APIS Y SERVICIOS***, e ir habilitando los mismos.

A continuación, se pueden ver dichos servicios habilitados en la figura 23.

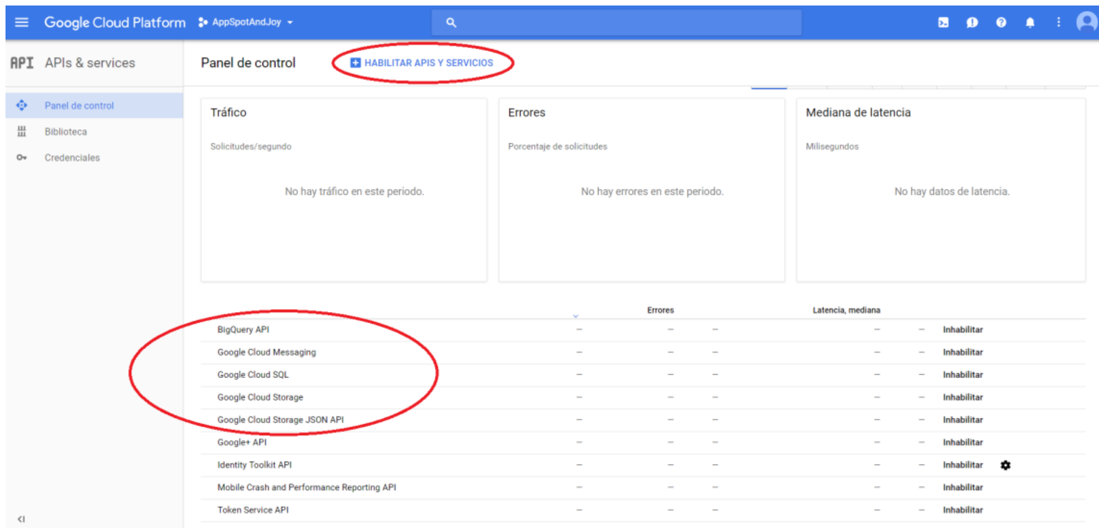

Figura 23. Google Cloud (APIs disponibles)

Al habilitar Google Cloud Messaging si no se hizo durante la integración del Sign In, la opción no aparecerá y al habilitarla, generará automáticamente una clave de API asociada a dicho servicio en este proyecto. Algo similar a lo que puede verse en la figura 24.

Clave de API creada

Para usar esta clave en tu aplicación, transfiérela como un parámetro `key=API_KEY`

Tu clave de API

AIzaSyAmjC-ypcMxAEV-9YVt8pQe1yxCFQC-lKc

Restringe la clave para impedir el uso no autorizado en producción.

CERRAR RESTRINGIR CLAVE

Figura 24. Google Cloud (Creando una nueva clave de API)

Incluso se puede restringir con la huella digital SHA-1 de nuestro proyecto Android y su nombre de paquete para limitar su uso en nuestro proyecto, tal y como se puede ver en la figura 25.

Clave de API

AIzaSyAmjC-ypcMxAEV-9YVt8pQe1yxCFQC-lKc

Nombre

Clave de API AppSpotAndJoy

Restricción de clave

Si restringes una clave, puedes especificar qué sitios web, direcciones IP o aplicaciones pueden usarla. Más información

- Ninguna
- URLs de referencia HTTP (sitios web)
- Direcciones IP (servidores web, tareas cron, etc.)
- Aplicaciones para Android
- Aplicaciones para iOS

Restringir el uso a tus aplicaciones Android (Opcional)

Añade el nombre del paquete y la huella digital del certificado de firma SHA-1 para restringir el uso de tus aplicaciones de Android

Puedes encontrar el nombre del paquete en el archivo AndroidManifest.xml. A continuación, usa el comando siguiente para obtener la huella digital:

```
$ keytool -list -v -keystore mystore.keystore
```

Nombre de paquete	Huella digital de certificado SHA-1
com.spotandjoy.tfg.jota.appspotandj	52:F3:DE:90:20:D1:CC:FC:26:DA:29:83:22:27:10:D8:8A:DC:FF:F2

+ Añadir nombre de paquete y huella digital

Nota: Pueden pasar hasta 5 minutos antes de que se aplique la configuración

Guardar Cancelar

Figura 25. Google Cloud (Restricción del proyecto vía SHA-1)

Dicha clave será necesaria, al igual que el número de serie de este proyecto en Google Cloud para realizar el siguiente paso. Para poder obtenerlo, basta con seleccionar la sección de Inicio y en el apartado *Información del proyecto* aparece, entre otros datos, el número de serie del mismo. En la figura 26, se puede apreciar la información del presente proyecto.

Información del proyecto

Nombre de proyecto

AppSpotAndJoy

ID de proyecto

project--7549623552779505412

Número del proyecto

1034720422002

Ir a la configuración del proyecto

Figura 26. Google Cloud (Propiedades del proyecto)

5.1.3.3.2.3. Configuración del portal nimBees

Se accede de nuevo al portal web de la plataforma, y se selecciona la sección *Applications*: https://api.nimbees.com/apps

Se procede a definir nuestro proyecto registrado en la plataforma, para ello se debe crear una nueva *APP* dotándola de un nombre.

En la pantalla aparece de forma automática la nueva aplicación. La seleccionamos y vamos a una pantalla estadística con información sobre el uso de la aplicación como: dispositivos y notificaciones registradas.

Se selecciona la pestaña que dice *App Configuration*, donde nos aparece un formulario con campos generados automáticamente por el portal (*App Key* y *App Secret*) y nos pide para configurar la misma en sistemas operativos Android insertar una clave de API y el número de serie del proyecto (en el apartado anterior se explicó cómo obtenerlos) La vista deberá ser similar a la mostrada en la figura 27. Se selecciona *SAVE APPLICATION INFORMATION* para guardar los cambios.

Figura 27. Configurando nuevo proyecto en nimBees

5.1.3.3.2.4. Configuración de nimBees en el proyecto

Como último paso a realizar se procede a configurar nuestro proyecto, para poder vincularlo al portal web.

Para ello, antes de nada se deben añadir las siguientes dependencias de los servicios de Google en el módulo **:app**

```
compile 'com.google.android.gms:play-services-gcm:11.0.1'
compile 'com.google.android.gms:play-services-location:11.0.1'
compile 'com.google.android.gms:play-services-maps:11.0.1'
```

A continuación, se define la carpeta *Assets* mediante *New* > *Folder* > *Assets* en la raíz main del proyecto. Dentro de ella se crea un fichero de recursos (Resource File) donde se debe añadir tanto la APP Key como la APP Secret generada en el proyecto creado en el paso anterior, como el número de serie de proyecto que se registró en el portal.

Esta es la información ***mínima*** que se debe registrar en dicho documento. Aunque se pueden definir otros parámetros como un *NotificationManager* por defecto (se verá en apartados posteriores). En la figura 28, puede verse el fichero en cuestión del presente proyecto.

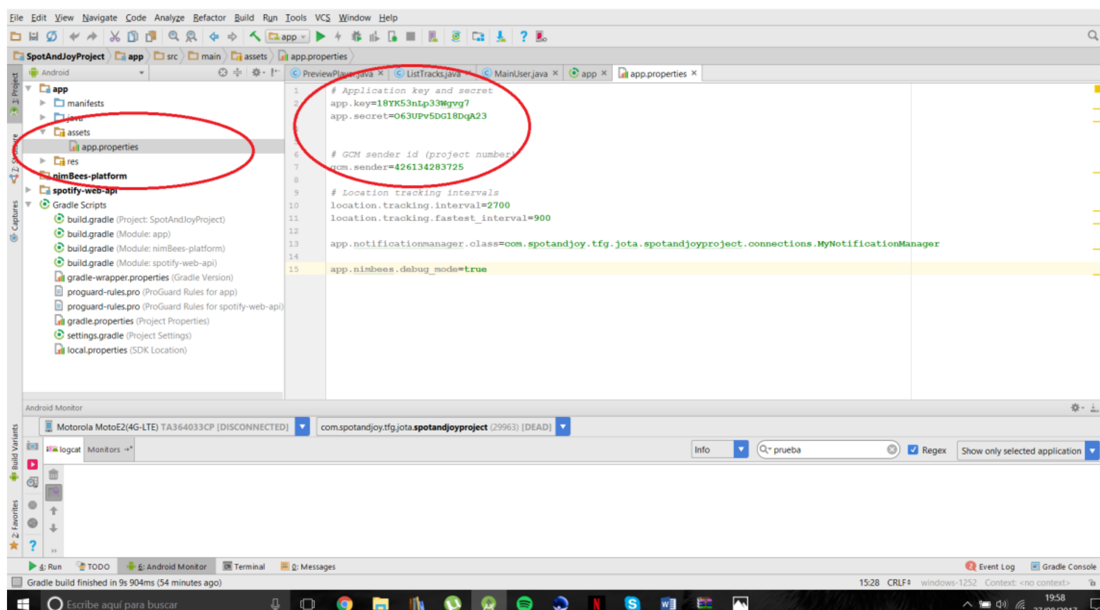

Figura 28. Propiedades de nimBees en la aplicación

Se da por sentado que en el documento *Manifest* se encuentra el permiso para que la aplicación pueda tener acceso a Internet, así como los permisos de acceso a sistemas de localización:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

5.1.4. Otras librerías

Otras librerías de menor escala que han sido utilizadas para la realización de este proyecto.

5.1.4.1. Picasso

Mencionar que la integración de la misma en el sistema se debe exclusivamente a intereses estéticos, para conseguir de forma rápida y eficiente la integración de imágenes de forma dinámica en nuestro código sin complicaciones.

En lo referente a la integración de la misma en el sistema, mencionar que, aunque basta con descargar un archivo desde su portal web, no ha sido necesaria su adición dentro de las librerías del proyecto, ni añadir ninguna dependencia. Para más detalles, leer el último párrafo del apartado que viene a continuación.

5.1.4.2. Guava

Conjunto de librerías que añaden nuevos tipos de colecciones. Su uso en este proyecto permite unir elementos de determinadas cadenas, delimitándolas con un carácter concreto. Por ejemplo, mostrar una lista de autores de un álbum, separando sus nombres por el carácter ‘,’.

Basta con añadir la siguiente dependencia dentro del módulo **:app** del proyecto.

```
compile 'com.google.guava:guava:20.0'
```

5.1.5. Conclusiones y complicaciones durante la fase de estudio

En este punto de análisis comienzan a aparecer las primeras limitaciones para la aplicación. Se trata de la aparición de Internet como elemento necesario para poder llevar a cabo la ejecución de la aplicación. Todas las plataformas previamente descritas a excepción de *Picasso* necesitan conexión a Internet para efectuar sus funcionalidades.

También mencionar que la dificultad para poder integrar las librerías previamente descritas fue mayor a lo esperado, pues la plataforma ***nimBees*** ya traía incorporadas dependencias a otras como *Picasso*, redundancias que fueron eliminadas de forma manual de la sección *External Libraries* del proyecto.

5.2. Diseño

En el presente apartado se describe todo el proceso de diseño llevado a cabo para el desarrollo de esta aplicación. Dicho diseño garantiza el cumplimiento de los objetivos y especificaciones a conseguir, proporcionando una idea completa del software desarrollado en este proyecto. Se debe definir qué arquitectura *software* va a garantizar que se puedan cumplir todos los objetivos y requisitos, cómo se traducen los componentes identificados en la fase de análisis y diseñar la interfaz de la aplicación.

5.2.1. Arquitectura

La arquitectura utilizada para la resolución de este proyecto se apoya en un modelo Cliente/Servidor. En el caso del proyecto, y como puede verse en la siguiente figura, una aplicación móvil constituye la parte cliente y servidor, ejerciendo uno de los dos roles (administrador, servidor o usuario, cliente). A su vez todos los dispositivos móviles serán clientes de servicios proporcionados a través de las plataformas nimBees (envío de mensajes) y Spotify (obtención de elementos de la plataforma como canciones, álbumes o playlists)

La comunicación entre los distintos dispositivos móviles de una sesión será mediante un sistema de Petición-Respuesta para la mayoría de los casos.

A continuación, se muestra en la figura 29 el diseño de la arquitectura en rasgos generales.

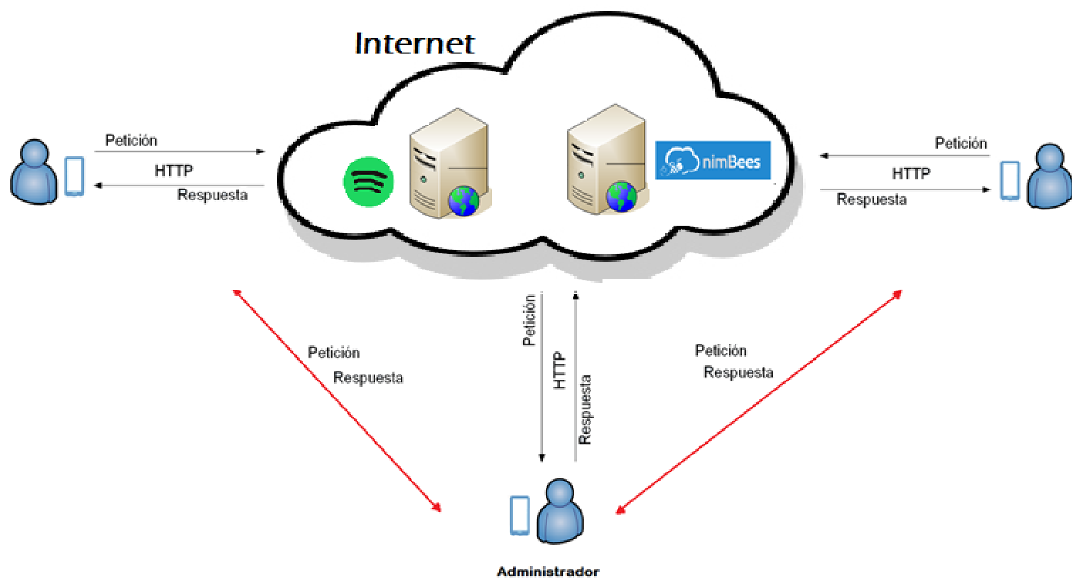

Figura 29. Arquitectura general de la aplicación

El cliente es quien inicia las solicitudes a las plataformas, teniendo un papel activo en la comunicación, y esperando respuesta del servidor. Por otro lado, en lo referente a las funcionalidades multiusuario, el móvil cuyo rol desempañado sea el de *Administrador* será el encargado de ofrecer al resto de usuarios la información en tiempo real a la ejecución, al igual que el resto de dispositivos serán a éste tipo de usuarios a los que puedan enviar canciones, solicitar entrar o salir de su sala…

Por lo tanto, a partir de este punto, se comienza a hablar de dos roles claramente distintos, tanto a nivel de ejecución de algunas funcionalidades como a nivel de implementación y arquitectónico (este punto será abordado con los servicios)

A continuación, se procede a mostrar los diseños arquitectónicos definitivos utilizados para la presente aplicación. Los diseños varían en función del **rol** desempeñado (si es un usuario administrador o no), pues no todos los componentes (vistas, servicios…) interactúan de la misma forma, sino que hay una gestión del rol que permite en todo momento limitar el uso apropiado durante la ejecución de la aplicación.

Aunque a rasgos generales la secuencia sea muy similar, no es así las funcionalidades internas necesarias para llevar a cabo la ejecución de las tareas.

5.2.1.1. Administrador

Usuario con la capacidad de buscar y reproducir música. Posee una serie de usuarios bajo su control, pudiendo recibir canciones de estos. A continuación, se muestra el diagrama definitivo de su arquitectura para esta aplicación, mostrando a su vez su flujo de navegabilidad entre los componentes.

A continuación, se muestra en la figura 30 la arquitectura de la aplicación para un dispositivo móvil bajo el rol de administrador.

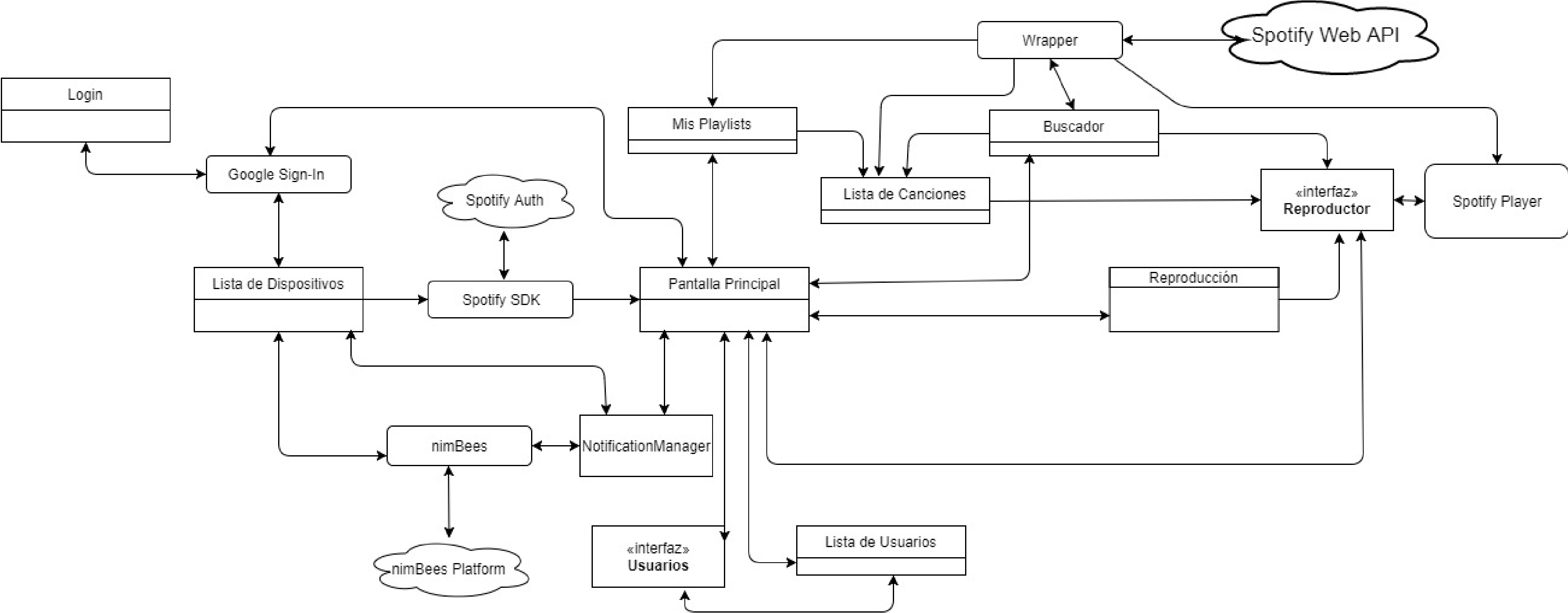

Figura 30. Arquitectura software (Administrador)

Se procede a describir cada uno de los componentes que interactúa en el diagrama anteriormente descrito, qué abarca en la arquitectura y qué es lo que permite dentro de ella.

Activities

La arquitectura presenta tres actividades en total.

- Login (Sign-In): permite acceso dentro de la aplicación
- Lista de Dispositivos (Devices): permite buscar dispositivos, crear una nueva reproducción o volver a Login.
- Pantalla Principal (MainUser): se supone que se ha creado por este dispositivo una nueva reproducción. En esta actividad se ofrece la posibilidad de acceder a un menú con diferentes funcionalidades: cargar playlists, buscar canciones, administrar usuarios conectados a la sesión, ver qué está siendo reproducido actualmente y qué sonará después, qué usuarios hay en la sala, incluso poder expulsar alguno de ellos y cerrar la sesión, en dicho caso, se volverá a la lista de dispositivos.

Spotify SDK (Spotify Auth)

Gestiona las credenciales de Spotify y verifica que, siendo éstas correctas, dicho usuario posea una cuenta ***Premium*** dentro de la plataforma para poder crear la reproducción. Dichas credenciales (el *token* de acceso) son almacenadas dentro de la aplicación cuando se cumplen todos los requerimientos explicados.

Wrapper (Spotify Web API)

Conjunto de llamadas que permiten obtener información necesaria de la Spotify Web API siendo los resultados cargados en un modelo definido dentro del mismo.

Fragments

Diferentes vistas con diferentes funcionalidades ofrecidos por el menú de la pantalla principal.

- Cargar Playlists: Haciendo uso de las credenciales y mediante el *Wrapper*, devuelve una lista con las *playlists* que dicho usuario posee almacenadas dentro de su cuenta de Spotify.
- Buscador: Permite buscar dentro de la Spotify Web API una determinada canción, álbum o grupo (solamente una categoría por búsqueda) Si el criterio seleccionado a buscar ha sido el de “Canción”, si se selecciona algún elemento resultado de la búsqueda, éste es directamente de la misma dentro del reproductor mediante su interfaz de acceso. En caso de estar bajo algún otro criterio, se cargará un fragment donde, haciendo uso del *Wrapper* permita obtener la lista de canciones del elemento seleccionado.
- Lista de canciones: Muestra una lista de canciones con la posibilidad de marcar y desmarcar a las mismas con el fin de que solamente las marcadas serán añadidas al reproductor a través de la interfaz cuando estén seleccionadas.
- Reproducción: Carga del reproductor la canción que se encuentre actualmente en reproducción y aquellas próximas (máximo 10) pendientes a reproducir, siempre y cuando estés reproduciendo y haya canciones pendientes.
- Usuarios: **exclusiva del administrador**, permite al usuario que está administrando la sesión poder gestionar aquellos usuarios que están conectados a esta reproducción (si los hay), teniendo la opción de eliminar a cualquiera de ellos en cualquier momento (no queremos usuarios molestos) Los usuarios son obtenidos de un servicio encargado de gestionar a los mismos.

nimBees y notificationManager

Gestiona las peticiones de acceso, petición de manifiesto cuando se buscan dispositivos, salidas o envío de canciones por parte de aquellos usuarios no administradores.

Servicios

- Reproductor (PlayerService): **exclusivo del administrador**, se encarga de gestionar la reproducción dentro del dispositivo, añadir nuevas canciones y proporcionar información del estado del mismo (si suena o no, qué canción suena, cuáles se reproducirán a continuación…)
- Usuarios (UsersService): **exclusivo del administrador**, se encarga de gestionar altas y bajas de aquellos usuarios conectados dentro de la sesión, así como comprobar la existencia de alguno, o proporcionar cuáles son los mismos.

5.2.1.2. Usuarios comunes o estándar

Usuario común de la aplicación, es decir, aquel que está conectado a algún reproductor y puede enviar canciones al mismo. A continuación, se muestra el diagrama definitivo de su arquitectura (figura 31) para esta aplicación, mostrando a su vez su flujo de navegabilidad entre los componentes.

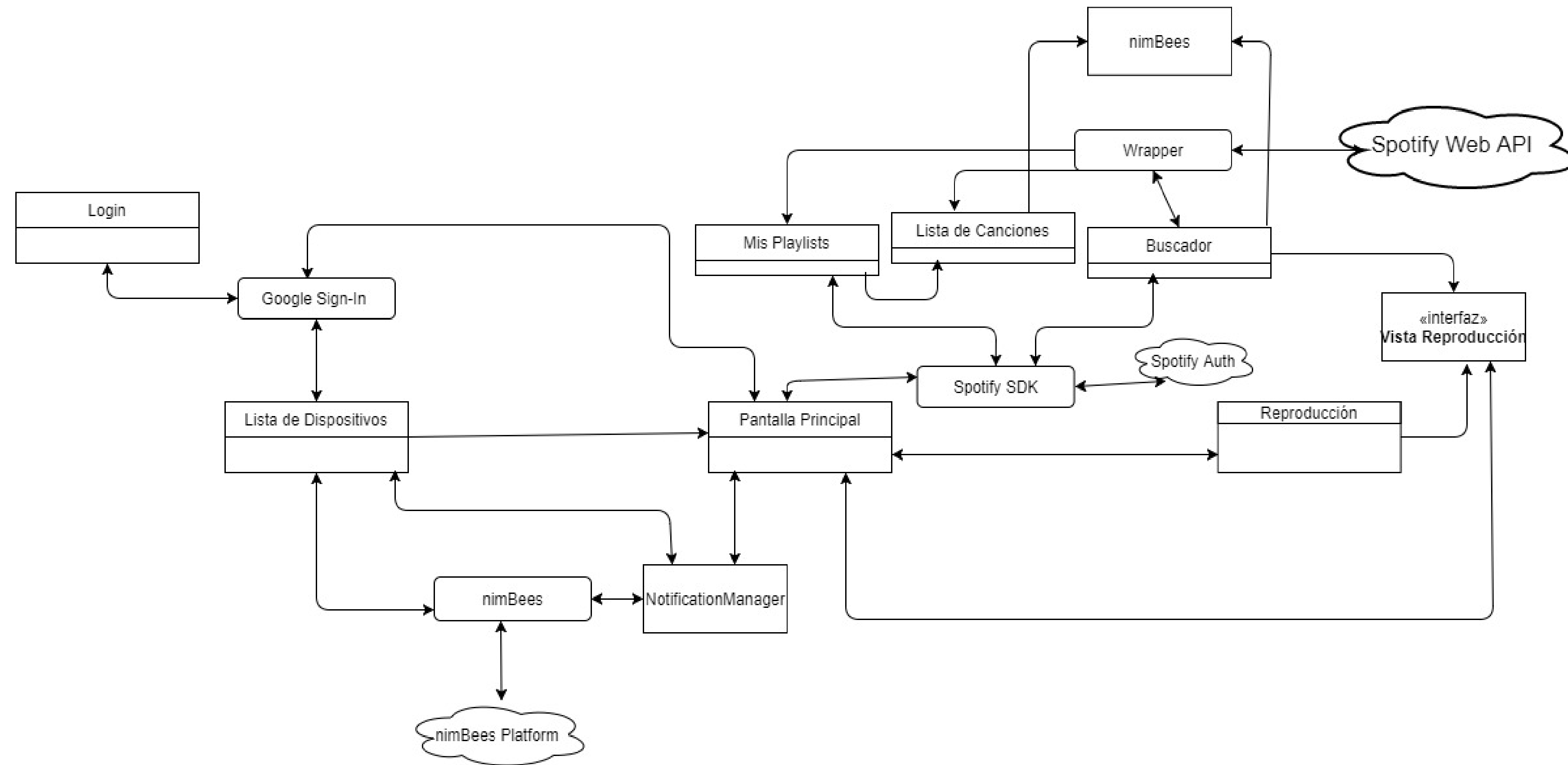

Figura 31. Arquitectura software (Usuario estándar)

El aspecto del diagrama es bastante similar al anterior, teniendo en cuenta que ambos están implementados bajo los cimientos de una arquitectura común, solo que algunas funcionalidades están limitadas por su rol. La navegabilidad de ciertos componentes recibe un leve cambio, o directamente, ciertos servicios no son ejecutados para dar pie a otros.

En este apartado sólo se reescriben aquellos puntos cuya funcionalidad ha podido variar o aquellos nuevos.

Las *activities* y *fragments* son exactamente los mismos. Solo que **desaparece** el *fragment* de **usuarios**, ya que es **exclusivo** para el **administrador** de la sesión.

Las canciones seleccionadas a reproducir serán enviadas mediante **nimBees** al dispositivo administrador de la sesión.

Spotify SDK (Spotify Auth)

Esta funcionalidad es ejecutada únicamente si no se poseen credenciales de Spotify almacenadas en dicho dispositivo por la aplicación y se intenta acceder a las *playlists* o al buscador desde la pantalla principal.

nimBees y notificationManager

Gestiona las respuestas a mensajes gestionados por el administrador, tales como que se manifieste para ser mostrado en la lista de dispositivos, solicitar acceso al mismo y qué información ofrece éste cuando se conecte dicho dispositivo a la sesión. Además, se encarga de enviar canciones al dispositivo administrador, así como obtener del mismo la información actualizada de la reproducción.

Servicio Vista de Reproducción *(PlayerView Service)*

Almacena y gestiona la información obtenida gracias a **nimBees** por parte del administrador para mantener la lista de reproducción mostrada en el *fragment* Reproductor actualizada.

5.2.2 Vistas de la aplicación

Partiendo del esqueleto arquitectónico descrito en el apartado anterior, el cual garantiza el cumplimiento de los objetivos marcados a conseguir por la presente aplicación. Ahora, partiendo de la idea inicial planteada por el prototipo básico, se pretende a mostrar las diferentes vistas desarrolladas para la aplicación. Garantizando una interfaz de usuario que siga satisfaciendo los objetivos propuestos y a su vez, respetando algunas partes del estilo decorativo presentes en Spotify.

En la figura 32 se puede observar las vistas definidas para las tres *activities* desarrollados: la pantalla de bienvenida, la de búsqueda de dispositivos y la pantalla principal de una sesión.

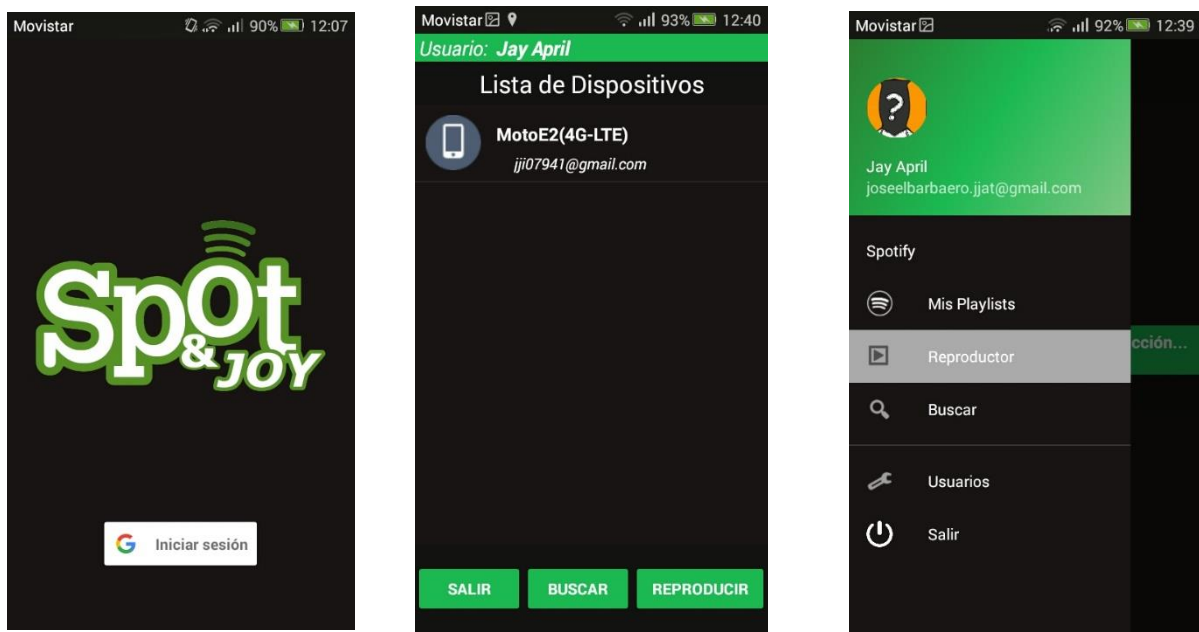

Figura 32. Vistas activities

A continuación, en la figura 33, se muestran las vistas definidas para los *fragments* (nombrados de izquierda a derecha): *reproducción, mis playlists y buscador*.

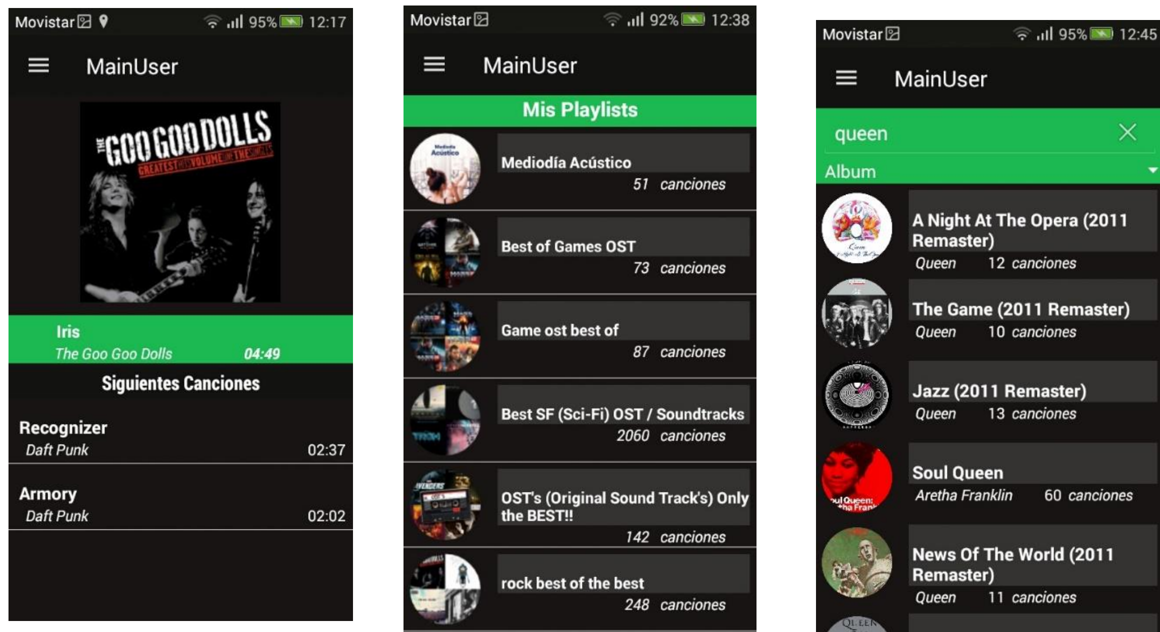

Figura 33. Vistas de fragments (1)

Para finalizar, se muestran en la figura 34 las vistas de los fragments: *lista de canciones* y *usuarios.*

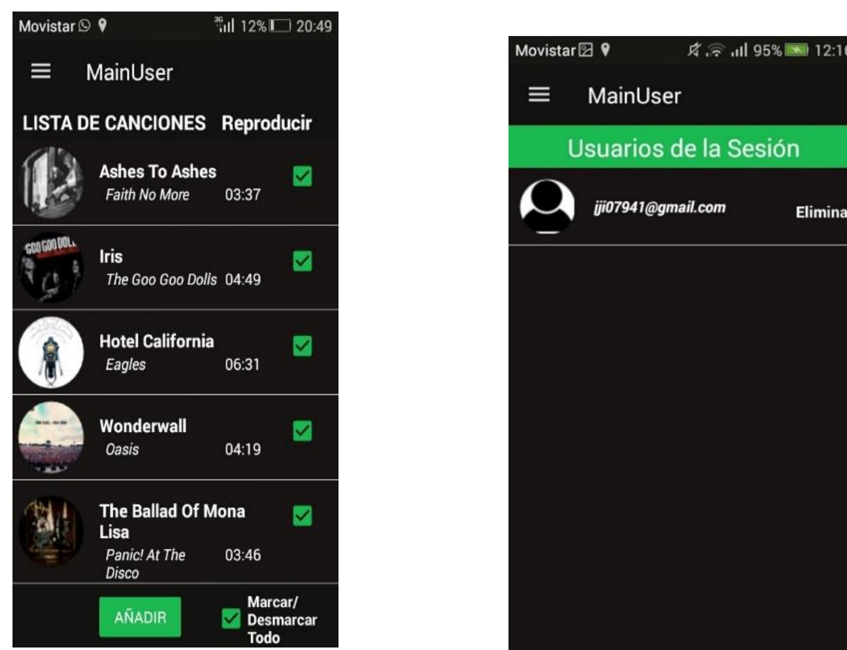

Figura 34. Vistas de fragments (2)

5.2.3. Colores y estilos

Se ha intentado preservar la estética de Spotify [2] en lo referente a la gama de colores utilizada por dicha aplicación que, aunque simple en cantidad, estéticamente da una presencia propia y ha permitido que las vistas previamente descritas, tengan ese toque respecto a la contrapartida original (tabla 1).

Color	Cod.	Utilidad
Blanco	#FFFFFF	El color del texto
Negro	#000000	Algunos fondos para ítems de listas
Negro (Spotify)	#191414	Color de fondo principal utilizado en las vistas.
Verde (Spotify)	#1ED760	Utilizado como fondo para botones y elementos a resaltar.

Tabla 1. Colores de la Aplicación y Uso

- ***Styles***

Otro punto a destacar es el del estilo de las imágenes. Convertir las imágenes de los distintos elementos de Spotify tratados (canciones, álbumes o playlists) y transformar dichas imágenes volviéndolas circulares. Para ello se ha definido una clase dentro del código llamada *Styles* encargada de dicha transformación.

- ***MyTime***

La clase *MyTime* se encarga de transformar el tiempo expresando en milisegundos (unidad trabajada con los metadatos disponibles por Spotify) en una cadena formato *mm:ss*

- **Animaciones**

Se define el directorio *anim* donde se ha incorporado un archivo de recursos encargado de animar en el *fragment* Reproductor el título de la canción que se encuentre sonando en dicho instante (si no la hay, no se limpia la animación)

Dicha animación es un elemento *translate* de 15 segundos de duración en modo bucle tal que el texto aparecerá en la parte derecha de la pantalla e irá desacelerando acorde se mueva horizontalmente en dirección a la parte opuesta de la misma.

- **Navigation View**

Se personaliza un *Navigation View* disponible en la Pantalla Principal con los datos disponibles de la cuenta Gmail con la que el usuario procedió a realizar el logueo en la aplicación. Cargando la imagen que tiene almacenada como foto de perfil en dicho correo, al igual que su nombre y el propio correo.

El *Navigation View* posee a su vez un menú que ofrece las funcionalidades disponibles a llevar a cabo desde la Pantalla Principal.

En lo referente al diseño de los mismos, decir que se ha añadido la opción:

```
android:checkableBehavior="single"
```

Dicha opción, unida a un selector definido dentro del directorio *drawable* permiten que, al seleccionar un determinado ítem dentro del menú desplegado en la Pantalla Principal, éste pase a estar seleccionado, indicando al usuario dónde se encuentra en todo momento dentro de la Pantalla Principal (archivo *nav_state_background.xml*)

- **Imágenes por defecto**

Cuando un usuario no tiene una imagen de perfil en su correo Gmail, en su pantalla principal se cargará una imagen por defecto cuyo archivo se encuentra en el directorio *drawable* del presente proyecto. A continuación, se puede ver la imagen del icono por defecto en la figura 35.

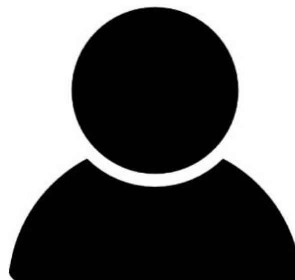

Figura 35. Icono de usuario por defecto

Este mismo criterio es aplicado en previas a cargar una canción, por defecto se muestra una imagen del directorio *drawable*. Al igual que cuando no hay ninguna canción en curso, cuya imagen se corresponde con la figura 36.

Figura 36. Icono de canción por defecto

Los dispositivos encontrados a la hora de realizar una búsqueda de administradores activos, poseen una imagen representativa disponible también en el directorio en cuestión. La imagen en cuestión es la mostrada en la figura 37.

Figura 37. Icono de dispositivo móvil

- **Otras imágenes**

El logo de la aplicación se encuentra basado en uno de los primeros diseños del logo de la aplicación Spotify, cuya imagen se corresponde con la figura 38.

Figura 38. Logo de la aplicación [2]

- **Icono de la aplicación**

Haciendo uso de la aplicación **Iconion** para su definición y de la funcionalidad ofrecida por **Android Studio** para su ajuste de tamaño para la aplicación, *Configure Image Assets*, se define el icono para la presente aplicación, el cual se encuentra inspirado por el color verde y la forma de *Spotify*. Su imagen se corresponde con la figura 39.

Figura 39. Icono de la aplicación

5.2.4. Listas personalizadas

Para cada una de las listas *(ListView* o *RecyclerView*) se ha procedido a diseñar un *layout* a nivel de celda con su respectivo adaptador para la vista (bajo algún determinado modelo, que serán descritos en la parte de *Implementación* de esta documentación)

Los **adaptadores** básicamente se encargan de comunicar un determinado objeto de modelo con un determinado *layout*, pudiendo convertir dichos objetos en distintas filas para las listas en base a la vista definida por la vista seleccionada.

A continuación, se muestra un ejemplo de este tipo de vistas donde hay un atributo *ImageView* con el icono del elemento (por ejemplo, la imagen correspondiente a una determinada canción) su nombre, su autor y su duración, y cuyos elementos pueden verse en la figura 40.

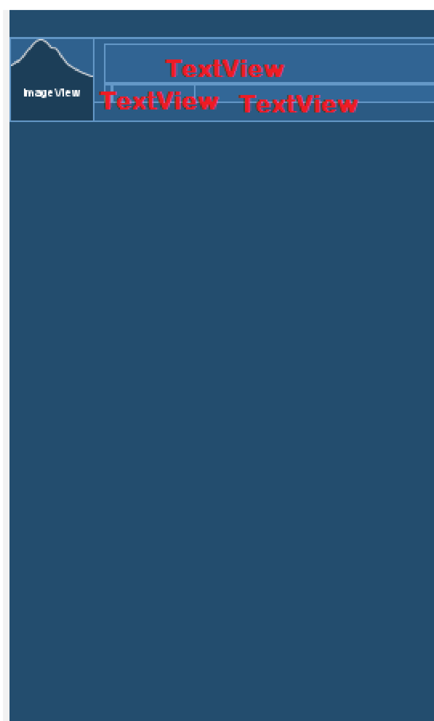

Figura 40. Elemento de una lista personalizado (layout.xml)

En el código correspondiente a este proyecto, y dentro del directorio *layouts* hay un archivo correspondiente a cada ítem de cada lista desarrollada en esta aplicación, con sus respectivas variaciones en cada vista.

Se retoma este punto durante el apartado de implementación, reflejando el comportamiento de los *AdapterViews* y sus respectivos modelos.

5.3. Implementación

Llegados a este punto, se ha realizado un estudio del problema, y se ha definido una posible solución para poder abordarlo. Se ha diseñado el esqueleto de la solución, su arquitectura, y se ha diseñado un envoltorio "a medida" para esa arquitectura, la interfaz. A continuación, se define el proceso de implementación del sistema, el proceso de desarrollar y extender la arquitectura, así como comunicarla con las vistas que componen la interfaz.

En el presente punto, se explicarán uno a uno todos los componentes arquitectónicos implementados durante el proceso de realización del proyecto, así como qué permiten hacer cada uno de ellos, o qué clases han sido necesarias definir para que las distintas funcionalidades puedan llevarse a cabo.

5.3.1. Activities

Tal y como se describe en apartados posteriores, hay un total de tres implementadas en el sistema: *Sign-In* o pantalla de Login*, Devices* o pantalla de dispositivos *y MainUser* o pantalla principal. Cada activity representa a una pantalla de la aplicación.

5.3.1.1. Sign-In

Esta *activity* permite, pulsando un botón, acceder a los servicios proporcionados por Google Sign In para acceder a una determinada cuenta Gmail, apoyándose en una variable del tipo *GoogleApiClient* que sirve como punto de acceso a la API ofrecida por Google.

Cuando el resultado ha sido satisfactorio, se cambia de activity (Devices) pasando por un *bundle* información sobre dicho usuario (*GoogleSignInAccount*, devuelta por la API cuando la operación de *logueo* ha sido satisfactoria*)*

5.3.1.2. Devices

Pantalla de control de dispositivos. Se trata de una *activity* que permite, una vez logueados en la aplicación, registrar el *email* del usuario del dispositivo en la plataforma **nimBees**. A partir de ese momento, se dispone de tres botones que permiten ejecutar tres funcionalidades: *signOut* o salir de la aplicación, buscar dispositivos cercanos que se encuentren en reproducción o proceder a crear una nueva.

5.3.1.2.1. SignOut

Se vinculan el resto de variables a sus respectivos elementos del *layout* y se configura acceso a la API Google de nuevo, con el fin de poder realizar la acción *signOut* al pulsar el botón **Salir**, en dicho caso, se vuelve a la actividad anterior.

5.3.1.2.2. Buscar Dispositivos

Otra posibilidad disponible es la de realizar una búsqueda de dispositivos, para ello, se declara una variable List con objetos de modelo tipo *UserDevice* (*véase* 5.3.3.2. UserDevice*)* para cargar mediante el respectivo *adapter* (*véase* 5.3.6. Adapters) en la *ListView*. mostrando los resultados.

El resto de detalles referente a la búsqueda de dispositivos es abordado en el apartado *Comunicaciones*.

5.3.1.2.2.1. Solicitud de acceso a dispositivo

Una vez obtenidos dispositivos resultados de la búsqueda, se puede solicitar acceso a los mismos. Dicha funcionalidad se lleva a cabo gracias a la definición de un *itemClickListener* que permite obtener el objeto de modelo de la posición seleccionada. Este punto es retomado en el apartado *5.5.7-Comunicaciones*.

Hacer un ligero hincapié en la llamada efectuada cuando se recibe respuesta por parte del administrador, donde, sin entrar en más detalles, explicar que se cambia de *Activity* a MainUser (Pantalla Principal) indicando que este dispositivo no es **administrador** (boolean pasado por bundle a **false**) También se entrega la dirección de correo del usuario administrador cuyo mensaje acabamos de recibir, registrado así en la pantalla principal a quien debemos enviar las solicitudes y, para terminar, una lista de elementos tipo *PlayerTrack* que contiene una lista de canciones pasadas por el administrador y que servirán a este dispositivo para saber qué está siendo reproducido y qué será reproducido en las inmediaciones (máximo 10 elementos)

Estos puntos serán retomados tanto en *Clases implementadas* como en *Comunicaciones.*

5.3.1.2.4 Crear una sesión de reproducción

Al pulsar el botón de reproducir, el sistema comprueba si existen credenciales registradas (*token* de acceso de Spotify disponible en el archivo *CredentialsHandler)*

A continuación, se muestra en la figura 41 el conjunto de variables más destacables de esta pantalla, siendo todas referente a la autenticación de Spotify.

```
//Spotify SDK Client ID
final String CLIENT_ID="bcfa37f05d8341c8a91754ad828b18e9";

//Spotify SDK Redirect URIs
final String REDIRECT_URI="appspotandjoy://callback";

//Spotify Code for Activity Result
final int REQUEST_CODE=1000;

//Spotify player, sera necesario para comprobar si este dispositivo podra realizar una reproduccion
//(siendo Premium en Spotify) En dicho caso, pasara a ser configurado
SpotifyPlayer mSpotifyPlayer;
```

Figura 41. Variables Activity Devices

Client ID* y *REDIRECT_URI, son proporcionadas por la aplicación definida dentro de Spotify (*véase, apartado 5.1.5.2.2.2. Configuración de la plataforma*)

Partiendo de base de una primera reproducción (no hay token registrado) se lanza una *activity* desde una ventana ofrecida por *Spotify Authentication* que permite introducir nuestras credenciales de usuario o registrarnos en Spotify.

Cuando las credenciales son correctas, éste devuelve un *token resultante* para este usuario, el cual es introducido en nuestro gestor de credenciales.

Ahora se debe verificar que el usuario en cuestión posee cuenta ***Premium.*** Para ello se define una variable *SpotifyPlayer* (proporcionada por la librería que contiene el mismo nombre) y que permite, añadiendo el *implement* ***ConnectionStateCallback*,** comprobar si el *token* acreditado previamente ha permitido o no la conexión a la librería *Player*. En caso de que no sea así, basta comprobar si el error vendrá dado *onLoggedFail*, y se procede a mostrar en pantalla que no se posee una cuenta Premium y, por tanto, no se puede producir una reproducción.

En el caso contrario, si todo ha ido bien, el resultado vendrá dado por el método *onLoggedIn* donde además de iniciar y configurar el servicio de reproducción (*véase 5.3.4.3. Player Service* para más información) se cambia de actividad a *MainUser* especificando que el dispositivo es administrador (se le pasa vía *bundle* un boolean con valor **true)**, además del objeto *GoogleSignInAccount* tal y como se ha hecho para pasar a esta *activity*)

5.3.1.3. MainUser

Antes de nada, mencionar que se ha utilizado un formato de Activity proporcionado por Android Studio llamado *NavigationDrawer Activity* como base, siendo ésta la pantalla principal de la aplicación, pues define gran parte de las funcionalidades a conseguir. También es donde se realiza la diferenciación del

usuario por rol (administrador o no) haciendo uso de un boolean, el cual viene especificado a través del *bundle* de la activity anterior.

El funcionamiento de esta actividad reside en proporcionar un menú con distintas funcionalidades para el usuario y, tratándose de un contenedor, donde se cargará un *fragment* u otro en función de la opción del menú seleccionada. Por defecto, se cargará el *fragment* Player en su interior. También posee el control principal de los eventos de las comunicaciones (véase *5.3.7. Comunicaciones*)

Se hace uso de la implementación *LocationListener* en la propia actividad que permita geolocalizar al dispositivo y guardar su respectiva localización.

En caso de no ser administrador, además se proporciona otro tipo de información, como es la dirección del administrador a la que se encuentra conectado actualmente el dispositivo o qué canciones tiene en curso en su vista de *Reproducción*.

5.3.1.3.1. Orden de ejecución

¿Qué ocurre internamente cuando se ejecuta la *activity MainUser*? Son muchas las variables que entran en juego en este punto, he aquí el motivo de la realización de una sección especial para especificar lo más claramente posible qué se está ejecutando en esta *activity*.

- Registro de servicios: se registran todos los servicios necesarios (PlayerService, UserService y PlayerViewService) Sin indagar mucho sobre ellos, solamente decir que, haciendo uso de la clase *ServiceConnection*, se vinculan e inicializan todos los servicios (aunque cada rol haga uso de los suyos)
- Se ejecuta un *BroadcastReceiver* el cual pasa a ser explicado en el apartado *5.5.8-BroadcastReceiver*)
- Se gestionan los servicios nimBees para localización tal y como se hizo en la activity *Devices*.
- Se obtienen valores a través del *bundle*

- Se actualiza el *FloatingActionButton* siendo mostrado en la activity en el caso de no poseer credenciales de Spotify. Es decir, si somos un usuario no ***Premium*** y no se han insertado credenciales de Spotify (no hay token)
- Se vinculan el resto de elementos de la vista con sus respectivos valores. Como puede ser que *NavigationView* cargue sus valores o se modifique el menú de opciones lateral acorde al rol (véase figura 42)

```
//configuracion y linkado de atributos
private void initHeader() {

    navigationView= (NavigationView) findViewById(R.id.nav_view);

    //si no ees un administrador, ocultamos la funcionalidad users
    if(!isAdmin){
        Menu menu=navigationView.getMenu();
        menu.findItem(R.id.nav_users).setVisible(false);
    }

    //se chequea por defecto la funcionalidad Reproductor
    navigationView.getMenu().findItem(R.id.nav_player).setChecked(true);

    navigationView.setNavigationItemSelectedListener(this);


    View header = navigationView.getHeaderView(0);

    nickGmail=(TextView)header.findViewById(R.id.nick_user_gmail);
    emailGmail=(TextView)header.findViewById(R.id.email_user_gmail);
    photoHeader=(ImageView)header.findViewById(R.id.logo_user_gmail);



    nickGmail.setText(acct.getDisplayName());
    emailGmail.setText(acct.getEmail());
```

Figura 42. Navigation View (Implementación)

- Se inicializa servicio GoogleSignInAPI para habilitar la posibilidad de realizar la opción *SignOut*
- Se carga inicialmente el fragment *Player* en el contenedor definido en la activity (véase figura 43)

```
private void initBody(){
    //iniciar un fragment dentro de la activity
    android.support.v4.app.FragmentTransaction transaction=getSupportFragmentManager().beginTransaction();
    Player player=new Player();
    transaction.replace(R.id.mainContent,player);
    transaction.commit();
}
```

Figura 43. Carga de fragment Player

- En el método *onNavigationItemSelected* proporcionado por defecto al definir este tipo de aplicación se gestiona el identificador seleccionado, en función de los disponibles del menú del *NavigationView* que permiten el reemplazo del *fragment* actual del contenedor por el correspondiente en función del elemento seleccionado. Este aspecto está atado a restricciones más allá de ocultar la opción de usuarios a un dispositivo **no administrador**. Además, permite comprobar que las funcionalidades "*buscar"* y "*mis playlists"* posean el token de autenticación para *Spotify*. En caso de no poseerlo, se debe ejecutar el método *openLoginWindow* tal y como se hacía en la activity *Devices*, encargado de llamar a una actividad de Spotify que solicita credenciales al usuario dentro de su plataforma. Si no se posee una cuenta, igualmente puede ser creada desde esta ventana. En caso de no poseerlas, no se reemplazará el *fragment* en cuestión y si se vuelve a intentar acceder volverá a solicitar credenciales.
- Por otro lado, *LocationListener* se encargará de actualizar la última localización registrada del dispositivo.

5.3.2. Fragments

Contenidos en el interior de la activity *MainUser*, los *fragments* permiten llevar a cabo unas determinadas funcionalidades específicas y mostrar sus respectivas interfaces en el contenedor de la *activity*. *MainUser* Todos ellos son accesibles desde la propia actividad a través de las opciones disponibles en el *NavigationDrawer* a excepción de *Lista de Canciones*.

5.3.2.1. Mis Playlists

Dadas las credenciales de un usuario en Spotify (se posee almacenado un *token* de acceso), permite realizar la búsqueda haciendo uso de los servicios ofrecidos por el *wrapper* de la Web API enviando una petición a la misma. Dicha acción será realizada en una tarea ejecutada en segundo plano *(background)* utilizando para ello un elemento del tipo AsyncTask.

Dicho AsyncTask es ejecutado dentro una vez la vista ha sido creada (*onViewCreated)* y recibiendo como parámetros un elemento *ProgressDialog* para mostrar durante la ejecución del método *doInBackground()…* , el contexto del fragment, y una lista de elementos tipo Playlist ofrecida por el *Wrapper* donde se cargarán los resultados obtenidos (si los hay)

Durante la ejecución, el método *onPreExecute()* del AsyncTask mostrará un mensaje para mostrar en el ProgressDialog el cuál pasará a ser mostrado durante la ejecución de todo el método *doInBackground* y será ocultado una vez haya finalizado la tarea a ejecutar en segundo plano dentro del método *onPostExecute()* donde también se procederá a mostrar los resultados en una *ListView* vinculada al *fragment* (si los hay) en caso contrario, la lista es ocultada y se muestra un lugar un mensaje informando de no haber encontrado resultados dentro de un *TextView*.

¿Cómo se realiza la obtención de los elementos *Playlist*s? Pues gracias al *wrapper* basta con una simple llamada para obtener los 20 primeros resultados obtenidos dentro de este criterio. Es necesario cargar el *token* de acceso de nuestro archivo gestor de credenciales (véase, *5.3.5. Cestor de Credenciales*) para poder hacer uso de dicha funcionalidad. A continuación, se puede ver el proceso en la figura 44.

```
@Override
protected Void doInBackground(Void... params) {
    SpotifyApi api=new SpotifyApi();
    api.setAccessToken(CredentialsHandler.getToken(getActivity()));
    Map<String, Object> options = new HashMap<>();
    int cont=0;
    options.put(SpotifyService.OFFSET,cont);
    while (api.getService().getMyPlaylists(options).items.size()>0){
        List<PlaylistSimple> auxList=api.getService().getMyPlaylists(options).items;
        for(int i=0;i<auxList.size();i++){
            listResults.add(auxList.get(i));
        }
        cont=cont+20;
        options.put(SpotifyService.OFFSET,cont);
    }
    return null;
}
```

Figura 44. Obtención de Playlists

Para obtener todos los resultados, se hace uso de una variable del tipo *hashmap* reconocida por el *wrapper* que permite la inserción de opciones definidas en el mismo, con un determinado valor.

Se ha utilizado una variable contadora inicializada a 0, tal que cada vez que se ejecute dentro de un bucle la obtención de listas con éxito, ésta aumentará 20, y se

introducirá de nuevo dentro del *hashmap* como la nueva opción de desplazamiento a realizar. Si al comprobar dentro del bucle que, con ese desplazamiento el resultado es mayor que 0, continúa cargando playlists y actualizando el contador de desplazamiento.

A su vez, cada resultado mostrado en pantalla podrá ser seleccionable (*onItemClickListener*) cargando así el objeto de modelo vinculado a la posición deseada y, en tal caso, reemplazar el *fragment* por Lista de Canciones, al cual se le deberá cargar el valor “2” (*véase 5.3.2.3. Buscador*) como criterio a cargar, el identificador de la lista (**id**) y el propietario de la misma (**owner**)

5.3.2.2. Reproductor

Como se menciona anteriormente, es el *fragment* cargado por defecto. Y su objetivo no es otro que el de mostrar el estado actual de la reproducción. Básicamente, si en el servicio de Reproducción (Player) está sonando algo, esa canción debe aparecer en la respectiva vista. Y lo mismo ocurre con las primeras diez más inmediatas pendientes de reproducir (en caso de haberlas)

En este punto es necesario hablar muy brevemente de los servicios de nuevo debido a una cuestión muy simple: si es el servicio de reproducción el encargado de mostrar la información actualizada en tiempo real, ¿qué ocurre con aquellos usuarios no administradores que no interactúan con este servicio? (Se recuerda que el servicio *Player* es únicamente utilizado por usuarios administradores) Pues es en este punto en el que se decide implementar un servicio ajeno encargado de almacenar la información que el administrador ofrezca a aquellos usuarios que se encuentren conectados a él. Dicho punto será retomado en puntos posteriores, tanto por los propios servicios *Player* y *PlayerView*, como con las comunicaciones y el broadcastreceiver.

5.3.2.3. Buscador

La aplicación proporciona un sistema de búsquedas en base a una determinada cadena a buscar haciendo uso del *wrapper* de Spotify y de un determinado criterio de filtrado donde, haciendo uso de un *Spinner* se puede seleccionar entre distintos elementos a buscar: Canción, Álbum o Playlist. Antes de continuar, mencionar que se necesita obtener de parte de la activity *MainUser* la información de si dicho usuario es admin o no y, en caso de no serlo, el correo del mismo. También se vinculará el servicio Player para poder interactuar con él (cuando sea necesario)

Otro punto a destacar es que se trabaja con un elemento tipo *RecyclerView* en lugar de una *ListView* y esto se debe precisamente a conseguir optimizar los resultados e ir cargándolos acorde se vayan necesitando. No es necesario cargar unos posibles 70 mil resultados que pueda dar perfectamente Spotify de una tirada.

El *fragment* posee un elemento del tipo *SearchView* que permite introducir la cadena y realizar la búsqueda considerando la misma y el criterio seleccionado. Cuando el texto es introducido, se hace uso *onQueryTextListener* para proceder a realizar la búsqueda accediendo al método de la interfaz ***ActionListener*** (*Search.java*)

El cual ofrece:

- Insertar un *token* de autenticación pasado por parámetro, el cuál como se aclara en puntos anteriores, siempre es necesario para acceder al *wrapper*.
- Realizar una búsqueda por una cadena y un criterio dados por parámetros.
- Cargar más resultados, ya la vista posee un *ScrollListener* que permite ir cargando más contenido gestionando la vista con un *LayoutManager.*

Para cargar los resultados, al igual que en el resto de vistas, se ha definido un tipo de *adapter* por cada tipo de resultado posible: *SearchResultsAdapter* para canciones, y *SearchResultsPlaylists Adapter y SearchResultsAlbum Adapter* para los respectivos restantes. Cada *adapter* trabaja con un elemento de modelo disponible gracias al *wrapper* (*Track* para canciones y *Playlist* y *Album* para sus respectivos de nuevo) Para ver cómo se gestiona cada *adapter* ver el apartado *Tal-Adapter*.

La interfaz ***ActionListener** es implementada en **SearchPresenter***. Donde se destaca el uso de una clase llamada *SearchPager* que permite la gestión tanto de la página a mostrar, como a cargar más resultados cuando el *ScrollListener* reconozca un desplazamiento. Para ello, accede al *Wrapper* reajustando continuamente la opción de *Offset* en cada búsqueda y realizando la misma en función del valor que se seleccionase en el *spinner.* A continuación, se muestra en la figura 45, cómo se gestiona la búsqueda, en función del tipo seleccionado en el *spinner*

```
switch (tipo){
    case "Canción":
        mSpotifyApi.searchTracks(query, options, new SpotifyCallback<TracksPager>() {
            @Override
            public void success(TracksPager tracksPager, Response response) {
                listener.onComplete(tracksPager.tracks.items,null,null);//si ha ido bien,
            }

            @Override
            public void failure(SpotifyError error) {
                listener.onError(error);//si ha habido un error, cargamos el mismo
            }
        });
        break;

    case "Album":

        mSpotifyApi.searchAlbums(query, options, new SpotifyCallback<AlbumsPager>() {
            @Override
            public void failure(SpotifyError error) {
                listener.onError(error);//si ha habido un error, cargamos el mismo
            }

            @Override
            public void success(AlbumsPager albumsPager, Response response) {
                listener.onComplete(null,albumsPager.albums.items,null);
            }
        });
```

Figura 45. Fragment Buscador (SearchPager)

Otro punto importante a destacar aquí es, que para evitar tener que cargar continuamente en *background* cada vez que se desee cargar nuevos resultados, se ejecutan los métodos haciendo uso de los *callbacks* ofrecidos por el *wrapper* tal y como se puede apreciar en la imagen anterior.

SearchPresenter se encarga de añadir resultados de los tres tipos, para mostrar en la vista cuando estos son encontrados. Solo que, en base al valor obtenido por el spinner, se sabe dónde reside el verdadero resultado para ser mostrado, tal y como puede verse en la figura 46.

```
@Override
public void addData(List<Track> items, List<AlbumSimple> itemAlbums, List<PlaylistSimple> itemPlaylists) {
    searchKind=spinner.getSelectedItem().toString();
    switch (searchKind) {
        case "Canción":
            mAdapter.addData(items);
            break;
        case "Album":
            mAlbumAdapter.addData(itemAlbums);
            break;
        case "Playlist":
            mPlaylistAdapter.addData(itemPlaylists);
            break;
    }
}
```

Figura 46. Fragment Buscador (Obtener Datos)

Para terminar falta especificar qué puede hacer un usuario en base a los resultados obtenidos. Pues aquí la respuesta viene dada una vez más en función de qué estuvo buscando el usuario y, para el caso de que la respuesta a la cuestión anterior sea canciones, dependerá de si el mismo es un administrador o no.

Si se trata de una canción y siendo dicho usuario administrador, ejecutará el servicio *Player* para enviar la canción al reproductor. En caso de no serlo, enviará un mensaje al usuario que si lo sea con la canción a reproducir (véase *5.3.7. Comunicaciones*)

Cuando la canción sea gestionada de una forma u otra, se reemplazará el fragment actual por el *Reproductor*.

En caso de tratarse de algún otro criterio, el sistema trabaja igual para ambos roles. Simplemente marcará con un número entero qué criterio ha sido el utilizado, siendo “1” para álbumes y “2” para *playlists*.

Se reemplazará el fragment actual por el de *Lista de Canciones*, donde además del número anteriormente descrito se le pasará el identificador del elemento y, en caso de tratarse de una *playlist* ***(id)***, el identificador del usuario dueño de la misma (**owner**)

5.3.2.4. Lista de Canciones

El acceso a éste *fragment* es llevado a cabo desde *Mis Playlists* o *Buscador* donde, básicamente se cargan los resultados en función del elemento seleccionado en los *fragments* anteriores.

Al igual que el *fragment* anterior, éste también se encuentra vinculado al servicio *Player* para ejecutar sus métodos cuando sea oportuno.

Dicho *fragment* cuenta con dos funcionalidades asíncronas ejecutadas en dos *AsyncTasks* distintos: uno para obtener resultados y otro, para enviar las canciones resultantes seleccionadas a reproducir.

En este *fragment* interfiere un nuevo objeto de modelo, con su respectivo *adapter* (*TrackSelect* y *TrackSelectAdapter*, respectivamente, véanse tal- modelo y tal- adapters) En esencia, dicho elemento se trata de un elemento *Track* perteneciente al definido en el *Wrapper* junto a un **boolean** que indicará si dicha *track* se encuentra seleccionada o no.

Para mostrar los resultados se hace uso de un *AsyncTask* llamado *LoadTasks,* el cual recibirá como parámetros un *ProgressDialog* para mostrar durante la ejecución del segundo plano (*background*), el tipo de criterio a buscar ("1" para álbum o "2" para playlists) y una lista donde cargar los resultados formada por elementos del modelo *TrackSelect*. Este proceso puede verse ilustrado en la figura 47.

```
//Para searchkey 1
private void searchTracksOnAlbum(){
    SpotifyApi spotifyApi=new SpotifyApi();
    spotifyApi.setAccessToken(CredentialsHandler.getToken(context));
    List<TrackSimple> trackSimples=spotifyApi.getService().getAlbum(idToLoad).tracks.items;
    for(int i=0;i<trackSimples.size();i++){
        String idTrack=trackSimples.get(i).id;
        Track track=spotifyApi.getService().getTrack(idTrack);
        listResults.add(track);
    }
}

//Para searchkey 2
private void searchTracksOnPlayList(){
    SpotifyApi spotifyApi=new SpotifyApi();
    spotifyApi.setAccessToken(CredentialsHandler.getToken(context));
    List<PlaylistTrack> listPlaylistTrack=spotifyApi.getService().
            getPlaylist(idOwner,idToLoad).tracks.items;
    for(int i=0;i<listPlaylistTrack.size();i++){
        Track track=listPlaylistTrack.get(i).track;
        listResults.add(track);
    }
}
```

```
@Override
protected void onPreExecute() {
    super.onPreExecute();
    dialog.setMessage("Cargando canciones...");
    dialog.show();
}

@Override
protected Void doInBackground(Void... params) {
    switch (searchKey){
        case 1:
            searchTracksOnAlbum();
            break;
        case 2:
            searchTracksOnPlayList();
            break;
    }
    loadingModel();
    return null;
}
```

47. Carga de resultados doInBackground(...)

Por defecto, todas las canciones serán cargadas con el valor de marcado **true** dentro del respectivo modelo, siendo mostrada ésta en la pantalla con el *checkbox* de la vista correspondiente marcado.

Ahora y llegados a este punto, se tiene a disposición tres funcionalidades para con los resultados.

- **Marcar/Desmarcar todos los resultados**: como su propio nombre indica, recorre la lista de objetos accediendo a su vista y poniendo el valor de checkeo a **true** o a **false** en función de si la opción de la vista se encuentra marcada o no.
- **Marcar o desmarcar manualmente**: todas y cada una de las canciones de la vista pueden ser marcadas y desmarcadas de forma manual, alterando el valor del modelo cargado en la posición seleccionada dentro de la *ListView* de resultados.
- **Enviar canciones**: al igual que ocurría en el buscador, en este apartado se debe hacer hincapié en la distinción del rol del usuario, pues varía completamente la funcionalidad. En cualquiera de ambos casos, se ejecutará un *AsyncTask* encargado de procesar la tarea (*LoadTask)* El mismo también recibe por parámetro el contexto de la aplicación y el acceso a un *ProgressDialog* para mostrar durante la ejecución en segundo plano (*background*) en la cual se efectuará el envío de las canciones ya sea al reproductor si el usuario en cuestión es administrador o, proceder a enviar las canciones a aquel usuario que sea el administrador de la sesión a la que se encuentra el usuario conectado. La ejecución en segundo plano consiste en recorrer la lista de canciones e ir comprobando el valor del **boolean** que indica si se encuentra marcada o no. En caso de estarlo, siendo administrador de la sesión se manda al servicio a reproducir. En caso contrario, el funcionamiento es otro muy distinto.

  Sin haber descrito las comunicaciones en los puntos anteriores, se procede a hablar de manera breve sobre ellas, en lo referente al envío de canciones de parte de usuarios de una sesión a su administrador.

  Se deben enviar las canciones al reproductor, pero, ¿cuántas se deben mandar en cada mensaje?, o, por otro lado, ¿se mandan de una sentada?

Para agilizar los tamaños de los envíos, y sin entrar mucho en detalle, decir que las canciones seleccionadas se irán mandando de 10 en 10 hasta terminar de recorrer la lista en cuestión y ayudándose de un contador que hará que, a cada 10 canciones seleccionadas, éstas puedan ser mandadas y el contador se vuelve a poner a 0.

Al llegar al final y terminar de recorrer el sistema debe comprobar si el contador es distinto de 0, en caso contrario se deberá a que esta última tanda de canciones no es distinta de 10 y se debe proceder a su envío.

5.3.2.5. Usuarios

Este último *fragment* el cual se recuerda que su acceso es exclusivo desde la Pantalla Principal de usuarios administradores, permite cargar la lista de usuarios almacenados en el servicio *User* y poder dar de baja de la sesión a cualquiera de ellos (expulsarlos)

Mencionar que el modelo utilizado para los usuarios utiliza el modelo *UserDevice* (*véase 5.3.3. Clases implementadas*) con su respectivo *Adapter* (*véase 5.3.6. Adapters*)

Para ello, el sistema debe ser vinculado en su creación al propio servicio y una vez obtenidos los usuarios comprobar que los haya (distinto de 0), estos son mostrados en una *ListView*. En caso de no haberlos, en lugar de mostrar la lista se mostrará un *TextView* indicando que no hay usuarios conectados actualmente.

En caso de haber usuarios, cada uno de ellos puede ser eliminado seleccionando el *TextView* asociado a cada uno que dice **Eliminar**. Sin entrar en detalles en las comunicaciones, decir que éste se encarga de enviar un mensaje al usuario cuyo correo está en la posición del ítem seleccionado obligándole a desconectar de la sesión.

5.3.3. Clases implementadas

Antes de continuar explicando funcionalidades complejas y, estando ya explicadas cómo funcionan las vistas de la aplicación de manera interna acorde a la necesidad de cumplir los objetivos marcados durante la fase de Análisis de la realización del presente proyecto, se pretende describir las clases implementadas que sirven de modelo para los objetos utilizados.

5.3.3.1. Wrapper de Spotify

Tal y como se menciona en apartados anteriores, el *wrapper* de acceso a los servicios ofrecidos por la Spotify Web API son bastante amplios y ofrecen una enorme cantidad de información sobre ellos: listado de todos los artistas de un determinado álbum, listado de imágenes asociadas a dicho álbum, las direcciones **uri's necesarias** para efectuar la **reproducción**. Y lo mejor de todo es que ofrece varias alternativas a la hora de obtener dicha información, ofreciendo para todas sus peticiones callback's para cargar la información en los objetos del modelo sin necesidad de efectuar tareas asíncronas.

Los elementos a destacar más representativos y utilizados durante la implementación del presente proyecto son los mostrados en la tabla 2.

Nombre	Descripción
Track	Representa a una canción en la Spotify Web API.
Album	Representa a un álbum en la Spotify Web API.
Playlist	Representa a una playlist en la Spotify Web API.
AlbumSimple	Elemento álbum con una menor carga de atributos disponible.
PlaylistSimple	Elemento playlist con una menor carga de atributos disponible.
AudioFeaturesTrack	Conjunto de propiedades para una determinada canción, como pueden ser *keys* o armadura de clave *(véase 5.3.4.3.3. Enviar y reproducir)*

Tabla 2. Objetos de modelo del Wrapper utilizados

5.3.3.2. UserDevice

Almacena la información de un determinado usuario de dispositivo móvil. Es utilizado tanto en *Devices* a la hora de mostrar resultados como en la sección *Users* disponible en la pantalla principal de aquellos usuarios administradores para mostrar aquellos usuarios que se encuentran conectados a su sesión. En dicho modelo se almacena el correo de dicho usuario, su nombre y la foto de perfil utilizada por el mismo (sólo cuando se utiliza para mostrar la lista de usuarios del administrador, en caso de búsquedas, se carga en dicho atributo un icono desde la carpeta *drawable*, tal y como se describe en el apartado 4. Diseño)

5.3.3.3. TrackSelect

Tal y como se menciona en el apartado *5.3.2.4. Lista de Canciones*, dicho elemento consta de un elemento *Track* del modelo proporcionado por el *wrapper* y de un **boolean** encargado de notificar si la Track se encuentra seleccionada o no, obteniendo la información desde la interfaz de usuario cada vez que se modifique.

5.3.3.4. Compatibility

Un par de valores que almacenan dos variables tipo ***int*** (**key y mode**) que representan una determinada clave musical de una canción concreta. Dicho punto es abordado en mayor profundidad en el Servicio Player.

5.3.3.5. SystemTrack

Utiliza como base el elemento *Track* ofrecido por el modelo del *wrapper*, junto al par de elementos descritos en el apartado anterior (información referente a la clave musical de la canción, donde **mode** (valor, 0 o 1) hace referencia a si la escala: mayor o menor, y **key** (valor, 0 a 11) hace referencia a la "armadura" de la clave. Este punto se procede a explicar de forma detallada en el apartado *Player*.

5.3.3.6. PlayerTrack

Hace referencia a un objeto definido seleccionando aquellos atributos ofrecidos por el elemento *Track* disponible en el *wrapper* que otorgan la información mínima a mostrar en el fragment *Reproductor* de aquellos usuarios no administradores.

Se procede a explicar con mayor profundidad en el apartado *5.3.4.1. PlayerViewService*.

5.3.4. Servicios

A lo largo del desarrollo de este punto *5. Implementación* se ha hablado de determinados servicios presentes en el código que dan pie a una serie de métodos que permiten interactuar al código de las *activities* y *fragments* con ciertos objetos ofrecidos por dichos servicios.

Dichos métodos son utilizados o no en función del rol desempeñado durante la ejecución de la presente aplicación descrita, siendo dos de ellos utilizados por los administradores: *Player* y *User, y* PlayerView por el resto.

Proceso de creación de un servicio

Antes de proceder a explicar cada uno de ellos, se procede a explicar la estructura utilizada para implementar los diferentes servicios, puesto que la metodología empleada ha sido la misma para los tres.

- Se define una interfaz con los métodos necesarios para conseguir la interacción entre el servicio y la aplicación.
- Se define una clase con los atributos necesarios para utilizar los métodos anteriormente definidos en la interfaz y se utiliza a la misma para implementar dicha interfaz (*implements*) El sistema tachará dicha acción como errónea al no encontrarse implementado el método *onBind* por parte del nuevo servicio. Basta con generar dicho método (*override*)

- Se define una variable de este último dentro de una nueva clase a la que a su vez esta última habrá sido extendida como servicio (*extends*)
- Se define dentro del propio servicio una clase que extienda del tipo *Binder* y que implemente un método que devuelva la implementación de la creada para implementar la interfaz, siendo esta última el tipo de retorno de dicho método. (comúnmente se han denominado en todos los servicios implementados como *getService()*)
- Se debe definir también un método *public static* que devuelva un Intent que devuelva el contexto del servicio.
- Finalmente se debe definir un último atributo en el servicio del tipo *IBinder* implementándolo sobre la clase que define el método getService, y se cambia el tipo de retorno del método *onBind* retornando ahora dicho atributo.

En la imagen 48, puede verse la definición de uno de los servicios (Usuarios) en Android Studio.

```
/*Servicio de gestion de usuarios*/
public class UserService extends Service{

    private final IBinder mBinder=new UserBinder();//binder (para obtener servicio)

    private final PreviewUser mUser=new PreviewUser();//instancia PreviewUser (implementaciones

    public static Intent getIntent(Context context){
        return new Intent(context,UserService.class);
    }

    public class UserBinder extends Binder{
        public  IUser getService() { return mUser;}//se retorna la interfaz con los metodos
        // proporcionados por la clase PlayerPreviewImpl proporcionada por el servicio
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        mUser.setAppContext(getApplication());
        return mBinder;
    }

    @Override
    public void onDestroy() { super.onDestroy(); }
}
```

Figura 48. Ejemplo de Servicio de la aplicación (Usuarios)

5.3.4. Servicios Implementados

A continuación, se procede a explicar qué ofrecen mediante sus respectivas interfaces a los usuarios y en caso del Reproductor *Player* qué es lo que se ejecuta internamente a la hora de solicitar acciones como son la inserción de una determinada canción.

5.3.4.1. PlayerView Service

Se comienza explicando este servicio accesible por usuarios no administradores. El cual se encuentra implementado por un par de atributos, siendo el primero un elemento del modelo *PlayerTrack* que se encarga de mostrar al usuario no administrador la canción actualmente en reproducción cuando sea necesario y una lista de los mismos elementos que representa las canciones próximamente inmediatas a reproducir.

Las funcionalidades ofrecidas por su interfaz permiten modificar los atributos previamente descritos y la obtención de los mismos. Se puede decir que se trata de un contenedor para que los usuarios no administradores guarden el estado actual de la lista de reproducción y puedan mostrarla cuando se precise.

Cuestiones sobre de dónde y cómo reciben dicha información serán abordadas durante las comunicaciones.

5.3.4.2. Users Service

Servicio que, tal y como se ha explicado anteriormente, es exclusivo para usuarios administradores de la aplicación, y que permite llevar a cabo la gestión de las altas y bajas de usuarios para con la sesión y poder obtener la lista de aquellos usuarios registrados en la misma. El objeto de modelo utilizado es *UserDevice* (*véase apartado 5.3.3.2. UserDevice*) y utilizando los atributos de correo y foto de perfil disponibles en el mismo.

En su interior alberga una lista de usuarios (Arraylist) donde a la hora de insertar alguno de ellos siempre se comprueba si dicho usuario (su correo) se encuentra ya registrado (evitar correos duplicados) En caso de no estarlo, envía un mensaje al usuario en cuestión comunicándole que ya puede entrar en la sesión (*véase 5.3.7. Comunicaciones*)

A la hora de gestionar la baja, lo primero que se comprueba si el correo del usuario a dar de baja, pertenece o no a la sesión. En caso de ser un usuario de la sesión, se le pide al usuario en cuestión cerrar su sesión y, una vez recibida confirmación del mismo, se elimina de la lista (al igual que para la acción de dar de alta, se explica el tema de las comunicaciones en el apartado en el susodicho)

5.3.4.3. Player Service

Aunque todos y cada uno de los componentes descritos hasta el momento son importantes y cada uno cumple una serie de requisitos y especificaciones consiguiendo a través de distintas interacciones entre ellos el cumplir los objetivos marcados a conseguir para este proyecto, se puede decir que el siguiente servicio es el motor principal del mismo, pues se trata del reproductor musical implementado.

A diferencia de los servicios descritos anteriormente, la complejidad de la gestión de los atributos alojados en el servicio es más mayor y a su vez, ofrece un catálogo de métodos a través de su interfaz bastante mayor.

Primero de todo, resumir el conjunto de funcionalidades que el mismo ofrece de cara a las comunicaciones con las diferentes *activities* y *fragments*:

- Inicializar variables: se encarga de inicializar los atributos disponibles en la implementación de la interfaz.
- Añadir el contexto de actual de la aplicación: este método se encuentra disponible en resto de servicios previamente explicados.
- Entregar objeto *SpotifyPlayer* configurado: este objeto pertenece a la librería *Player* de Spotify, y es la que permite llevar a cabo la reproducción. Dicho objeto es configurado en la activity *Devices* al crear una reproducción y entregado al servicio en dicho momento.
- Login del reproductor: Entrega el *token* almacenado en el gestor de credenciales al reproductor, para que este se lo entregue a su *SpotifyPlayer* e intente realizar el *Login*. Si el proceso ha ido bien o

mal (no es **Premium**) se devuelve mediante mensajes **broadcasts** (*véase 5.3.7. Comunicaciones o 5.3.8. BroadcastReceiver*)

- Obtener la canción en reproducción: el servicio devuelve un objeto *SystemTrack* con la información de la canción en curso (**null** en caso no encontrar resultado)
- Obtener cola próxima a reproducir: devuelve un *ArrayList* de elementos *SystemTrack* con las canciones más próximas a reproducir (máximo 10. En caso de no haberlas devolverá 0 elementos)
- El estado actual del dispositivo: si está reproduciendo o no.
- Si la cola de canciones pendientes está vacía o no.
- Enviar canciones: la funcionalidad más compleja disponible de la aplicación. En esencia se limita a insertar una canción en el reproductor, éste en caso de estar vacío la pondrá a reproducir y en caso contrario, la encolará. El proceso interno necesario para llevar a cabo esta acción puede llegar a ser un largo camino tal y como se verá en puntos posteriores de este apartado.
- Desconexión: el último de los métodos, se limita en desconectar el SpotifyPlayer del servicio (*logout*)

Llegados a este punto y explicadas las funcionalidades proporcionadas por el servicio, se procede a indagar más allá en el interior del mismo para comprobar cómo se lleva a cabo la gestión del mismo en aras de efectuar la reproducción y de gestionar canciones en su lista particular.

5.3.4.3.1. Lista de reproducción

Tal y como se comenta previamente, la gestión de los atributos del reproductor es más compleja que en el resto de servicios. Esto se debe principalmente a que aparecen más consideraciones a la hora de realizar los objetivos como puede ser una gestión eficiente de la lista de las canciones a reproducir.

La lista en sí se trata de un elemento tipo *Array* de elementos del modelo *SystemTrack* (*véase el apartado 5.3.3.5. System Track*) y un elemento contador que notifica en todo momento el número actual de elementos disponibles dentro de dicha lista. En este apartado se pretende justificar la presencia de la misma dentro de este servicio y qué métodos ofrece al mismo para que pueda ser gestionada de manera eficiente.

- Obtener la canción disponible en la primera posición (posición 0)
- Obtener las primeras canciones disponibles (máximo primeras 10)
- Obtener el número total de canciones que actualmente se encuentran en la lista.
- Devolver si la lista se encuentra vacía o no (contador a 0)
- Modificar su valor del contador de elementos por uno dado por parámetro.
- Insertar una canción en la última posición (marcado por el contador. Éste es incrementado)
- Obtener la lista completa de las canciones (*Arraylist*)
- Introducir una lista completa de las canciones (*Arraylist*)
- Desplazar la lista de canciones a partir de una posición dada (deja hueco vacío a partir de dicha posición, desplazando todo el contenido una posición a la derecha hasta llegar a la misma)
- Añadir una canción en una posición dada (método utilizado conjuntamente con el anterior): Permite introducir una canción en la posición indicada. Incremente el valor del contador total de canciones.

5.3.4.3.2. SpotifyPlayer

Volviendo al elemento encargado de implementar la interfaz especificada para este servicio, y recordando que su utilización se encarga de permitir la reproducción, se hace uso de un objeto del tipo *SpotifyPlayer* para ello y además deberá implementar:

- **ConnectionStateCallback**: se recuerda que, al igual que ocurre con la activity *Devices*, es necesaria la comprobación dentro del servicio de que el reproductor es apto para la reproducción (si es ***Premium*** o no en Spotify) Quedando el mismo activo cuando el login se efectúe sin problemas (*onLoggedIn()*)
- **Player.NotificationCallback**: encargado de implementar aquellos eventos que tendrán lugar a lo largo de la reproducción.

Para llevar a cabo la reproducción de una determinada canción, el único elemento totalmente imprescindible es la **URI** (identificador de recurso uniforme) de la misma, la cual es proporcionada de forma sencilla por los objetos de modelo explicados previamente (*véase 5.3.3.1. Wrapper de Spotify*)

Destacar aquí el método definido por la implementación (Player.NotificationCallback) previamente explicada, *oPlayerEvent*. Donde se registran aquellos eventos que toman lugar en todo momento en el reproductor, a la hora de definir dos situaciones que tendrán lugar en algún momento de la ejecución al terminar de reproducir una canción: que se cambie la canción porque todavía hay más pendientes de reproducir, o que no las haya, dando por finalizada la reproducción.

Cuando termina la reproducción de una canción, *PlayerEvent* produce uno de los siguientes eventos:

- ***kSpPlayBackNofityTrackChanged***: el reproductor ha dado por terminada una canción.
- ***kSpPlayBackNotifyTrackDelivered*:** el reproductor ha dado por finalizada con éxito la entrega de una canción.

5.3.4.3.2.1. Cambio de canción

Gestionado dentro del evento ***kSpPlayBackNofityTrackChanged*,** se encarga de comprobar que, no estando el reproductor reproduciendo alguna canción y habiendo más canciones pendientes a reproducir, coger la primera de la lista, desencolarla, pasar su **URI** a *SpotifyPlayer* y notificar a la interfaz de la actualización (*véase 5.3.8. BroadcastReceiver*). El proceso implementado puede verse en la figura 49.

```
case kSpPlaybackNotifyTrackChanged:
    if(!listTracks.empty() && !mCurrentPlayBackState.isPlaying){
        currentTrack=listTracks.getTop();
        listTracks.removeSong();
        mPlayer.playUri(mOperationCallback,currentTrack.getTrack().uri,0,0);
        intent=new Intent();
        intent.setAction("CURRENT_UPDATE");
        appContext.sendBroadcast(intent);
    }
    break;
```

Figura 49. Gestión del cambio de canción

5.3.4.3.2.2. La última canción

Gestionado dentro del evento ***kSpPlayBackNotifyTrackDelivered*** cuando termina una reproducción de una canción de forma correcta, se debe comprobar si hay más elementos en la lista para reproducir (en caso de haberlo, se procede a coger, al igual que en el caso anterior, el primer elemento, desencolarlo y pasar su **URI** a *SpotifyPlayer*) En caso contrario, se marca un **boolean** que indica el estado de la reproducción a **false** y el elemento *currenTrack* que representa la canción en reproducción actual a **null** y se notifica a la interfaz de la actualización *(véase 5.3.8.BroadcastReceiver)* El proceso puede verse en la figura 50.

```
switch (playerEvent){
    case kSpPlaybackNotifyTrackDelivered:
        if(!listTracks.empty()){
            currentTrack=listTracks.getTop();
            listTracks.removeSong();
            nextSongs=listTracks.getTail();
            mPlayer.playUri(mOperationCallback,currentTrack.getTrack().uri,0,0);
            playing=true;
        }else{
            currentTrack=null;
            playing=false;
        }
        intent=new Intent();
        intent.setAction("CURRENT_UPDATE");
        appContext.sendBroadcast(intent);
        break;
```

Figura 50. Gestión de fin de canción

5.3.4.3.3. Enviar y reproducir (Mezclador Armónico)

Para enviar una canción al sistema es necesario un objeto de elemento *SystemTrack* donde, se recuerda que se trata de un elemento *Track* con un par de atributos referentes de la misma, y que son proporcionados por *AudioFeaturesTrack.*

En la siguiente imagen (figura 51) se muestra un caso de ejecución del envío de una determinada *Track* al reproductor.

```
api.getService().getTrack(idTrack, new Callback<Track>() {
    @Override
    public void success(final Track track, Response response) {
        api.getService().getTrackAudioFeatures(track.id, new Callback<AudioFeaturesTrack>() {
            @Override
            public void success(AudioFeaturesTrack audioFeaturesTrack, Response response) {
                SystemTrack systemTrack = new SystemTrack(track, audioFeaturesTrack.mode, audioFeaturesTrack.key);
                mPlayer.sendSong(systemTrack);
            }

            @Override
            public void failure(RetrofitError error) {

            }
        });
    }
```

Figura 51. Añadir canción al reproductor

Lo que se trata de una simple llamada a un método ofrecido por la interfaz del servicio, internamente es un algoritmo elaborado en base a una serie de comprobaciones. La inserción **no debe tratarse** simplemente de encolar al final de la lista a no ser que se trate de que esté vacío o, en caso de no estarlo, no haya canciones pendientes. En caso contrario, realizarán **comprobaciones** y, un **mezclador** se encargará de gestionar acorde a dónde debe situarse dicha canción dentro de la lista.

¿Pero qué hace realmente el mezclador? Pues bien, más allá de los objetivos fijados en la mayoría del proceso de creación de este proyecto, surge la idea de no limitarse a seguir una política *FIFO (First In First Out*) a la hora de recibir una canción dentro del reproductor, sino que se pretende considerar otro tipo de información.

¿Cómo conseguir entonces una mezcla de las canciones, qué política seguir? Se pretende conseguir que las canciones sean añadidas lo más armónicamente posible, es decir, que la siguiente canción a reproducir no provoque una desnivelación en cuando a valores musicales se refiere. Para resolver a esta pregunta y tras un proceso de estudio, el DJ Mark Davis nos otorga una respuesta con su denominada *Cameloth Wheel* o Rueda de Cameloth la cual permite comprobar haciendo uso de las ***claves*** **(keys)** de las distintas canciones, comprobar cuáles son compatibles entre sí de manera armónica.

Según Mark [6], cualquier persona que haga utilización de su ya, empleada mundialmente tabla, podrá comprobar si dos canciones son compatibles entre sí para conseguir una mezcla armónica de las mismas. En la figura 52 puede verse una imagen de la *Cameloth Wheel*.

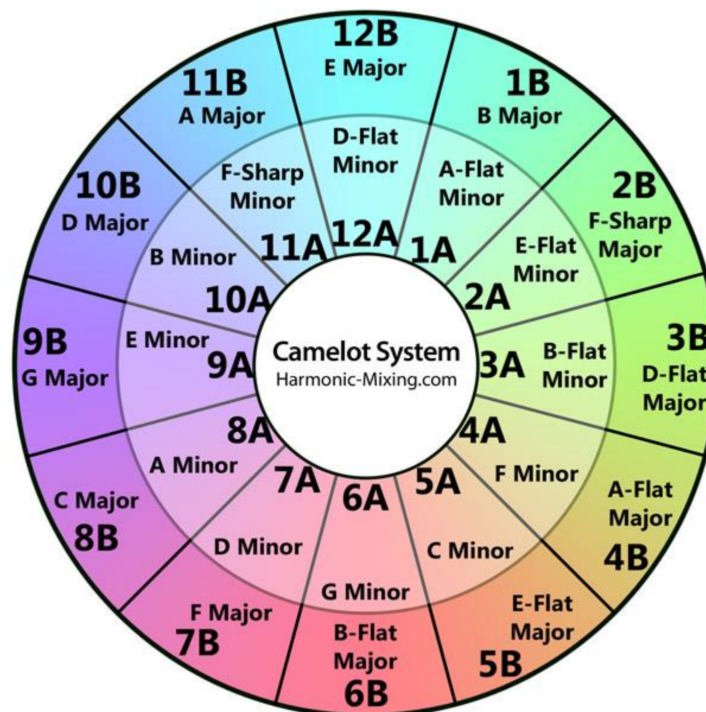

Figura 52. Cameloth Wheel [6]

Tal y como se puede apreciar en la figura anterior, en la rueda cada clave musical lleva asignado un código del 1 al 12, como si de un reloj se tratase. A su vez, se divide en dos sectores diferenciados (A y B, siendo el primero para las tonalidades menores y el segundo para las mayores)

La interpretación de la tabla es sencilla: una canción de una determinada clave (código de clave) será compatible con otra que trabaje con la misma clave, y con sus tres adyacentes: la correspondiente que tenga a cada lado y la del nivel opuesto de la misma clave (si es A, sería la B, y viceversa).

Por ejemplo: los códigos son compatibles con 1A, serían la propia 1A, 1B, 12A y 2A. Una 7B sería compatible con la propia 7B, 7A, 6B y 8B. Y así con todas.

Los códigos de clave representados tienen la siguiente correspondencia respecto a armaduras de claves musicales (tabla 3)

Cod.	Nivel	Clave
1	A	A-Flat Minor
2	A	E-Flat Minor
3	A	B-Flat Minor
4	A	F Minor
5	A	C Minor
6	A	G Minor
7	A	D Minor
8	A	A Minor
9	A	E Minor
10	A	B Minor
11	A	F-Sharp Minor
12	A	D-Flat Minor
1	B	B Major
2	B	F-Sharp Major
3	B	D-Flat Major
4	B	A-Flat Major
5	B	E-Flat Major
6	B	B-Flat Major
7	B	F Major
8	B	C Major
9	B	G Major
10	B	D Major
11	B	A Major
12	B	E Major

Tabla 3. Claves Cameloth Wheel

¿Cómo se interpreta dicha información en el servicio de reproducción? Primero de todo, hay que explicar cómo *Spotify* maneja la información referente a las claves. Se recuerda, que en el apartado de modelos se habla del elemento *Compatibility*, que permite gestionar un par de valores (*key* y mode) haciendo referencia a una armadura de clave. En la tabla número 5, se muestran las respectivas correspondencias de clases tonales.

- Mode es un valor "0" o "1" y representa la tonalidad, siendo 0 "Minor" y 1 "Major".
- Key es un entero de "0" a "11" que sigue la siguiente correspondencia de clave.

Clave	Contraparte tonal
0	C
1	D-flat
2	D
3	E-Flat
4	E
5	F
6	F-Sharp
7	G
8	A-Flat
9	A
10	B-Flat
11	B

Tabla 4. Correspondencia de clases tonales

5.3.4.3.3.1. Tabla de Compatibilidades

En base a lo anteriormente explicado, se procede a explicar el procedimiento llevado a cabo para hacer que nuestro reproductor pueda interpretar la información previamente descrita y poder definir su propia *Rueda de Cameloth* en base a la misma.

Se define una tabla compuesta por un *HashMap* donde cada clave del mismo será una correspondencia *mode-key* (objeto de modelo: Compatibility), sus valores serán una lista de elementos *Compatibility* también. De este modo, se consigue definir para cada uno de los emparejamientos *mode-key* posibles (24 en total) sus correspondientes compatibilidades aceptadas, siguiendo el comportamiento de la rueda.

Para desarrollar un algoritmo que permita rellenar de manera automática la tabla de forma correcta, se deben estudiar las correspondencias existentes entre los códigos de los sectores de la *Cameloth Wheel* con la gestión de claves tonales ofrecida por Spotify. En la figura 53, se pueden apreciar la correspondencia numérica para cada clave.

Pitch class

Pitch class	Tonal counterparts
0	C (also B♯, D𝄫)
1	C♯, D♭ (also B𝄪)
2	D (also C𝄪, E𝄫)
3	D♯, E♭ (also F𝄫)
4	E (also D𝄪, F♭)
5	F (also E♯, G𝄫)
6	F♯, G♭ (also E𝄪)
7	G (also F𝄪, A𝄫)
8	G♯, A♭
9	A (also G𝄪, B𝄫)
10, t or A	A♯, B♭ (also C𝄫)
11, e or B	B (also A𝄪, C♭)

Figura 53. Tabla de correspondencias de claves

Ahora, utilizando las correspondencias mostradas en la tabla, se analizan las compatibilidades posibles de las claves de la rueda con las correspondencias de claves reemplazando los códigos utilizados. Dicha tabla, se muestra en la figura 54.

MODE	KEY	NAME	WHEEL
0	0	c minor	5A
0	1	d-flat minor	12A
0	2	d minor	7A
0	3	e-flat minor	2A
0	4	e minor	9A
0	5	f minor	4A
0	6	f-sharp minor	11A
0	7	g minor	6A
0	8	a-flat minor	1A
0	9	a minor	8A
0	10	b-flat minor	3A
0	11	b minor	10A
1	0	c major	5B
1	1	d-flat major	12B
1	2	d major	7B
1	3	e-flat major	2B
1	4	e major	9B
1	5	f major	4B
1	6	f-sharp major	11B
1	7	g major	6B
1	8	a-flat major	1B
1	9	a major	8B
1	10	b-flat major	3B
1	11	b major	10B

Mode	Key	Comp [0]	Comp [1]	Comp [2]	Comp [3]
0	0	0-0	0-5	0-7	1-0
0	1	0-1	0-6	0-8	1-1
0	2	0-2	0-7	0-9	1-2
0	3	0-3	0-8	0-10	1-3
0	4	0-4	0-9	0-11	1-4
0	5	0-5	0-10	0-0	1-5
0	6	0-6	0-11	0-1	1-6
0	7	0-7	0-0	0-2	1-7
0	8	0-8	0-1	0-3	1-8
0	9	0-9	0-2	0-4	1-9
0	10	0-10	0-3	0-5	1-10
0	11	0-11	0-4	0-6	1-11
1	0	1-0	1-5	1-7	0-0
1	1	1-1	1-6	1-8	0-1
1	2	1-2	1-7	1-9	0-2
1	3	1-3	1-8	1-10	0-3
1	4	1-4	1-9	1-11	0-4
1	5	1-5	1-10	1-0	0-5
1	6	1-6	1-11	1-1	0-6
1	7	1-7	1-0	1-2	0-7
1	8	1-8	1-1	1-3	0-8
1	9	1-9	1-2	1-4	0-9
1	10	1-10	1-3	1-5	0-10
1	11	1-11	1-4	1-6	0-11

Figura 54. Correspondencias para la Tabla de Compatibilidades

Analizando la tabla anterior, la cual permitirá definir de una vez el algoritmo para inicializar la tabla de compatibilidades con todas sus correspondencias, se puede observar que dos de los casos a la hora de indicar cuáles son las adyacentes laterales compatibles no comienzan la secuencia con la clave 0, sino que una lo realiza con la 5 y la otra con la 7. Cuando en una columna se alcanza la clave número 11, la siguiente volverá a ser la 0, y así sucesivamente.

Con toda la información proporcionada por el proceso anteriormente descrito, se procede a definir el método de inicialización de la tabla.

- Se definen dos contadores: inicializados a 5 y a 7, para tratar con ellos las correspondencias adyacentes (se recuerda que en la tabla anterior se pueden observar los desplazamientos).
- Se recorre con un bucle externo 2 valores (0 y 1, que gestionen *mode*)
- Se recorre un bucle interno para 12 valores (0 a 11, que gestionen *key*)
- Se define una Arraylist del elemento *Compatibility*
- Se inserta el propio elemento de la iteración (su par *key-mode*)
- Se inserta el elemento de la tonalidad opuesta (si *mode* es 0, a 1 y viceversa)
- Se comprueba si el primer contador es mayor a 11, en caso contrario se vuelve a poner con el valor de clave a 0.
- Se inserta el elemento de la tonalidad con el modo correspondiente a la iteración y, como clave el valor de contador.
- Se realiza lo anteriormente descrito para el segundo contador.

Con esto finaliza la creación de la tabla que representa con total fidelidad la *Cameloth Wheel*.

Se ha definido un método que comprueba que, dada un par de elementos tipo *Compatibility* por parámetro, estos son compatibles entre sí.

Basta con utilizar uno de ellos como clave para buscar los valores del *HashMap* correspondiente y comprobar si el otro se encuentra en la lista de resultados compatibles o no.

5.3.4.3.3.2. Algoritmo de inserción

Para terminar este gran punto que es el servicio de reproducción de la aplicación, explicar cómo, utilizando todos los elementos descritos anteriormente se realiza un algoritmo que realice la inserción lo más armónicamente posible.

El algoritmo se limita a buscar dónde debe estar insertada cada nueva canción que quiera formar parte de la lista de reproducción, es decir, qué hueco le correspondería en base a una reproducción lo más armónicamente posible.

A la hora de tomar la decisión de insertar una canción, lo primero que debe hacerse es que la susodicha no se encuentre ya dentro del reproductor (no se insertarán canciones repetidas, aunque si es cierto que, si una canción ha sido reproducida y posteriormente, deshechada, entonces sí podrá ser insertada de nuevo sin problemas)

Se debe comprobar si existe alguna canción actualmente en reproducción y si hay o no alguna pendiente.

En caso de no estar reproduciendo o no haber canciones pendientes simplemente serán encoladas. Y, retomando el primer caso, poner a la misma inmediatamente a reproducir y desencolarla de la lista de canciones pendientes.

En caso de estar reproduciendo y haber una o más canciones pendientes a reproducir, se debe analizar con cuáles de las mismas (inclusive la actualmente en reproducción es compatible) e insertar la nueva canción **detrás de su última canción compatible**.

Para ello se comprueba si es compatible o no con la actual, y si es compatible o no con alguna de la lista (se utiliza una variable que indique la posición de la última canción seleccionada como compatible, siendo la misma inicializada a -1)

¿Qué puede ocurrir aquí? Que la canción sea compatible con alguna otra o no. En caso de no ser compatible con ninguna (ni de la lista, ni la que actualmente se encuentra en reproducción, simplemente se encola al final En caso contrario, se inserta tras la posición de la última compatible (si es compatible solo con la actual, se insertaría en la posición 0 como la nueva canción más inmediata a reproducir)

Para efectuar el desplazamiento se desplaza en la lista las canciones hacia la derecha un elemento hasta alcanzar la posición deseada, donde, posteriormente se procede a insertar la nueva canción.

5.3.5. Gestor de Credenciales

En general, decir que se encarga de almacenar el *token* resultado de la autentificación de Spotify, alojándolo en *SharedPreferences* pudiendo ser eliminado u obtenido cuando sea necesario. Este mecanismo nos proporciona poder acceder al *token* asignado por las credenciales de Spotify en todo momento, simplemente indicando por parámetro el conexto de la aplicación.

Cada vez que un usuario realiza alguna funcionalidad de la aplicación en la que Spotify esté involucrado: búsqueda de canciones, acceder a sus playlists… es necesario obtener el *token* para acceder a los servicios proporcionados por el *Wrapper*. Simplemente con una llamada al *getToken(...)* ofrecido, se consigue nuestro *token* particular.

5.3.6. Adapters

En varios puntos de la presente documentación se habla sobre unos determinados *adapters*. Éstos se encargan de vincular un determinado objeto de modelo a una determinada vista. Concretando, su utilización retoma la definición de *ítems* tanto para *ListViews*, donde se utilizan elementos del tipo *ArrayAdapter* como *RecyclerViews*, las cuales tienen su propio tipo de *Adapter* que permitan vincular una determinada vista (*layout*) con un determinado objeto de modelo.

En caso de definir un adaptador para *ListViews* o *RecyclerViews*, la gestión es muy similar.

Se debe indicar tanto el tipo de elemento de modelo que será tratado en el *Adapter*, definir un *Holder* con los atributos de la vista necesarios para ser vinculados con la misma e indicar qué atributos del modelo deben cargarse en qué atributos de la vista.

5.3.7. Comunicaciones

Si anteriormente se dijo que el *Reproductor* se trata del corazón de esta aplicación, del mismo modo se puede decir que las comunicaciones representarían las extremidades, pues se encarga de convertir una aplicación de usos limitados a un único usuario (aplicación monousuaria) en una aplicación que permita a otros usuarios de la aplicación puedan interactuar entre sí (aplicación multiusuario)

Para ello, se hace uso de la plataforma **nimBees**, la cual permite enviar desde un dispositivo concreto un objeto concreto a modo de mensaje y que éste sea recibido por otros que estén haciendo uso de la aplicación.

5.3.7.1. Mensaje

Para definir las comunicaciones, se utiliza un único objeto de modelo llamado *MyMessage* que contiene la información a tratar por los distintos dispositivos. Los atributos de dicha clase son los siguientes:

- Email del emisor del mensaje
- Modelo de dispositivo emisor del mensaje
- Email del receptor del mensaje
- Imagen del usuario emisor del mensaje
- Conjunto de identificadores de *Tracks* recibidas
- Contador de elementos a leer del conjunto de identificadores
- *ArrayList* de elementos *PlayerTrack*
- El tipo de conexión o mensaje: permite al *NotificationManager* encargado de recibir la notificación interpretar el mensaje adecuadamente.

5.3.7.2. Tipos de Mensaje

Se ha especificado en un tipo enumerado los posibles casos que pueden darse a la hora de cumplir los objetivos propuestos en el presente proyecto.

Teniendo en cuenta qué tipo de mensaje se esté enviando o recibiendo, se debe tratar una determinada información u otra. Permitiendo así la aparición de atributos ***null*** en un elemento del objeto de modelo anteriormente descrito (no será necesaria toda esa información en cada mensaje, sino que dependerá del tipo de mensaje que se desea enviar/recibir).

A continuación, se muestra la tabla 5, donde se recogen todos los tipos de mensajes implementados para abordar las comunicaciones, qué permite hacer cada uno y qué parámetros son necesarios para efectuar su envío de forma correcta.

Nombre	Acción	Atributos necesarios	Se recibe en
REQUEST_SEARCH_DEVICE	Solicita manifiesto a dispositivos móviles actualmente en reproducción.	emailOrigen tipo	MainUser
RESPONSE_SEARCH_DEVICE	Generado por un dispositivo móvil administrador en respuesta a la petición anterior. Devuelve correo y modelo del dispositivo, indicando como destino el email que ha solicitado la petición.	emailOrigen modeloOrigen emailDestino tipo	Devices
REQUEST_CONNECTION_DEVICE	El dispositivo origen solicita entrar en la sesión del usuario seleccionado. Se manda un mensaje a dicho usuario.	emailOrigen emailDestino tipo	MainUser
RESPONSE_CONNECTION_DEVICE	Se genera automáticamente en el dispositivo que ha recibido el mensaje anterior. El administrador debe comunicar al dispositivo que mandó la petición que puede acceder a su sesión, indicándole qué canciones tiene actualmente en la vista de Reproducción.	emailOrigen emailDestino tipo	Devices
REQUEST_DISCONNECT_BY_ADMIN	El administrador pide al dispositivo marcado como destino en este mensaje que debe cerrar su sesión.	emailOrigen emailDestino	MainUser
RESPONSE_DISCONNECT_BY_ADMIN	El dispositivo que ha recibido el mensaje anterior debe cerrar su sesión y mandar un mensaje de respuesta de confirmación al administrador para que lo elimine de su servicio	emailOrigen	MainUser

	de gestión de usuarios.	emailDestino	
REQUEST_DISCONNECT_BY_USER	Un dispositivo solicita desconexión a su administrador.	emailOrigen emailDestino	MainUser
RESPONSE_DISCONNECT_BY_USER	El usuario destino recibe la petición de desconexión del administrador.	emailOrigen emailDestino	MainUser
ABORT_DISCONNECT_BY_ADMIN	El administrador cierra su sesión y manda un mensaje reconocible por todos los usuarios conectados a la sesión cierren la activity de la misma.	emailOrigen	MainUser
REQUEST_SEND_SONG	Se solicita la inserción de canciones al administrador por un determinado dispositivo. Se deben leer los identificadores siendo el número total de estos definidos por un contador y proceder a la obtención de las canciones y sus respectivas inserciones.	emailOrigen emailDestino contador identificadores (String[] …)	MainUser
RESPONSE_UPDATE_SONG	El administrador devuelve a los usuarios conectados a la sesión una lista (*PlayerTrack*) de canciones para insertar en su servicio *PlayerView* (Actualizar la vista de la reproducción cuando proceda)	emailOrigen lista de canciones (*ArrayList* de elementos *PlayerTrack*)	MainUser

Tabla 5. Tipos de Mensajes

En la tabla anterior se muestra una sección donde se indica qué *Activity* de la aplicación debe recibir cada mensaje. Todos los mensajes son recibidos por el *NotificationManager* definido, donde son gestionados acorde a su tipo.

Aquí aparece un nuevo elemento que, aunque se ha utilizado en muchos de los puntos anteriormente descritos, y, aunque posea su propio apartado, es necesario introducirlo de manera breve. El ***BroascastReceiver*** es un componente ofrecido por Android que permite registrar eventos en la aplicación.

En el *NotificationManager* los mensajes son tratados acordes al tipo asociado al mismo y siguiendo lo anteriormente explicado en la tabla, tal y como puede verse en la figura 55.

```
public void handleCustomMessage(long idNotification, String content, Map<String, String> additionalContent) {
    MyMessage message=new Gson().fromJson(content,MyMessage.class);
    final String emailO=message.getEmailO();
    final String modelO=message.getModelO();
    final String emailD=message.getEmailD();
    final Uri imgO=message.getImgO();
    final TypeConnection type=message.getType();
    final int cont=message.getCont();
    final String[] ids=message.getIds();
    final List<PlayerTrack> ids_r=message.getIds_r();
    Intent intent=new Intent("DEVICES_CONNECTION");
    intent.putExtra("type",type);
    switch (type){
        case REQUEST_SEARCH_DEVICE:
            intent.putExtra("emailO",emailO);
            break;
        case RESPONSE_SEARCH_DEVICE:
            intent.putExtra("emailO",emailO);
            intent.putExtra("modelO",modelO);
            intent.putExtra("emailD",emailD);
            break;
        case REQUEST_CONNECTION_DEVICE:
            intent.putExtra("emailO",emailO);
            intent.putExtra("modelO", modelO);
            intent.putExtra("imgO",imgO);
            intent.putExtra("emailD",emailD);
            break;
        case RESPONSE_CONNECTION_DEVICE:
            intent.putExtra("emailO",emailO);
            intent.putExtra("emailD",emailD);
            intent.putParcelableArrayListExtra("ids_r", (ArrayList<? extends Parcelable>) ids_r);
            break;
        case REQUEST_DISCCONECT_BY_ADMIN:
            intent.putExtra("emailO",emailO);
            intent.putExtra("emailD",emailD);
```

Figura 55. NotificationManager de la aplicación

En función del tipo de mensaje, el elemento del modelo *PlayerTrack* debe ser convertido a *Parceable* para poder pasarlo a través del *bundle* de un *intent* cuya acción pueda ser reconocida por el *BroadcastReceiver* de una determinada activity.

El *broadcastReceiver* en cuestión comprobará la acción asociada a un *intent* recibido, siendo "DEVICES_CONNECTION" la definida para las comunicaciones. A continuación, se muestra un ejemplo de su aplicación en el proyecto a través de la figura 56.

```
if(intent.getAction().equals("DEVICES_CONNECTION")){//si se reconoce un nuevo mensaje
    Bundle bundle=intent.getExtras();
    TypeConnection type=(TypeConnection)bundle.get("type");//se obtiene el tipo
    String emailO,modelO,emailD,modelD;
    Uri imgO;
    String[] aux={};
    if(type!=null) {
        switch (type) {
            case REQUEST_SEARCH_DEVICE:
                //si es administrador
                if (isAdmin) {
                    emailO = bundle.getString("emailO");
                    emailD = acct.getEmail();
                    modelD = NimbeesClient.getUserManager().getUserData().getDeviceModel();
                    if (mLocation == null) {
                        getLocationAndSendMessage(emailD, modelD, emailO, null, TypeConnection.RESPONSE_SEARCH_DEVICE,
                    } else {
                        helperConnection.sendMessage(RADIUS, mLocation, emailD, modelD, emailO, null, TypeConnection
                    }
                }
                break;
            case REQUEST_CONNECTION_DEVICE:
                emailO = bundle.getString("emailO");
                modelO = bundle.getString("modelO");
                emailD = bundle.getString("emailD");
                imgO = (Uri) bundle.get("imgO");
                if (acct.getEmail() != null) {
                    if (acct.getEmail().equals(emailD)) {
                        mUsers.addNewUser(new UserDevice(emailO, modelO, imgO));
                        List<PlayerTrack> songsToSend = new ArrayList<>();
                        if(!mPlayer.isEmpty() || mPlayer.isPlaying()) {
                            Track currentTrack = mPlayer.getCurrentTrack().getTrack();
                            String title=currentTrack.name;
```

Figura 56. Ejemplo de BroadcastReceiver para comunicaciones

Tal y como se aprecia en la imagen anterior, la ejecución varía acorde al tipo de mensaje, destacando los métodos *sendMessage* y *getLocationAndSendMessage* que dependen si un atributo del tipo *Location* se encuentra a **null** o no.

Este atributo marca la localización actual, y en caso de no poseerlo, el segundo método anteriormente nombrado se encarga de hacerlo y, una vez obtenido, envía el mensaje.

5.3.8. BroadcastReceiver

Tal y como se define en el apartado anterior, este tipo de atributos permite a la aplicación notificar de la aparición de determinados eventos, pudiendo ser registrados los que necesitemos según su acción.

Esto ha permitido que, durante la ejecución de la aplicación, si termina una canción y pase otra a reproducir dentro del servicio, las vistas sean actualizadas acorde a la situación actual dentro del servicio, o que, si un nuevo usuario pasa a formar parte de la sesión de este dispositivo y nos encontrásemos en la vista de *Usuarios*, ésta se pueda actualizar de manera automática.

En otras palabras, se consigue comunicar a los servicios con la interfaz de usuario para que, de cierta manera, ésta pueda ser gestionada por los mismos (ejemplo en figura 57).

```
BroadcastReceiver broadcastReceiver=new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        switch (intent.getAction()){
            case "CURRENT_UPDATE"://se actualiza la lista en la interfaz
                if(mPlayer!=null) {
                    updateList();
                    getActivity().sendBroadcast(new Intent("UPDATE_ALL_USERS"));
                }
                break;
```

Figura 57. Ejemplo de broadcastReceiver en fragment Reproductor

5.3.9. Manifest

A continuación, se muestra el archivo *Manifest.xml* el cual proporciona información al sistema para que pueda efectuar la ejecución de la aplicación.

En el archivo se especifican las *activities* y servicios utilizados para efectuar la presenta aplicación, así la ruta de paquete del archivo encargado de personalizar la clase *Application*.

En la parte superior del mismo, se encuentra el permiso de acceso a **Internet** y a los **sistemas de localización**, por parte de la presente aplicación. A continuación, se muestra en la figura 58 el archivo *manifest* del proyecto.

```
<application
    android:name=".application.MyApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="SpotAndJoyProject"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
        android:name=".activities.SignIn"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".activities.Devices"
              android:screenOrientation="portrait"/>


    <activity
        android:name=".activities.MainUser"
        android:label="@string/title_activity_main_user"
        android:theme="@style/AppTheme.NoActionBar"
        android:screenOrientation="portrait"/>

    <service android:name=".player.PlayerService" />
    <service android:name=".users.UserService"/>
    <service android:name=".playerView.PlayerViewService"/>
```

Especificación de la clase Application personalizada para esta aplicación

Activities

Servicios

Figura 58. Documento Manifest.xml

5.4. Pruebas

Aunque no se trate del apartado más extenso, se trata de una etapa que, junto a la implementación, son las etapas que más tiempo han llevado a cabo durante la realización del presente proyecto.

Para garantizar el cumplimiento de todos los objetivos propuestos durante el desarrollo de las distintas fases de la aplicación, y haciendo uso de los teléfonos móviles *smartphones* descritos en el apartado *4.1.2-Smartphones*, se ha procedido a realizar multitud de escenarios posibles con los tres dispositivos, siendo el administrador uno de ellos (**se ha contratado una cuenta Spotify Premium para la realización de este proyecto**)

Con la ejecución de las pruebas aparecen limitaciones referentes al uso de esta aplicación, pero que no interfieren con la lista de objetivos finales a conseguir:

- Es recomendable que las sesiones de reproducción se realicen sin salir del alcance definido para la recepción de los mensajes respecto al dispositivo reproductor.
- Cuanto mayor sea el número de canciones a insertar o mayor sea el número de peticiones de otras a reproducir, pueden conseguir un gran consumo de recursos del dispositivo móvil, por lo que se recomienda una gama media-alta para poder disfrutar de la aplicación de la manera más fluida posible.
- No se deben utilizar los datos si se quiere ser un usuario administrador, incluso si se posee de una buena cobertura WiFi mucho mejor.

6. RESULTADOS Y DISCUSIÓN

En base a todo lo descrito anteriormente, se procede a comentar los resultados obtenidos durante las etapas realizadas a lo largo del desarrollo del presente proyecto. Se recalca el éxito total en lo referente al cumplimiento de los objetivos finales marcados a conseguir durante todo el desarrollo del proyecto.

Se ha procedido a desarrollar una aplicación que permite reproducir canciones para usuarios que posean una cuenta de Spotify ***Premium***, esta restricción no estaba prevista inicialmente, y restringe las funcionalidades del Administrador a aquellos usuarios que sí la posean.

Ciertos comportamientos provocan un deterioro del sistema, como puede ser el envío de muchas canciones por distintos usuarios al mismo tiempo, a un mismo administrador. Pensamos que puede tratarse debido al volumen de información a tratar en el dispositivo móvil receptor: deberá obtener las canciones, una a una, en función de su identificador. Una vez obtenida, se procede a conseguir su clave tonal y finalmente, se procede a añadirla al reproductor, donde el algoritmo de inserción debe realizar las comprobaciones adecuadas (que no esté ya en lista, constatar si la clave tonal es compatible con alguna de las ya insertadas…) Un proceso largo, donde muchas tareas entran en juego.

Cada vez que se produce alguna variación dentro del reproductor (cambie la canción actual a reproducir o haya alguna inserción en la lista de canciones pendientes) se procede a realizar el refresco de la lista de reproducción de todos los dispositivos (actualización automática). Para ello, el administrador deberá enviar un mensaje con la información actualizada. Este proceso repetido a lo largo del tiempo, supone un gran número de mensajes a enviar y considerando que se está utilizando la plataforma **nimBees** de **forma gratuita**, se posee una limitación de mensajes totales disponibles a enviar al mes (10000 mensajes máximo).

Estas contemplaciones pueden provocar un gasto continuo tanto de recursos del dispositivo móvil que ejerza de administrador (procesador, memoria, batería…), como limitar bastante el uso de la aplicación total a lo largo de un determinado mes.

Sería muy recomendable que si se desea albergar una sesión donde un número de usuarios elevado (más de dos) quiera unirse a la sesión de reproducción, se deba poseer un dispositivo de gama media-alta como mínimo.

Tal y como se menciona en el apartado anterior, otra limitación a considerar es la de encontrarse en las proximidades de algún *router* o dispositivo que permita una conexión Wi-Fi (cuanto mayor sea la conexión, mejor)

También se deberá tener activada la geolocalización del teléfono móvil activada para que se pueda efectuar el envío de los mensajes.

Otro aspecto no considerado en ningún momento es la gestión por parte dos dispositivos (siendo uno administrador y otro un usuario conectado a la sesión del anterior) cuando se separan una distancia significativa: el administrador seguiría reproduciendo y dicho usuario aparecería en su lista de usuarios. El usuario no seguirá dentro de la sesión y no tendrá acceso a ejecutar ninguna funcionalidad y deberá reiniciar la aplicación. El administrador por otro lado seguirá funcionando correctamente. Para evitar estos inconvenientes, se recomienda que el dispositivo administrador se encuentre en un punto estático y que aquellos usuarios que quieran hacer uso de sus servicios no se alejen demasiado.

7. CONCLUSIONES

En el presente apartado se pretende comentar las conclusiones finales y personales referentes al presente proyecto, siendo extraídas del largo proceso llevado a cabo durante todo el desarrollo del mismo.

En lo que se refiere al desarrollo de cualquier tipo de aplicación desde cero, partiendo de una idea, he de añadir que se trata de una de las experiencias más gratificantes y complejas . Partiendo de hipótesis y suposiciones, y mediante un largo proceso de estudio y aprendizaje, lo que comienza siendo una idea en tu cabeza pasa a ser plasmada en un papel, y de ahí, a una aplicación.

La presente aplicación ofrece a usuarios poder compartir sus canciones de sus respectivas *Playlists* en *Spotify*. Partiendo de esa frase, se ha desarrollado una aplicación móvil funcional que, además de lo anterior descrito, permite buscar canciones bajo criterios, administrar usuarios bajo el rol de un administrador el cuál será el encargado de ejercer la reproducción y un control de éste sobre el resto de usuarios que estén conectados a la sesión.

Destacar también el haber aprendido a realizar un proyecto de principio a fin, respetando sus etapa y sobretodo, los conocimientos sobre Android adquiridos a lo largo del desarrollo.

El futuro de Spot&Joy

¿Qué futuro le espera a Spot&Joy? Lo que *a priori* aparenta ser un producto finalizado, siempre se puede seguir mejorando, siempre habrá nuevas ideas.

La aplicación se encuentra disponible para sistemas operativos *Android,* pero, ¿qué hay de aquellos usuarios de otros sistemas operativos? Actualmente, con las tecnologías empleadas, se podría llevar a cabo de la aplicación en el sistema operativo *iOS* con el fin de conseguir que cualquier dispositivo iOS o Android, pueda conectarse a una sesión de reproducción sin importar el sistema operativo empleado.

Por otro lado, se pueden implementar diferentes políticas de reproducción siendo ésta elegida por el administrador de la sesión. Incluso generar estadísticas que muestren información sobre a todos los usuarios diferentes *tops* como artistas más escuchados o géneros más reproducidos…

Las oportunidades terminan donde la imaginación de uno lo hace.

REFERENCIAS BIBLIOGRÁFICAS

1. **FEKI, M. A., KAWSAR, F., BOUSSARD, M. , TRAPPENIERS L.** ***"The Internet of Things: The Next Technological Revolution".*** **2, 2013, Computer, Vol. 46, págs. 24-25.**

2. **SPOTIFY DEVELOPERS.** ***Spotify Android SDK.*** **https://developer.spotify.com/technologies/spotify-android-sdk/**

3. **SPOTIFY DEVELOPERS.** ***Spotify Android SDK.*** **https://developer.spotify.com/design/**

4. **StackOverflow. https://es.stackoverflow.com/**

5. **Spotify.** ***Wikipedia.*** **https://en.wikipedia.org/wiki/Spotify**

6. **Pitch Class.** ***Wikipedia.*** **https://en.wikipedia.org/wiki/Pitch_class**

7. **iOS vs Android. https://www.paradigmadigital.com/dev/versus-desarrollo-ios-vs-desarrollo-android/**

8. **How to Guide.** ***Harmonic Mixing.*** **http://www.harmonic-mixing.com/HowTo.aspx**

9. **Curso Android. http://www.androidcurso.com/.**

10. **Console.** ***Google Cloud.*** **https://console.cloud.google.com/**

11. **BroadcastReceiver.** ***Android Developers.*** **https://developer.android.com/reference/android/content/BroadcastReceiver.html**

12. **AsyncTask.** ***Developers Android.*** **https://developer.android.com/reference/android/os/AsyncTask.html**

13. **nimBees Platform. https://api.nimbees.com/**

14. **nimBees Documentation.**

http://doc.nimbees.com/index.php/Getting_Started

15. **Add Google Sign-In to Your Android App.** ***Google Developers.*** **https://developers.google.com/identity/sign-in/android**

16. Spotify Developers. ***Spotify Web API.*** **https://developer.spotify.com/web-api/**

17. BERROCAL, J., GARCÍA-ALONSO J., MURILLO RODRÍGUEZ, J.M., CANAL C. " ***Situational-Context: A Unified View of Everything Involved at a Particular Situation." ICWE 2016, págs 476-483.***

18. FLO Music. http://www.flomusic.com/

19. OutLoud. https://outloud.dj/

20. Services. ***Android Developers.*** **https://developer.android.com/guide/components/services.html?hl=es-419**

ANEXOS

Para desarrolladores

Si se intenta proceder a ejecutar el presente proyecto desde *Android Studio*. Es recomendable seguir el procedimiento presentado en el apartado 5.1.5. Conclusiones y complicaciones durante la fase de estudio en el caso de encontrar problemas al abrir e intentar lanzar la presente aplicación.

Manual de Usuario

A continuación, se pretende explicar mediante capturas de pantallas realizadas a la presente aplicación cómo se utiliza la misma.

Se presenta Spot&Joy, una aplicación con la que usted podrá compartir sus listas de reproducción de Spotify junto a otros usuarios.

Las canciones serán reproducidas por un único usuario el que hará de administrador con la condición de que posea una cuenta **Spotify Premium**.

Cuando uno ejecuta la aplicación, se observa una pantalla de bienvenida, con el logo de la aplicación y un botón. Pulsando dicho botón de iniciar sesión el sistema pide qué cuenta de correo *Gmail* quiere ser utilizada para esta sesión (véanse figuras 59 y 60).

Figura 59. Pantalla Inicial

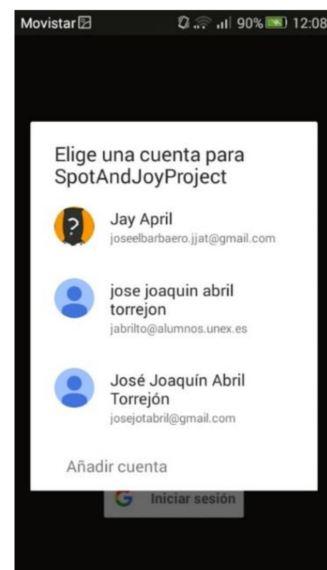

Figura 60. Seleccionar correo

Se accede en este punto a una lista que inicialmente aparece vacía. Dicha lista se limita a mostrar los resultados al realizar una búsqueda.

En la pantalla actual (figuras 61 y 62) se pueden realizar tres operaciones asociadas a tres botones disponibles en la barra inferior de la misma:

- Salir, que nos devuelve a la pantalla de bienvenida.
- Buscar dispositivos, devolverá una lista de dispositivos administradores disponibles dentro del alcance de la aplicación.

Figura 61. Lista de Dispositivos (sin resultados)

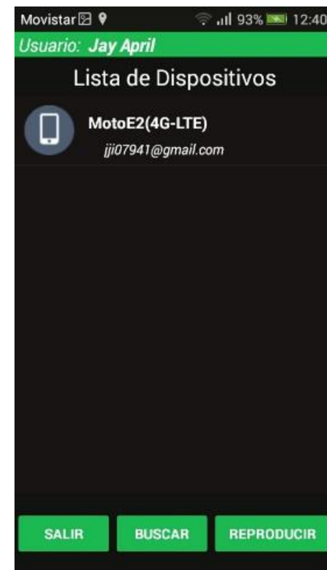

Figura 62. Lista de Dispositivos (con resultados)

- Administrar, permite crear una sesión haciendo uso de unas determinadas credenciales de **Spotify** con la condición tal y como se ha especificado anteriormente de que dicha cuenta sea ***Premium***.

Retomando este último paso, cuando se ejecuta el botón *Reproducir* se lanzará una ventana ofrecida por Spotify para introducir nuestras credenciales (figura 63).

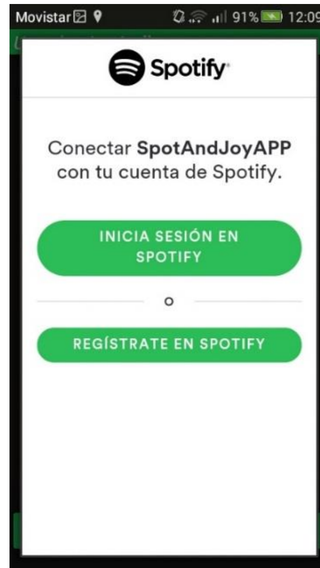

Figura 63. Credenciales de Spotify

En caso de no ser correctas, lanzará un mensaje avisando de que las credenciales del usuario introducidas son incorrectas (figura 64).

Figura 64. Credenciales de Spotify incorrectas

Si las credenciales han sido correctas, pero no pertenecen a una cuenta ***Premium***, no se podrá iniciar la sesión y se indicará con un mensaje mostrado en la pantalla indicando que es necesario una cuenta Premium para continuar.

Si las credenciales han sido validadas, iremos a la pantalla principal de la aplicación, y asignándonos el rol de **administrador** tendremos el control del reproductor de la sesión. Es decir, nuestro móvil podrá reproducir música (figura 65).

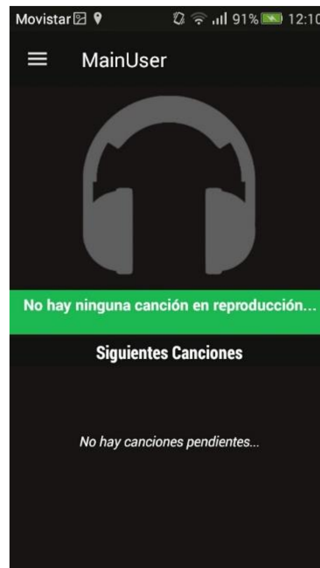

Figura 65. Reproductor admin

La pantalla que se observa inicialmente corresponde al reproductor, ofreciendo una vista del estado actual del mismo (como inicialmente no hay canciones en curso, nos indicará que está vacío)

Si se extiende el menú lateral se comprobará que hay diferentes funcionalidades al alcance del administrador (figura 66).

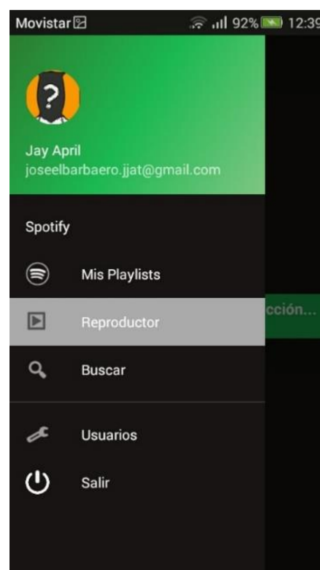

Figura 66. Pantalla principal (admin)

- Mis playlists: muestra las listas de reproducción que el usuario posee actualmente en su cuenta de *Spotify* (figura 67).

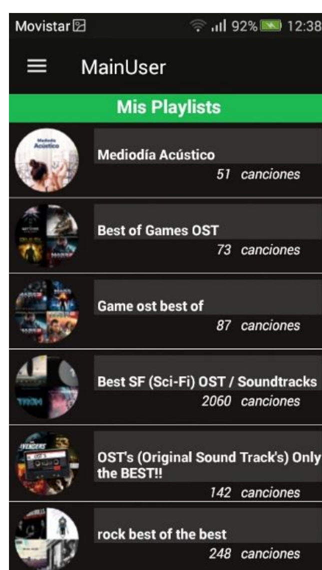

Figura 67. Mis playlists

- Buscador: permite buscar canciones de Spotify dentro de algún criterio (álbum, playlist o canción. Figura 68)

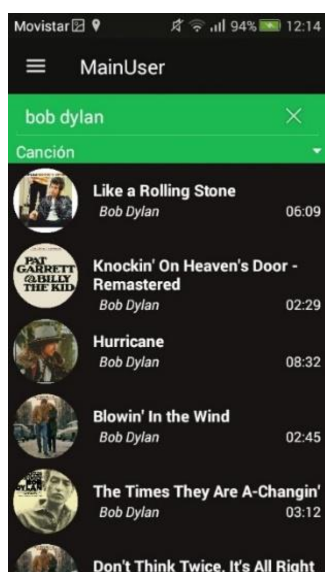

Figura 68. Buscador

Para los dos primeros criterios (playlists o álbum), cuando se seleccione algún resultado concreto, la vista cambiará y en su lugar mostrará una lista de canciones del álbum o playlist seleccionados.

En caso de haber buscado por una canción, la misma habrá pasado a reproducirse y se cambia la vista ofreciendo ahora en la vista de reproducción qué canción está siendo ejecutada actualmente.

Si, por el contrario, tal y como se menciona anteriormente, para el álbum o la playlist que fuese seleccionado, se procede a mostrar una vista con las canciones contenidas por dicho elemento (figura 69)

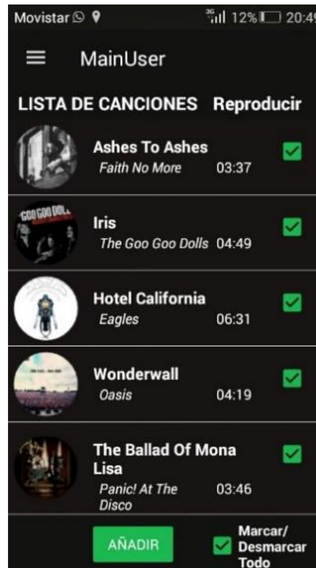

Figura 69. Lista de Canciones

En dicha vista el usuario podrá:

- Seleccionar exclusivamente aquellas canciones que se quieran enviar: haciendo uso de la casilla *Marcar/Desmarcar todas* o con marcación manual por cada elemento de la lista, se puede decidir cuáles canciones serán enviadas al reproductor (sólo lo harán aquellas que figuren como seleccionadas)
- Enviar canciones: pulsando el botón Enviar, las canciones pasarán al reproductor, las cuales pasarán a ser puestas en una lista de espera. Cuando el reproductor se encuentre vacío y reciba canciones, éstas pasarán a reproducirse de manera automática.

Si se ha enviado una canción (figura 70), la vista será como la de la imagen mostrada en la parte izquierda a continuación. Por el contrario, si fueron varias (figura 71), la imagen será como la de la derecha. Para este último caso, mencionar que la lista de canciones pendientes mostradas será de un máximo de diez (las primeras pendientes)

Figura 70. Reproductor (una canción)

Figura 71. Rerproductor (varias canciones)

Cada vez que se acceda desde el menú lateral a la opción “Reproductor”, se podrá previsualizar el estado actual de la reproducción bajo las condiciones previamente descritas.

Podrá pausar la canción en curso siempre que quiera, simplemente pulsando en la imagen correspondiente a la canción en curso (cuando esté reproduciendo).

En el menú aparece una opción llamada *Usuarios* y que es exclusiva para el administrador de la sala. En ella, podrá comprobar en tiempo real qué otros usuarios ejecutando la aplicación se encuentran actualmente dentro de su sesión de reproducción, permitiendo eliminar a cualquiera de ellos en cualquier momento (se puede evitar de este modo la presencia de usuarios molestos o no deseados)

La última opción sería la de Salir donde, si la aplicación se encuentra actualmente reproduciendo música, mostrará un mensaje avisando al usuario de la situación y recordándole que, si confirma querer salir, toda la información pendiente a reproducir o actualmente en reproducción se perderá (figura 72). Si se acepta, se cierra la sesión y se vuelve la vista de Dispositivos.

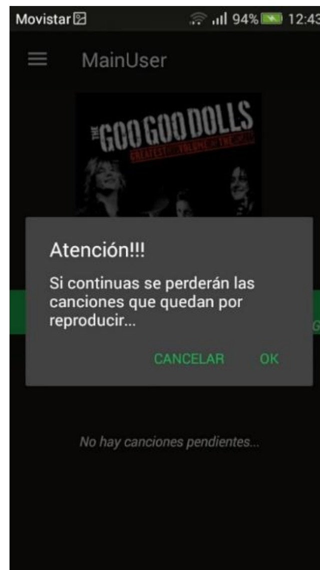

Figura 72. Salir (admin)

Suponemos que no tenemos una cuenta **Premium**, pero se puede acceder a un dispositivo que sí la tenga. Pues bien, si dicho dispositivo está cerca de nosotros (una distancia preferiblemente menor a 10 metros) ejecutando la aplicación *Spot&Joy* podemos buscarlo tal y como se vio anteriormente desde la lista de dispositivos una vez hayamos pulsado el botón buscar.

Ahora, se supone que hemos encontrado un dispositivo que actualmente se encuentra como administrador de una sesión. En dicho caso, se seleccionaría dentro de la lista y, tras un breve periodo de espera en el que el susodicho esté validando nuestro acceso, nuestra pantalla su cambiará y, al igual que ocurría con el administrador, se muestra el estado actual de la reproducción (el administrador actualiza cada vez que tiene cambios en su reproductor, a todos los usuarios conectados a su reproducción, figuras 73 y 74)

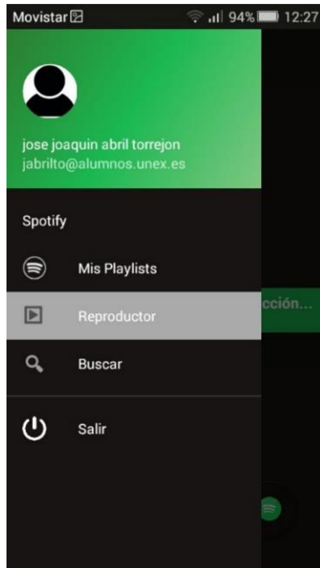

Figura 73. Pantalla Principal (estándar)

Figura 74. Vista del Reproductor (estándar)

No es completamente necesario poseer una cuenta de Spotify para poder ser un usuario estándar, pero nuestras actividades se limitarán a poder observar la pantalla anteriormente descritas. En caso de no tener las credenciales en la parte inferior de la pantalla aparece un elemento seleccionable, simplemente muestra un mensaje al usuario animando a registrarse o en Spotify o introducir sus credenciales en la aplicación.

Se sigue teniendo acceso a las opciones *Mis Playlists* y *Búsqueda* solo que, a la hora de seleccionar una canción o un grupo de canciones a reproducir, no lo harán en nuestro dispositivo. En su lugar, las canciones serán enviadas al reproductor el cuál se encargará de ponerlas pendientes o ponerlas a reproducir cuando quepa lugar. Para poder acceder a ellas se debe poseer al menos una cuenta **Spotify free** (Al seleccionar dichas opciones por primera vez, cargará la vista que permite introducir las credenciales de Spotify)

Se recuerda que el administrador puede expulsarnos de la sala en todo momento, en dicho caso, aparecerá un mensaje indicando que hemos sido expulsados por el administrador.

Si se desea salir de la sesión para buscar alguna más o dejar de utilizar la aplicación, basta con pulsar la opción *Salir* del menú.