

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN UNIVERSIDAD DE MÁLAGA



PROYECTO FIN DE CARRERA

**APLICACIÓN ANDROID PARA
MEDICIÓN DE RUIDO AMBIENTAL
Y SU REPRESENTACIÓN
GEOLOCALIZADA**

**INGENIERÍA TÉCNICA DE
TELECOMUNICACIÓN**

Málaga, 2015

Guillermo Orellana Ruiz

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD DE MÁLAGA

**Titulación: Ingeniería Técnica de Telecomunicación
Especialidad en Sonido e Imagen**

Reunido el tribunal examinador en el día de la fecha, constituido por:

D./D^a. _____

D./D^a. _____

D./D^a. _____

para juzgar el Proyecto Fin de Carrera titulado:

**APLICACIÓN ANDROID PARA MEDICIÓN DE RUIDO
AMBIENTAL Y SU REPRESENTACIÓN
GEOLOCALIZADA**

del alumno *D./D^a. Guillermo Orellana Ruiz*
dirigido por *D./D^a. Enrique Márquez Segura*

ACORDÓ POR _____ OTORGAR LA

CALIFICACIÓN DE _____

y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia.

Málaga, a ____ de ____ de ____

El Presidente:

El Vocal:

El Secretario:

Fdo.:_____ Fdo.:_____ Fdo.:_____

Universidad de Málaga
Escuela Técnica Superior de Ingeniería de
Telecomunicación

APLICACIÓN ANDROID PARA MEDICIÓN DE RUIDO AMBIENTAL Y SU REPRESENTACIÓN GEOLOCALIZADA

REALIZADO POR
Guillermo Orellana Ruiz

DIRIGIDO POR
Enrique Márquez Segura

Dpto. de: Ingeniería de Comunicaciones (IC)

Palabras clave: ruido, android, aplicación, geolocalización, mapa, calor

Titulación: Ingeniería Técnica de Telecomunicación Especialidad en Sonido e Imagen

Resumen: El presente proyecto consiste en desarrollar una aplicación móvil para el sistema operativo Android, cuya funcionalidad es la medición del nivel de ruido asociado a un punto geográfico, y la representación del mismo superpuesto a un mapa geográfico

Málaga, 10 de febrero de 2015

Este Proyecto Fin de Carrera está dedicado a mis padres,
por el apoyo y la paciencia que siempre me brindan,
y que sin duda ha sido esencial para llegar hasta donde estoy.

El autor

Acrónimos

API	Application Programming Interface, o interfaz de programación de aplicaciones
CAG	Control Automático de Ganancia
CSV	Comma Separated Values, o valores separados por coma
DSL	Domain Specific Language, o lenguaje específico del dominio
ETSIT	Escuela Técnica Superior de Ingeniería de Telecomunicación
GMS	Google Mobile Services, o servicios móviles de Google
GPS	Global Positioning System, o sistema de posicionamiento global
HAL	Hardware Abstraction Layer, o capa de abstracción del hardware
IDE	Integrated Development Environment, o entorno de desarrollo integrado
JSON	JavaScript Object Notation, o notación de objetos de Javascript

MEMS	Microelectromechanical Systems, o Sistemas Microelectromecánicos
PCM	Pulse Code Modulation, o modulación por impulsos codificados
PFC	Proyecto Fin de Carrera
POO	Programación Orientada a Objetos
SI	Sistema Internacional de medidas
SPL	Sound Pressure Level, o nivel de presión sonora
SWL	Nivel de potencia acústica
UMA	Universidad de Málaga
XML	eXtensible Markup Language, o lenguaje de marcado extensible

Índice

Objetivos del Proyecto	1
Motivación	1
Estado del arte	2
Herramientas y metodologías utilizadas	3
Estructura del documento	4
1 Introducción teórica	7
1.1 Acústica	7
1.1.1 Nivel de presión sonora	7
1.1.2 Potencia acústica y densidad espectral de potencia	8
1.1.3 Ruido	9
1.1.4 Modulación por impulsos codificados	10
1.2 Android	12
1.2.1 Interfaz	13
1.2.2 Componentes de una aplicación	14
1.2.3 Vistas	20
1.2.4 Procesos e hilos de ejecución	20
1.2.5 Manejo de archivos	22
1.2.6 Ubicación	23
1.2.7 Mapas	24

1.3	Otros	25
1.3.1	Control automático de ganancia (CAG)	25
1.3.2	Mapa de calor	26
2	Implementación	29
2.1	Análisis de requisitos	29
2.1.1	Diseño de la interfaz gráfica	30
2.1.2	Dispositivo Android	32
2.1.3	Micrófono externo	32
2.1.4	Calibración	33
2.2	Aplicación Android	33
2.2.1	Entorno de desarrollo integrado	33
2.2.2	Estructura del código	35
2.2.3	Interfaz gráfica	37
2.2.4	Captura de audio	43
2.2.5	Librerías externas	47
3	Plan de pruebas y verificación	55
3.1	Funcionamiento de la aplicación	55
3.2	Precisión de las medidas	61
3.2.1	Realización de medidas en laboratorio	61
3.2.2	Realización de medidas de campo	63
3.3	Realización de los objetivos	65
3.3.1	Capacidad de medir la magnitud del ruido ambiente	65
3.3.2	Capacidad de determinar la posición del dispositivo	65
3.3.3	Capacidad de asociar ambas mediciones	65
3.3.4	Capacidad de almacenar los datos obtenidos	65
3.3.5	Capacidad de mostrar los datos obtenidos sobre un mapa	66

4 Conclusiones y líneas de trabajo futuras	67
4.1 Conclusiones	67
4.2 Líneas de trabajo futuras	68
A Obtención de clave para API de Google Maps	71
Referencias	78

Índice de figuras

1.1	Espectro de ruido blanco [Wik14c]	10
1.2	Espectro de ruido rosa [Wik14a]	10
1.3	Muestreo y cuantización de una señal en PCM de 4 bits[Wik14b] . . .	11
1.4	Diagrama de la estructura del sistema operativo Android[Goo14]	12
1.5	Notificaciones minimizadas en el área de notificaciones[Goo14] . . .	14
1.6	Diagrama del ciclo de vida de una actividad en Android. [Goo14].	17
1.7	Diagrama del ciclo de vida de un servicio [Goo14]	19
1.8	Mapa de calor mostrando el grado de salinidad de los océanos con el esquema de colores arcoíris.[Wik15]	26
1.9	Niveles RGB e intensidad percibida en el esquema <i>cubehelix</i> [Gre11] .	27
2.1	Borrador para el aspecto de las distintas pantallas de la aplicación .	31
2.2	Entorno de desarrollo integrado Android Studio	33
2.3	Entorno de desarrollo integrado Eclipse	34
2.4	Organización del código fuente.	37
2.5	Editor de archivos de recurso XML con vista previa.	38
2.6	Edición completamente visual de la interfaz gráfica de usuario de la aplicación.	39
2.7	Patrón de cajón de navegación implementado en la aplicación.	41

2.8 Actividad Opciones en un dispositivo con Castellano como lenguaje del sistema.	42
2.9 Actividad Opciones en un dispositivo con cualquier otro lenguaje del sistema.	42
2.10 Arquitectura del audio en Android [Pro14]	44
2.11 Diagrama de interacción de la aplicación y los servicios.	48
3.1 Fragmento Mapa de Calor	56
3.2 Cajón de navegación abierto.	56
3.3 Fragmento Medición en reposo.	57
3.4 Fragmento Medición durante una medición.	57
3.5 Diálogo para la elección del archivo de sesión.	58
3.6 Fragmento Mapa de Calor mostrando datos cargados.	58
3.7 Fragmento Sesiones mostrando la lista de archivos de sesiones anteriores.	59
3.8 Actividad Opciones , donde se configuran distintos aspectos de la aplicación.	59
3.9 Extracto de un archivo CSV producido por la aplicación	60
3.10 Vista del mapa de la calle antes de superponer la medida.	63
3.11 Vista del mapa de la calle con la medida superpuesta.	63
3.12 Vista del mapa del parque antes de superponer la medida.	64
3.13 Vista del mapa del parque con la medida superpuesta.	64
A.1 Consola de APIs de Google sin proyectos.	71
A.2 Proyecto API Project creado por defecto y en blanco.	72
A.3 Servicio <i>Google Maps Android API v2</i> activado.	73
A.4 Sección API Access de la consola de APIs de Google.	74

A.5	Ventana de diálogo donde se introducen los datos necesarios para la clave.	75
A.6	Sección de la consola mostrando los datos de acceso a la API.	76

Índice de fragmentos de código

2.1	Descripción de la interfaz gráfica de usuario del fragmento <code>Medición</code>	40
2.2	Preparación de <code>AudioRecord</code>	45
2.3	Captura de muestras con <code>AudioRecord</code> y cálculo de la p_{rms}	46
2.4	Cálculo del SPL.	46
2.5	Sección de dependencias dentro del archivo <code>build.gradle</code>	47
2.6	Solicitud de conexión a los GMS.	48
2.7	Ubicación de la clave de API de Google Maps en el manifiesto principal.	49
2.8	Inclusión del fragmento <code>MapFragment</code> en un archivo de diseño de interfaz gráfica de usuario.	50
2.9	Inclusión del fragmento <code>MapFragment</code> en un archivo de diseño de interfaz gráfica de usuario.	51
2.10	Inclusión del fragmento <code>XYPlot</code> en un archivo de diseño de interfaz gráfica de usuario.	52
2.11	Configuración y puesta en marcha del componente <code>XYPlot</code> de la librería <code>AndroidPlot</code>	53
2.12	Uso del componente <code>CSVWriter</code> de la librería <code>OpenCSV</code>	54
2.13	Uso del componente <code>CSVReader</code> de la librería <code>OpenCSV</code>	54

Índice de Tablas

1	Usuarios de teléfonos móviles inteligentes [eMa14]	3
2	Cuota de mercado de los distintos sistemas operativos móviles [RL14]	3
3.1	Tabla de comparacion de mediciones	62

Introducción y visión general

Objetivos del Proyecto

El presente proyecto consiste en el desarrollo de una aplicación en Android para su uso en teléfonos móviles inteligentes cuya funcionalidad sea medir el nivel de ruido presente en distintos lugares, y después representarlo sobre un mapa.

Motivación

Es difícil encontrar actividades que no generen cierto nivel sonoro, ya sean naturales o producidas por el ser humano. Estos sonidos pueden ser categorizados en base a muchos parámetros, como lugar, duración, tipo, etc. pero cuando el sonido es molesto y no deseado, se trata de ruido. Este ruido puede llegar a ser perjudicial para la audición, el físico y el psique de seres vivos, convirtiéndose en contaminación acústica.

La contaminación acústica es hoy día un factor clave en el deterioro de la calidad ambiental de un territorio. Según estudios de la Unión Europea “80 millones de personas están expuestas diariamente a niveles de ruido ambiental superiores a 65dBa y otros 170 millones, lo están a niveles entre 55-65dBa” [Uni03]

Normalmente, las mediciones de ruido conllevan el uso de aparatos diversos y muy caros tales como sonómetro, calibrador, pantalla anti-viento, trípode, micró-

fono, preamplificador, etc. Esto supone una importante barrera de accesibilidad a mediciones de ruido. En este proyecto se plantea tomar provecho de la alta penetración en el mercado que tienen los teléfonos inteligentes, y conseguir poner la medición de ruido a un nivel bastante más accesible, a costa de pérdida de precisión. Sin embargo, a cambio se obtienen una mayor facilidad y simplicidad a la hora de realizar mediciones.

Por otra parte, la vasta mayoría de estos dispositivos incorporan de serie un receptor GPS, micrófono y almacenamiento, además de acceso a internet y pantalla táctil. Son características que, de buscarlas todas juntas, sólo se encontrarían en sonómetros de la más alta gama, ya que no son esenciales en una medición profesional. Sin embargo, permiten mejorar la experiencia del usuario y compensar en gran parte la pérdida de precisión.

Estado del arte

El ámbito de la medición y caracterización de ruido está hoy día bastante desarrollado, y no es difícil encontrar buenas herramientas y precisos aparatos que realicen las tareas pertinentes. Sin embargo, el coste de los mismos suele ser bastante elevado, no existiendo soluciones accesibles o de bajo presupuesto. El precio de un sonómetro profesional de clase 1, ronda entre 3 y 5 mil euros, lo que contrasta con el precio de un teléfono inteligente de gama alta, con precios rondando el medio millar de euros.

Por otra parte, la penetración de mercado de los teléfonos inteligentes crece día a día, superando 1.75 mil millones de usuarios para final de 2014[eMa14]. Como se observa en la tabla 1, cerca de dos quintos de la base total de usuarios de teléfonos móviles, utiliza teléfonos móviles inteligentes. Esto supone cerca de un cuarto de la población mundial.

	2012	2013	2014	2015	2016	2017
Usuarios totales (miles de millones)	1.13	1.43	1.75	2.03	2.28	2.50
% de incremento	68.4 %	27.1 %	22.5 %	15.9 %	12.3 %	9.7 %
% de usuarios móviles	27.6 %	33.0 %	38.5 %	42.6 %	46.1 %	48.8 %
% de población mundial	16.0 %	20.2 %	24.4 %	28.0 %	31.2 %	33.8 %

Tabla 1: Usuarios de teléfonos móviles inteligentes [eMa14]

Dentro del segmento de usuarios de teléfonos móviles inteligentes, existen varios sistemas operativos móviles, cada cual provee su propia plataforma de desarrollo, entorno de desarrollo y ecosistema de aplicaciones. A raíz de la información de la tabla 2, se observa que Android es el líder del mercado, con sobrada ventaja.

Período	Android	iOS	Windows Phone	BlackBerry OS	Otros
T3 2014	84.4 %	11.7 %	2.9 %	0.5 %	0.6 %
T3 2013	81.2 %	12.8 %	3.6 %	1.7 %	0.6 %
T3 2012	74.9 %	14.4 %	2.0 %	4.1 %	4.5 %
T3 2011	57.4 %	13.8 %	1.2 %	9.6 %	18.0 %

Tabla 2: Cuota de mercado de los distintos sistemas operativos móviles [RL14]

Herramientas y metodologías utilizadas

Para el desarrollo de la aplicación, la principal herramienta utilizada ha sido el entorno de desarrollo integrado Android Studio. Es un entorno de relativa novedad, ya que aunque su primera versión estable fue liberada en Diciembre de 2014, su desarrollo se remonta a Mayo de 2013. Sin embargo, no ha sido desarrollado desde cero,

sino que está basado en el popular entorno de desarrollo integrado IntelliJ IDEA, de JetBrains. Android Studio está disponible de manera gratuita para las plataformas mayoritarias Windows, Mac OS X y Linux. Al ser un entorno de desarrollo enfocado completamente al ecosistema Android, provee múltiples ayudas y mejoras sobre un entorno de desarrollo integrado más genérico como puede ser Eclipse. Características que han hecho decantarse por Android Studio y no Eclipse se analizan en el apartado 2.2.1.

Además de el código generado para el proyecto, han sido utilizadas dos librerías Java. La librería OpenCSV, genérica de Java, provee una implementación sólida del formato de archivo CSV y nos abstrae de él, y la librería AndroidPlot, específica para Android, nos permite hacer representaciones gráficas de conjuntos de datos sin tener que entrar en pormenores de la generación de dichas imágenes.

Ha sido necesario también un teléfono inteligente con Android, tanto durante la fase de desarrollo como durante las posteriores de utilización y realización de pruebas.

Para la fase de pruebas, se ha hecho uso de equipo básico de laboratorio de acústica, incluyendo monitores de estudio y sonómetro.

Estructura del documento

Introducción teórica

En el capítulo 1 se hará una introducción a los conceptos teóricos requeridos para la comprensión del trabajo realizado. Se explicarán los parámetros acústicos tratados a lo largo de la memoria. A su vez, también se explicarán conceptos relativos a la programación en Android, tratados a lo largo del proyecto.

Implementación

En el capítulo 2 se explicará cómo se ha llevado a cabo toda la implementación de la aplicación, empezando por un análisis de los requisitos necesarios para la realización de esta, y siguiendo con los aspectos relevantes a la programación de la aplicación: métodos, interfaz gráfica, algoritmos, etc.

Plan de pruebas y verificación

En el capítulo 3 se explicarán los procedimientos llevados a cabo para cerciorar el correcto funcionamiento de la aplicación.

Conclusiones y trabajo futuro

En este apartado se pretende representar las conclusiones obtenidas durante la realización de este proyecto, así como plasmar posibles ideas que se consideran interesantes de cara a una futura continuidad en el desarrollo de la aplicación.

Capítulo 1

Introducción teórica

1.1. Acústica

1.1.1. Nivel de presión sonora

Definimos como presión acústica o presión sonora el cambio local de presión respecto a la presión atmosférica ambiente (media o en equilibrio) causada por una onda sonora. En el SI, la presión atmosférica (y por ende la acústica) se mide en pascales (Pa), que es igual a una fuerza de un newton (1 N) actuando sobre una superficie de un metro cuadrado ($1m^2$).

Sin embargo, para mediciones se utiliza el Sound Pressure Level, o nivel de presión sonora (SPL) definido en la ecuación 1.1, una medición logarítmica de la presión sonora efectiva relativa a un nivel de referencia. Por definición:

$$L_p = 10 \log_{10} \left(\frac{p_{\text{rms}}^2}{p_0^2} \right) = 20 \log_{10} \left(\frac{p_{\text{rms}}}{p_0} \right) \text{ dB(SPL)} \quad (1.1)$$

Donde p_{rms} es el valor cuadrático medio de la presión sonora, que es calculado como se muestra en la ecuación 1.2, medido en Pa, y p_0 es la presión sonora de referencia, también medida en Pa.

$$p_{\text{rms}} = \left[\frac{1}{T} \int_0^T p^2(t) dt \right]^{\frac{1}{2}} \quad (1.2)$$

El valor de p_0 más comúnmente utilizado es $p_0 = 20\mu\text{Pa}(RMS)$, que representa el umbral mínimo de audición humana, y será el utilizado en el presente proyecto.

1.1.2. Potencia acústica y densidad espectral de potencia

Es potencia acústica, o potencia sonora, la medición de la cantidad de energía sonora por unidad de tiempo emitida por una fuente determinada, definida en la ecuación 1.3. Dicha potencia en el Sistema Internacional de medidas (SI) se mide en Watios (W). Dada una fuente sonora, la potencia sonora de dicha fuente es independiente del entorno y de la distancia. La potencia sonora es la potencia total producida por la fuente en todas direcciones. Esto contrasta con la presión sonora, que sí es dependiente de la distancia, ya que es una medición en un punto concreto.

$$P_a = f \cdot v = A \cdot p \cdot u \cdot v \quad (1.3)$$

Es posible también definir el Nivel de potencia acústica (SWL) 1.4 de manera análoga al SPL, utilizando en este caso una $P_0 = 1\text{pW}$.

$$L_W = 10 \log_{10} \left(\frac{P}{P_0} \right) \text{ dB(SWL)} \quad (1.4)$$

Espectro de frecuencia

El espectro de frecuencia de una señal en el dominio del tiempo es una representación de dicha señal en el dominio de la frecuencia. El espectro de frecuencia puede ser generado mediante una transformada de Fourier de dicha señal, y los valores resultantes son normalmente amplitud y fase representados en función de la frecuencia.

1.1.3. Ruido

De forma general, es posible definir como ruido cualquier sonido no deseado que interfiera en la recepción de un sonido. El ruido acústico es aquel ruido producido por la mezcla de ondas sonoras de distintas frecuencias y distintas amplitudes. La mezcla se produce a diferentes niveles ya que se conjugan tanto las frecuencias fundamentales como los armónicos que las acompañan.

De las múltiples maneras de clasificar el ruido acústico, son de interés en este proyecto dos:

Ruido en función de la intensidad y el periodo

- Ruido fluctuante

Ruido fluctuante es aquel cuya intensidad no es constante sino que fluctúa a lo largo del tiempo, ya sea de forma periódica o aleatoria.

- Ruido impulsivo

Ruido impulsivo es aquel cuya intensidad aumenta bruscamente durante un impulso. En comparación con el tiempo que transcurre entre un impulso y otro, la duración del impulso es breve.

Ruido en función de la frecuencia

Existen múltiples modelos de ruido en función de su frecuencia, y la mayoría se implementan en generadores de ruido para ser utilizados en mediciones acústicas.

Los principales son:

- Ruido blanco

Quizás el modelo de ruido más conocido, ruido blanco es aquel con la misma cantidad de potencia sonora en cualquier banda de un ancho de banda

determinado. Es llamado así por analogía a la luz blanca. Cuando se realiza el diagrama de espectro de un ruido blanco, se observa un espectro plano en frecuencia. Para una señal de audio, es suficiente que esta condición se cumpla en el rango audible, es decir de 20 Hz a 20.000 Hz.

- Ruido rosa

El ruido rosa es un ruido con una densidad espectral inversamente proporcional a la frecuencia, de manera que la potencia de ruido de cada octava sea idéntica.

Al visualizar su espectro de frecuencia, se aprecia claramente la caída.

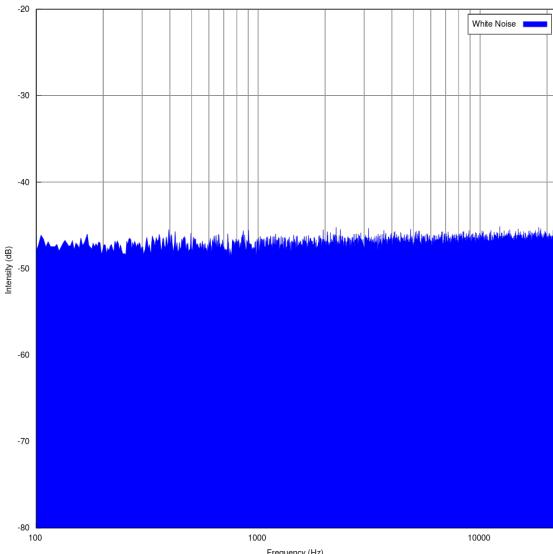


Figura 1.1: Espectro de ruido blanco
[Wik14c]

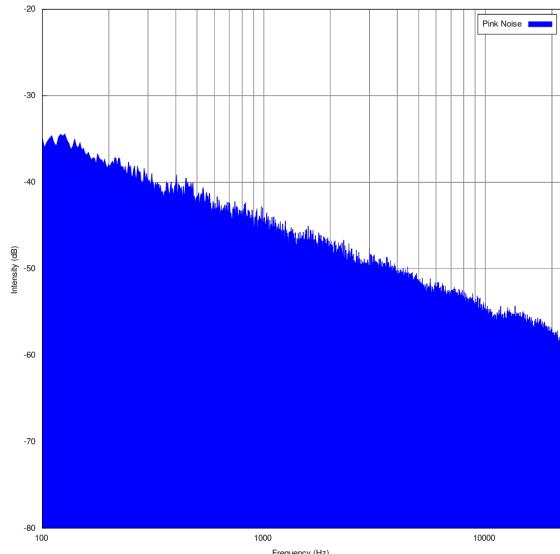


Figura 1.2: Espectro de ruido rosa
[Wik14a]

1.1.4. Modulación por impulsos codificados

La Pulse Code Modulation, o modulación por impulsos codificados (PCM), es un método utilizado para representar digitalmente señales analógicas muestradas. Es

la forma básica de audio digital en ordenadores, discos compactos, telefonía digital y la gran mayoría de aplicaciones de audio digital. En un flujo de audio PCM, la amplitud de la señal analógica se muestrea regularmente en intervalos uniformes, y cada muestra se cuantifica al valor más próximo dentro de un rango de posibles valores digitales predefinidos.

En la figura 1.3, se observa cómo se realiza el proceso de muestreo y cuantización de una forma de onda senoidal durante un período completo. Las líneas verticales a intervalos regulares representan los instantes muestreados, uno cada período de muestreo. Las líneas horizontales muestran los distintos niveles de cuantización establecidos para este muestreo. En este ejemplo, se realiza una cuantización en 4 bits, lo que nos da $2^4 = 16$ niveles diferentes disponibles, del 0 al 15. Los valores de la señal en cada instante muestreado se redondean al valor cuantizado más próximo.

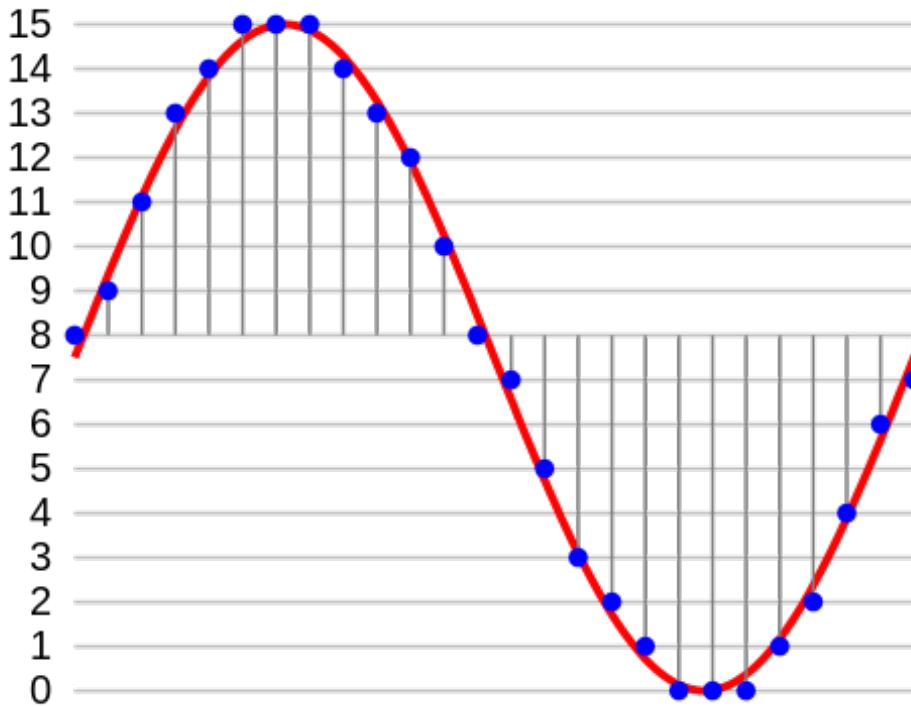


Figura 1.3: Muestreo y cuantización de una señal en PCM de 4 bits[Wik14b]

Este es el formato en el que el sistema operativo del teléfono inteligente nos proveerá las muestras de audio, y sobre el que trabajaremos.

1.2. Android

Android es un sistema operativo que en sus inicios fue concebido para dispositivos móviles con pantalla táctil. Sin embargo, ha evolucionado en una plataforma que actualmente también engloba relojes inteligentes, televisores, automóviles e incluso electrodomésticos.

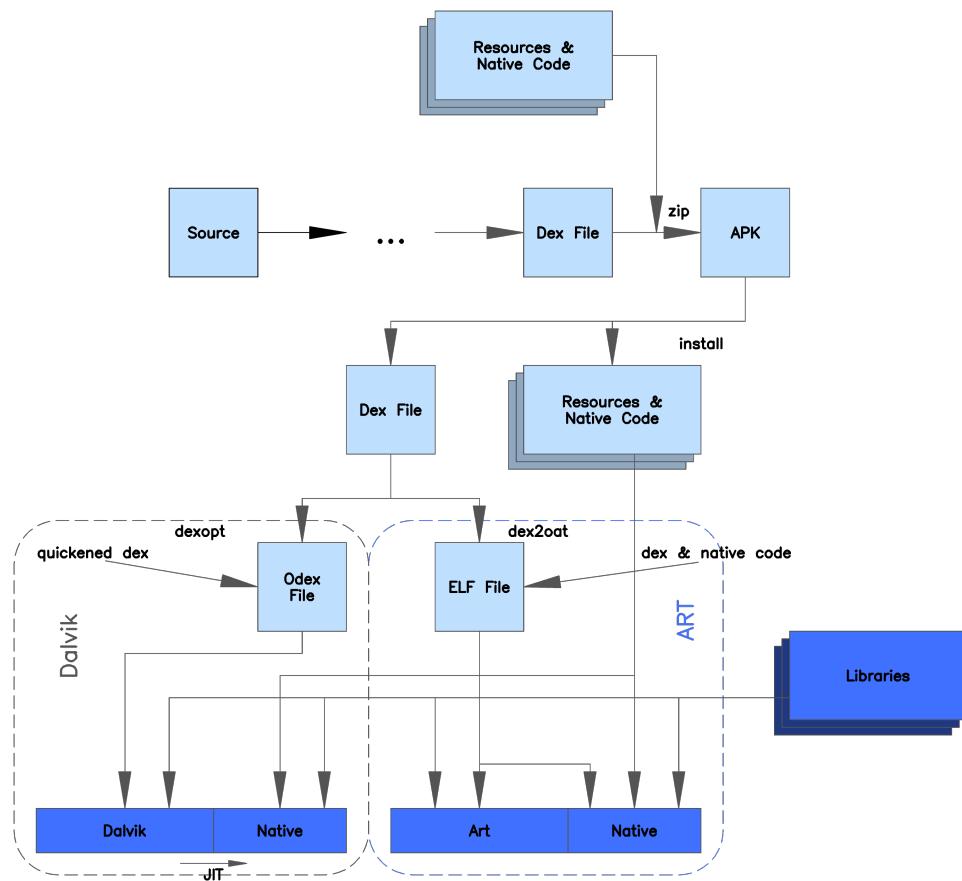


Figura 1.4: Diagrama de la estructura del sistema operativo Android[Goo14]

Está basado en el núcleo de Linux. Sobre este, corre el entorno de ejecución propio de Android, que provee una máquina virtual (ya sea Dalvik o ART). Este a su vez ejecuta el código de las aplicaciones, escritas mayoritariamente en Java, aunque se permiten extensiones en C/C++. Las aplicaciones son precompiladas a un código intermedio denominado **Dex** y este es empaquetado y comprimido en el archivo estándar de aplicación Android, el **APK**.

Cada aplicación instalada en el dispositivo, es optimizada según los recursos y tipo de máquina virtual disponibles en el dispositivo en concreto. Al ser iniciada, es ejecutada dentro de un *cajón de arena*, técnica que aisla los procesos de cada aplicación. Esto resulta en una mayor robustez y seguridad del sistema, ya que el fallo de una aplicación no afecta a los recursos de las demás. Tampoco podría una aplicación maligna, si se diera el caso, modificar o apropiarse de los recursos de otras. En la figura 1.4 se observa la relación entre todos los elementos anteriormente descritos.

1.2.1. Interfaz

La interfaz de usuario por defecto de Android está basada en una manipulación directa, usando entrada táctil con gestos que vagamente corresponden a acciones físicas reales, tales como deslizar, golpear, pellizcar... Para manipular objetos en pantalla. Además, la mayoría de dispositivos dispone de un teclado virtual, manipulado de la misma manera.

La respuesta del sistema a la entrada del usuario está diseñada para ser inmediata y dar una sensación de fluidez, utilizando las capacidades de vibración presentes en la mayoría de los dispositivos para proveer una respuesta háptica (no visual, no auditiva). Adicionalmente, algunas aplicaciones utilizan la información proveída por sensores tales como acelerómetros, giroscopios y sensores de proximidad para responder a interacciones adicionales, como por ejemplo ajustar la orientación de la

pantalla cuando el dispositivo se encuentra apaisado o controlar alguna parte de la aplicación basándose en el azimut relativo del dispositivo.

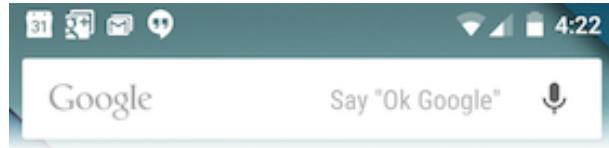


Figura 1.5: Notificaciones minimizadas en el área de notificaciones[Goo14]

Una parte importante de la interfaz general de Android son las notificaciones, presentes en la barra de estado. Una notificación es un mensaje que una aplicación muestra al usuario fuera de los límites habituales de la interfaz de usuario de la aplicación. Las aplicaciones solicitan al sistema que emita una notificación por ellas, proveyendo el contenido, y el sistema operativo es quien maneja el resto del ciclo de vida de la notificación. En la figura 1.5 se pueden observar cuatro notificaciones minimizadas en el área de notificaciones

1.2.2. Componentes de una aplicación

Contexto

Uno de los conceptos más importantes cuando se utiliza la plataforma Android es el contexto, `Context`. La clase `Context` en sí misma no es más que una interfaz a información global acerca del entorno de una aplicación, y como tal es abstracta. Sin embargo, es importante ser consciente de qué elementos representan un contexto válido y qué elementos no. Un contexto permite acceso a recursos y clases específicos de la aplicación, y llamadas al sistema para operaciones a nivel de aplicación. Por ejemplo lanzar actividades, emitir mensajes de difusión o recibirlós.

Actividades

En android, una actividad (**Activity**) representa una única cosa concreta que el usuario puede realizar en la aplicación. La mayoría de las actividades interaccionan con el usuario, por tanto la clase **Activity** se encarga de crear una ventana donde se puedan insertar los componentes de la interfaz de usuario. Aunque las actividades suelen ser vistas por el usuario como ventanas a pantalla completa, también pueden ser usadas en otras múltiples maneras, ya sea como ventanas flotantes, o incrustadas dentro de otra actividad (mediante un **ActivityGroup**)

Ciclo de vida de una actividad

Las distintas actividades de las distintas aplicaciones instaladas en el dispositivo Android son gestionadas en forma de una *pila de actividades*.

Cuando el sistema Android se inicia, se presenta al usuario una pantalla principal, desde donde puede lanzar varias acciones y aplicaciones. A partir de ahí, cuando una nueva actividad es empezada, se emplaza arriba de la pila y se convierte en la actividad en ejecución. Las actividades previas, si las hubiera, siempre permanecen por debajo en la pila, y no se traerán al frente hasta que la nueva actividad finalice.

Una actividad tiene cuatro estados básicos:

Activa

Una actividad está *activa* cuando está presente en primer plano en la pantalla, es decir, arriba de la pila. También se puede decir que la actividad está *En ejecución*.

Pausada

Si una actividad ha perdido el foco (ha dejado de estar en primer plano) pero todavía es visible, es decir, si una nueva actividad que no ocupa la totalidad de la pantalla o es transparente obtiene el primer plano, se encuentra *pausada*.

Una actividad pausada se conserva completamente íntegra (mantiene todos los estados y se mantiene suscrita al gestor de ventanas) pero puede ser matada por el sistema en condiciones extremas de baja memoria disponible.

Parada

Si una actividad se encuentra oculta por completo, el sistema la deja *parada*. Mantiene estados e información de los miembros, pero sin embargo al no ser visible por el usuario es más probable que el sistema se deshaga de ella para liberar recursos cuando hagan falta.

Muerta

Cuando el sistema decide mover la actividad fuera de memoria, puede o bien finalizarla o matar el proceso. Cuando sea mostrada de nuevo al usuario, debe ser completamente reiniciada y restaurada a su estado previo.

Los cuatro estados y las acciones que llevan a una aplicación de un estado a otro, junto con los métodos de la actividad que son invocados en cada etapa, están representados en la figura 1.6.

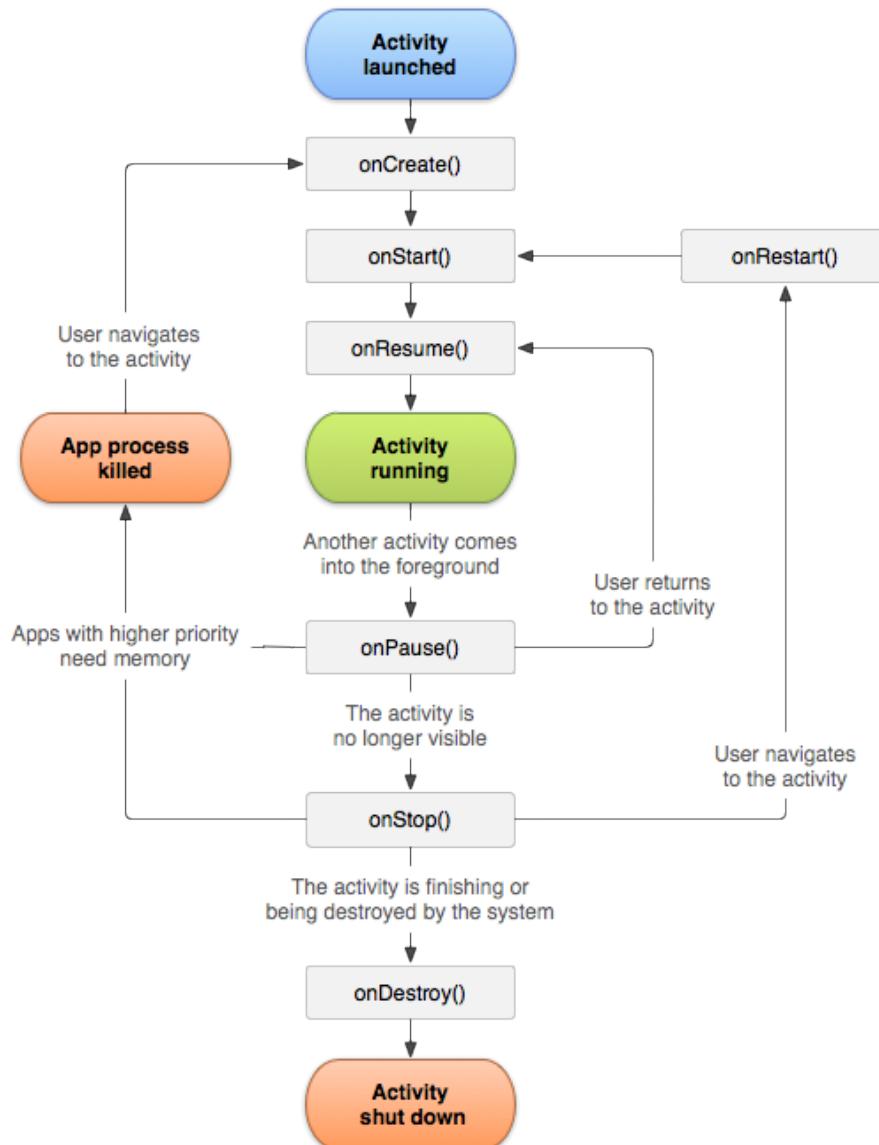


Figura 1.6: Diagrama del ciclo de vida de una actividad en Android. [Goo14].

Servicios

Un Servicio (**Service**) es un componente de la aplicación que puede realizar tareas de larga duración en segundo plano y no provee ninguna interfaz de usuario. Otro componente de la aplicación puede iniciar un servicio y este continuará en marcha en segundo plano, incluso si el usuario cambia a otra aplicación distinta. Además, un componente puede adherirse (**bind**) a un servicio para interactuar con él, e incluso realizar comunicación inter-procesos (IPC por sus siglas en inglés, Inter-Process Communication). Por ejemplo, un servicio puede manejar llamadas de red, reproducir música, realizar operaciones en el sistema de archivos, todo en segundo plano.

Un servicio puede tomar dos estados:

Started

Un servicio está en estado **Started** (empezado) cuando un componente de la aplicación, por ejemplo una actividad, lo empieza llamando al método **startService()**. Una vez empezado de esta manera, un servicio puede continuar en segundo plano de manera indefinida, incluso si el componente que lo empezó ha sido destruido. Normalmente, suelen ser servicios que realizan una única operación y no devuelven ningún resultado. Por ejemplo, puede descargar o subir un archivo en la red. Cuando la operación ha sido completada, el servicio debe pararse a si mismo.

Bound

Un servicio está en estado **Bound** (adherido) cuando un componente de la aplicación se adhiere a él llamando al método **bindService()**. Un servicio adherido ofrece una interfaz servidor-cliente que permite a los componentes interaccionar con el servicio, mandar peticiones, obtener resultados, e incluso hacerlo entre distintos procesos mediante comunicación inter-proceso (IPC).

Un servicio adherido solamente es activo durante el tiempo que otro componente esté adherido a él. Varios componentes pueden estar adheridos en un momento dado al servicio, pero cuando todos se desadhieren del servicio, el servicio es destruido.

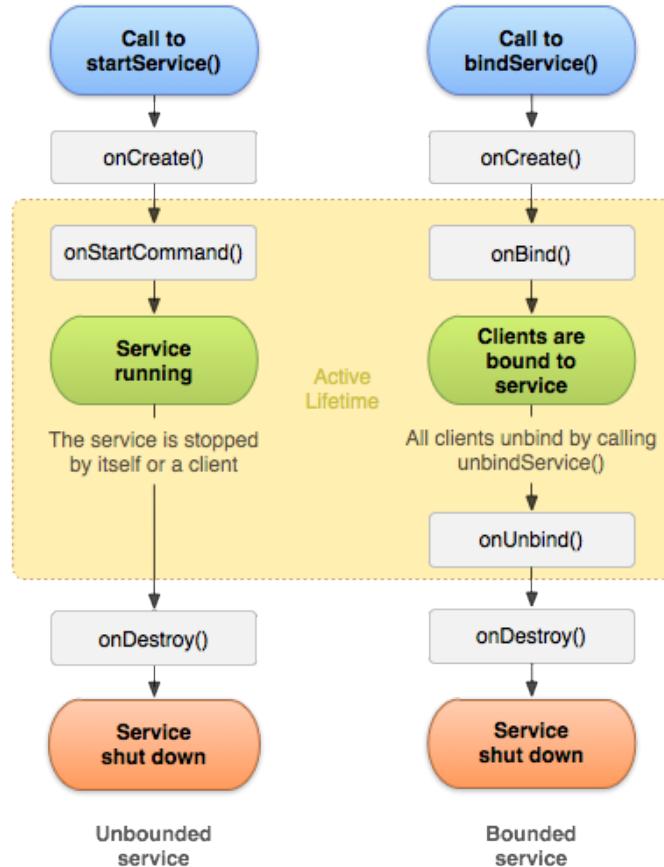


Figura 1.7: Diagrama del ciclo de vida de un servicio [Goo14]

Es necesario tener cuidado, dado que un servicio corre en el hilo principal de ejecución del proceso que lo llama, a no ser que se especifique lo contrario. Esto implica que, si el servicio va a realizar alguna tarea intensiva en CPU, o alguna operación bloqueante, se debe de crear un nuevo hilo de ejecución dentro del servicio.

para ese propósito. De no hacerlo, se corre el riesgo de que la aplicación deje de responder y sea matada por el sistema operativo.

Los dos tipos de ciclo de vida de un servicio, sus etapas y los métodos invocados en el transcurso de ellas están representados en la figura 1.7.

1.2.3. Vistas

La interfaz gráfica de usuario en una aplicación Android está construida usando una jerarquía de vistas, objetos de la clase `View`, y grupos de vistas, objetos de la clase `ViewGroup`. Los objetos `View` suelen ser artílugos (widgets) de la interfaz de usuario, tales como botones o campos de texto, y los objetos `ViewGroup` son contenedores invisibles que definen cómo se posicionan las vistas que dependen de ellos, por ejemplo dispuestas en forma de rejilla o lista vertical.

Android provee un vocabulario XML que se corresponde con las subclases de `View` y `ViewGroup`, y permite definir la interfaz de usuario en XML usando una jerarquía de elementos de interfaz de usuario.

1.2.4. Procesos e hilos de ejecución

En sistemas operativos, son básicos los conceptos de proceso e hilo de ejecución. En Android, el sistema operativo comienza un nuevo proceso Linux por cada primer componente de cada aplicación, con un único hilo de ejecución. Por defecto, todos los componentes de la misma aplicación corren en los mismos proceso e hilo de ejecución, el hilo principal de ejecución (*main thread* en inglés).

En caso de que un componente de una aplicación sea inicializado, y ya exista un proceso para dicha aplicación en el sistema que otro componente de la misma aplicación ha inicializado, el nuevo componente se inicializa dentro del proceso original y usa el mismo hilo de ejecución. Sin embargo, se puede configurar una aplicación

de manera que diferentes componentes corran en procesos separados, y siempre se pueden crear hilos de ejecución adicionales para cualquier proceso.

Procesos

Como ya ha sido expuesto anteriormente, por defecto todos los componentes de la misma aplicación corren en el mismo proceso, y la mayoría de las aplicaciones no deberían de cambiarlo. Sin embargo, es posible controlar qué proceso pertenece a qué componente de ser necesario.

El sistema operativo puede decidir apagar un proceso, cuando la cantidad de memoria disponible sea baja y haya requerimiento de ella por otro proceso que sirva de manera más inmediata al usuario. En este caso, los componentes dentro de dicho proceso que es apagado, son destruidos. Cuando estos componentes sean necesarios de nuevo, un nuevo proceso será comenzado por el sistema operativo para ello.

Hilos de ejecución

Previamente se ha mencionado que todos los componentes de la misma aplicación corren en el mismo hilo de ejecución, el hilo principal de ejecución o *main thread*. Este hilo es de suma importancia, dado que carga con la responsabilidad de despachar los eventos al widget de la interfaz de usuario que sea pertinente, incluyendo los eventos de dibujado en pantalla. Es también el hilo de ejecución en el que la aplicación interactúa con los componentes básicos de interfaz de usuario de Android, también conocidos como *Android UI toolkit* (ubicados dentro de los paquetes java `android.widget` y `android.view`). Por todo esto, no es extraño encontrar denominado este hilo de ejecución como el hilo de la interfaz de usuario, o *UI Thread*.

Dado que todos los componentes que corren en el mismo proceso son instanciados en el hilo de ejecución principal, las llamadas del sistema operativo a cada

componente son despachadas desde dicho hilo. En consecuencia, todos los métodos que responden a retrollamadas del sistema (*system callbacks*), como por ejemplo para indicar que una tecla ha sido pulsada, siempre corren en el hilo principal de ejecución del proceso.

Cuando el usuario toca un botón en la pantalla, el hilo principal de la aplicación despacha el evento de toque al widget pertinente, que reacciona cambiando su estado a presionado y manda una petición de invalidación a la cola de eventos. El hilo de ejecución principal entonces desencola la petición y notifica al widget que debe redibujarse.

Cuando una aplicación realiza trabajo intensivo en respuesta a una interacción del usuario, el modelo de hilo de ejecución único puede resultar en una falta de rendimiento. Concretamente, de suceder todo el procesamiento en el hilo principal de ejecución e iniciar tareas de larga ejecución tales como acceso a red o consultas a bases de datos, resultará en un bloqueo completo de la interfaz de usuario y su correspondiente hilo principal. Cuando el hilo de ejecución está bloqueado, no se pueden despachar eventos, eventos de dibujado en pantalla incluidos. Desde el punto de vista del usuario, esto se traduce en una aplicación que parece colgarse. En peores casos, en los que el hilo principal de ejecución está bloqueado por más de unos cuantos segundos (cinco segundos en la actualidad) el sistema operativo entrará en acción y mostrará una pantalla explicando que la aplicación ha dejado de responder, y matará la aplicación bloqueada.

Para evitar dicha penalización en el rendimiento, deben usarse hilos de ejecución alternativos para toda tarea que bloquee la ejecución o sea de alta carga procedural.

1.2.5. Manejo de archivos

Para guardar la información recabada en las sesiones de medición realizadas en la aplicación, se usará almacenamiento en archivos de texto plano. Concretamente, se

utilizará el formato Comma Separated Values, o valores separados por coma (CSV). En este formato, como su nombre indica, cada valor de un registro está separado del contiguo por dicho signo de puntuación. Cada registro está separado de otro por un salto de línea.

La simplicidad que brinda este formato, además de la legibilidad de los archivos al estar estos en texto plano, se presumen adecuados para este proyecto.

Con el propósito de no invertir esfuerzo en desarrollar, probar y verificar una interfaz Java para archivos CSV, se ha utilizado la librería OpenCSV para los propósitos de abstracción del formato.

1.2.6. Ubicación

GPS

El Global Positioning System, o sistema de posicionamiento global (GPS) es un sistema de navegación por satélite que provee información sobre la posición y el reloj del dispositivo de recepción. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos. Está constituido por 24 satélites en órbita geosíncrona y se basa en un sistema de triangulación para determinar posiciones con precisión de metros.

El funcionamiento del GPS necesita la recepción de la señal de al menos cuatro de los 24 satélites en órbita. En base a dichas señales, que incluyen la identificación del satélite emisor y la hora de reloj de emisión, el aparato receptor sincroniza su reloj interno y calcula el tiempo que tardan en llegar las señales al equipo. Con dicha información, el dispositivo es capaz de conocer su distancia a cada uno de los satélites cuya señal es recibida, y por ende la posición absoluta del punto de medición.

Servicios Móviles de Google

Conocer la localización GPS del dispositivo móvil es crítico en el funcionamiento de la aplicación. Es posible en un dispositivo Android utilizar el receptor GPS a bajo nivel y recibir actualizaciones de posición en formato NMEA. Sin embargo, no se aprovecharían cantidad sustancial de optimizaciones que Google ofrece como parte de Google Mobile Services, o servicios móviles de Google (GMS) tales como tiempo de respuesta mejorado, localización mixta y consumo de batería mejorado.

GMS es un servicio en segundo plano propietario de Google presente en todos los dispositivos Android que satisfagan las condiciones de entrada de la empresa. En este proyecto, la aplicación se conectará a dicho servicio para requerir actualizaciones de posición junto con información extra, por ejemplo la precisión de dicha medida.

1.2.7. Mapas

Se entiende mapa como la representación gráfica y métrica de una porción de territorio, generalmente sobre una superficie bidimensional, con una expresión clara sin sacrificar exactitud. Como parte de sus GMS, Google pone a disposición de aplicaciones en la plataforma Android su servicio de mapas Google Maps, accesibles a través de la Application Programming Interface, o interfaz de programación de aplicaciones (API) Google Maps Android. Esta API permite no solo mostrar un mapa con datos provenientes de los servidores de Google, sino que además permite añadir información adicional, tales como iconos, marcadores, polígonos o imágenes superpuestas. Para realizar el mapa de calor, en este proyecto es de interés esa última capacidad.

1.3. Otros

1.3.1. Control automático de ganancia (CAG)

Un Control Automático de Ganancia (CAG) es un circuito con realimentación cuyo propósito es que la amplitud de la señal a su salida se mantenga controlada a pesar de posibles variaciones de amplitud en la señal de entrada. Dependiendo del diseño del circuito, el nivel medio o de pico es utilizado para ajustar dinámicamente la ganancia del circuito a un valor adecuado.

El CAG tiene usos en varios campos. Fue el componente que hizo a los receptores de radio AM dejar de ser lineales [LS53, cap. 27 sec. 3], ya que sin un CAG, una radio AM tendría una relación completamente lineal entre la señal que la portadora introduce y la señal a la salida. Sin embargo, dado que la fuerza de la señal recibida varía considerablemente, la adición de un CAG permite mantener una salida del receptor menos fluctuante. En los sistemas de radar, se utiliza como método para subsanar ecos indeseados, y en los sistemas de cinta de audio de baja gama se utiliza para corregir el nivel de señal en las grabaciones.

En los teléfonos móviles, se aplica a la señal de micrófono con intención de proporcionar una mayor inteligibilidad de la palabra durante las conferencias. Esto se logra evitando que variaciones en la distancia micrófono-fuente afecten dramáticamente a la intensidad del sonido.

La vasta mayoría de los fabricantes de teléfonos móviles inteligentes, incorporan semejante función en alguna fase del procesado de la señal de entrada de micrófono, ya sea mediante circuitos analógicos o procesado digital de señal. Es importante tener esto en cuenta, ya que puede ser causa de una aparente compresión de la señal, si se diera el caso.

1.3.2. Mapa de calor

Un mapa de calor es una representación gráfica de datos donde los valores en un plano se representan como colores. Es común utilizarlos superpuestos a otro gráfico o mapa para representar intensidad de una magnitud sobre este. Un ejemplo de mapa de calor se puede apreciar en la figura 1.8.

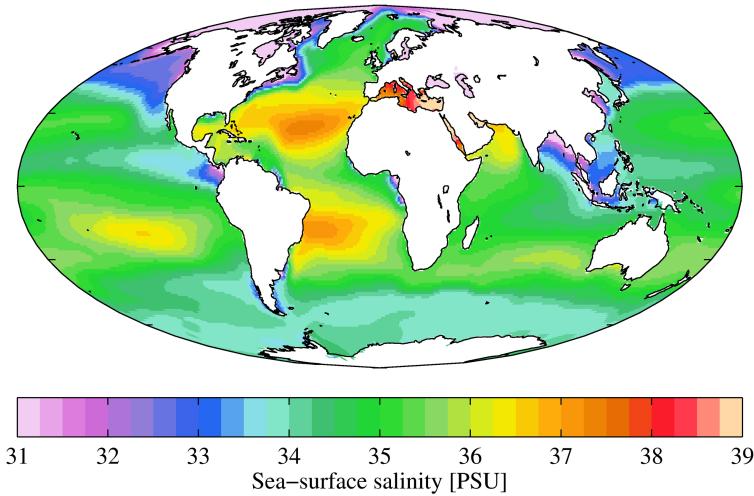


Figura 1.8: Mapa de calor mostrando el grado de salinidad de los océanos con el esquema de colores arcoíris.[Wik15]

Existen muchos esquemas de colores que pueden ser utilizados para ilustrar un mapa de calor, cada uno con ventajas y desventajas en su percepción. Esquemas de colores abarcando los colores del arcoíris son muy comunes, ya que el ojo humano percibe una mayor cantidad de colores distintos que tonos de gris, y por tanto incrementa el nivel de detalle percibido en la imagen.

Sin embargo, en la comunidad científica es una práctica desaconsejada, dado que la escala de colores del arcoíris carece de una percepción natural de orden, pierden validez al ser transformados a escala de grises (especialmente al hacer aparecer las regiones amarillas y cyan más importantes de lo que deberían) y, debido a la respuesta no uniforme a los distintos colores primarios del ojo humano, los cambios de

color llevan el ojo humano a percibir gradientes que en realidad no existen.

El esquema de colores *cubehelix*, descrito por primera vez en [Gre11], soluciona dicho problema creando un esquema de colores en el que la intensidad de color percibida sea lineal y creciente, y al pasarlo a escala de grises se mantenga el orden correcto. El nombre de este esquema de colores proviene de la hélice que se forma alrededor de la diagonal del cubo de colores al representarlo en dicha forma tridimensional.

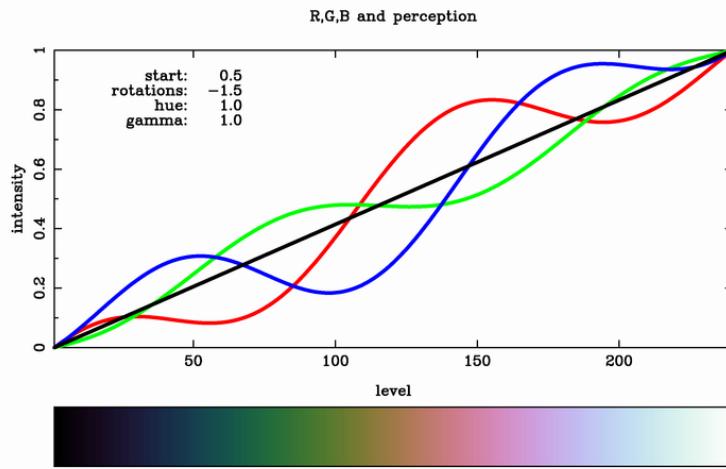


Figura 1.9: Niveles RGB e intensidad percibida en el esquema *cubehelix*[Gre11]

En este proyecto, se utilizarán mapas de calor para representar el nivel de presión sonora registrado en puntos geográficos, de manera que la visualización sea clara y concisa. Tras una primera aproximación con el esquema de colores por defecto, que va del verde al rojo pasando por el amarillo, se ha implementado también el esquema *cubehelix*, dando la posibilidad al usuario de elegir entre ambos.

Capítulo 2

Implementación

2.1. Análisis de requisitos

Para el desarrollo de este proyecto se han tenido en cuenta una serie de requisitos previos mínimos, necesarios para una correcta implementación del mismo. Los requisitos funcionales de la aplicación fueron acordados en los siguientes:

- Capacidad de medir la magnitud del ruido ambiente

El usuario debe de ser capaz de realizar mediciones de magnitud del ruido ambiente bajo demanda, así como pararlas a discrección.

- Capacidad de determinar la posición del dispositivo

La aplicación debe de ser capaz de determinar la posición geográfica del dispositivo con una precisión suficiente para ser utilizada en el posterior procesado de los datos.

- Capacidad de asociar ambas mediciones

Cada medición de magnitud del ruido ambiente debe de ir acompañada por la posición geográfica en la cual se realizó la misma.

- Capacidad de almacenar los datos obtenidos

Los datos recabados en cada sesión de medición deben de ser almacenados de alguna manera para la posterior disponibilidad de estos a discrección del usuario.

- Capacidad de mostrar los datos obtenidos sobre un mapa

Los datos obtenidos deberán de poder ser representados sobre un mapa geográfico de una manera clara y concisa.

Adicionalmente, los siguientes requisitos no funcionales fueron considerados:

- Extensibilidad futura La aplicación debe de ser fácilmente extensible en un futuro. Para ello se tendrá en mente la modularidad y el código autoexplicativo durante el desarrollo de la aplicación
- Tolerancia a fallos La aplicación debe de tolerar pequeños fallos sin que estos entorpezcan en medida alguna la posible medición en curso.
- Rendimiento La aplicación no debe de ser una carga importante para el rendimiento del sistema, ya que esto podría resultar en mediciones imprecisas o pérdida de muestras por incapacidad del sistema de copar con la carga.
- Usabilidad La aplicación está enfocada hacia la accesibilidad de las mediciones de ruido ambiente hacia un público más amplio, por tanto no deberá de ser de difícil manejo.

2.1.1. Diseño de la interfaz gráfica

Como paso previo a la implementación en código, se han diseñado unos borradores de las distintas pantallas de las que constará la aplicación. La aplicación se ha

dividido en cuatro pantallas básicas: mapa, grabar, sesiones y opciones, las cuales se pueden observar en la figura 2.1.

Mapa

La pantalla inicial, que muestra un mapa geográfico y nos permite superponer los datos previamente recabados

Grabar

Aquí se interactúa con los sensores del dispositivo, y se comandan las acciones de grabación y guardado de archivos.

Sesiones

Se presentan todos los datos recabados en pasadas grabaciones.

Opciones

Permite ajustar los parámetros que en la aplicación hayan sido decididos como configurables.



Figura 2.1: Borrador para el aspecto de las distintas pantallas de la aplicación

2.1.2. Dispositivo Android

El desarrollo de este proyecto ha sido realizado y testeado en dos dispositivos. El primero, modelo HTC Desire HD poseedor de la versión 4.2.2 de Android; el segundo un Google Nexus 5, bajo la versión Android 5.0, lo cual garantiza que la aplicación conserva toda su funcionalidad en los modelos más modernos, tanto en software como en hardware.

No obstante, también se ha comprobado su funcionalidad en una rango de dispositivos más amplio, tales como: Samsung Galaxy S3, Samsung Galaxy Nexus, Sony Xperia P. No se ha percibido en ningún momento pérdida alguna de funcionalidad.

La versión mínima de Android requerida para el correcto funcionamiento de esta aplicación, es el nivel de API 15, correspondiente a Android 4.0.3. Es posible hacerla funcionar en niveles más bajos (antiguos), pero requiere esfuerzo adicional, tal y como se esboza en el apartado de Conclusiones y Trabajo Futuro.

2.1.3. Micrófono externo

Los micrófonos empotrados en los teléfonos móviles tienen un objetivo muy claro: la transmisión de voz vía redes celulares. Por tanto, es de esperar que muestren cierto sesgo en diseño cuando se les intenta usar para otro propósito.

Un perfecto ejemplo de ello es el CAG. El CAG es de verdadera utilidad para mejorar los niveles sonoros realizando una llamada, pero entra en conflicto con el propósito de este proyecto, ya que necesita de una señal sin pre-procesamiento alguno. El efecto de compresión que realiza el CAG, proporciona unas medidas sin sentido alguno.

Adicionalmente, el tamaño y posición empotrada del mismo, los hace muy sensibles a interferencias indeseadas tales como la vibración del propio teléfono.

Uno de los modelos utilizados en el desarrollo, el Google Nexus 5, permite la

desactivación del CAG, por tanto se obtienen medidas aceptables. No obstante, para mejores resultados, se debe de usar un micrófono externo sin CAG.

2.1.4. Calibración

La aplicación debe de ser capaz de adaptarse a distintos dispositivos, no basta con trabajar correctamente únicamente en el dispositivo en el que fue desarrollada. Para ello, se habilitará en las opciones de la aplicación un apartado en el que se pueda introducir un valor de compensación en la medida. Esto ayuda a subsanar previsibles diferencias entre los sistemas de captación sonora de distintos dispositivos móviles.

2.2. Aplicación Android

2.2.1. Entorno de desarrollo integrado

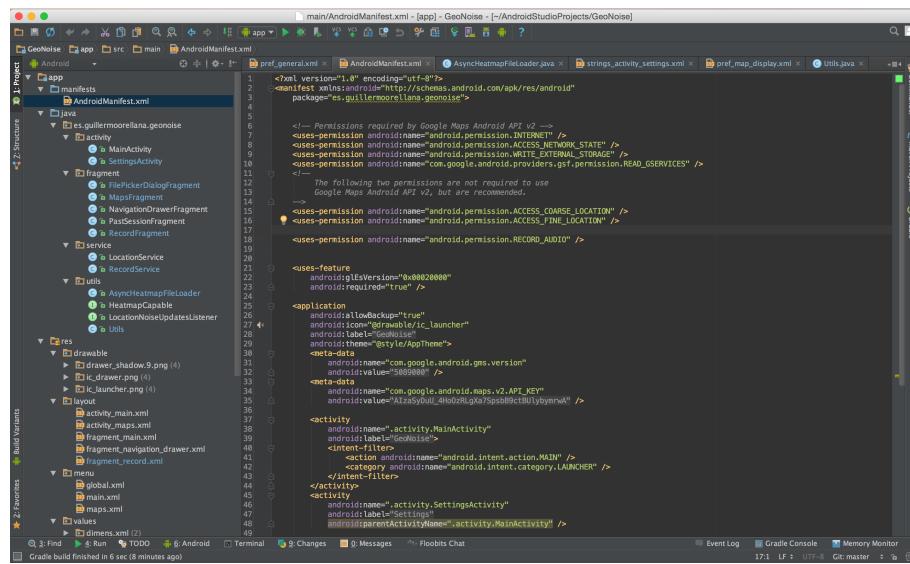


Figura 2.2: Entorno de desarrollo integrado Android Studio

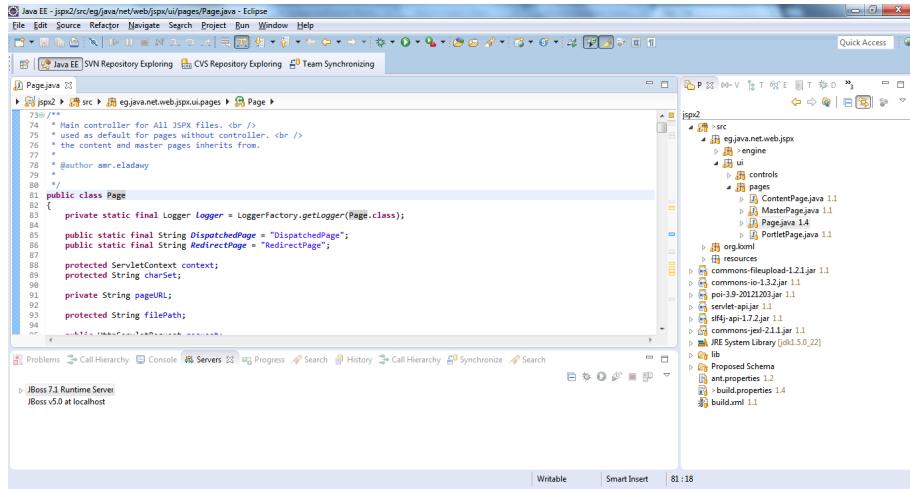


Figura 2.3: Entorno de desarrollo integrado Eclipse

Es posible elegir entre varios entornos integrados de desarrollo a la hora de desarrollar una aplicación para la plataforma Android. Los dos principales son Eclipse y Android Studio.

Android Studio es un Integrated Development Environment, o entorno de desarrollo integrado (IDE) específico para desarrollar en la plataforma Android. Está basado en el popular IntelliJ IDEA de la compañía JetBrains, pero fuertemente modificado, tanto en aspecto como en funcionalidad. Android Studio es de reciente desarrollo, llegando a su versión 1.0 inicial en Diciembre de 2014. La versión 1.0.2 se puede observar en la figura 2.2.

Eclipse es un IDE de propósito general, que provee un espacio de trabajo y un sistema de complementos muy extensible y flexible. Está escrito en Java, e inicialmente se enfocó al desarrollo en dicho lenguaje.

Gracias a su sistema de complementos, Eclipse puede ser utilizado para desarrollar aplicaciones en múltiples lenguajes: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby (incluyendo el framework Ruby on Rails), Scala, Clojure, Groovy, Scheme y Erlang. Eclipse fue también el primer IDE con soporte para desarrollo en la plataforma

Android. La figura 2.3 muestra la versión 4.4 Luna.

A la hora de realizar el proyecto, se consideraron ambas soluciones, y se optó por utilizar el entorno de desarrollo integrado Android Studio, por varias razones.

- Android Studio utiliza Gradle como su sistema de compilación, el cual resulta mucho más claro y menos tedioso de lidiar que ANT, el utilizado por Eclipse. Con Gradle, es mucho más sencillo declarar dependencias de librerías externas, característica utilizada en el apartado 2.2.5 y siguientes. Dichas dependencias pueden estar incluso basadas en el sistema de gestión de paquetes Maven; proveyendo facilidad adicional para gestionar versiones de dependencias.
- Android Studio posee una superior cantidad y calidad de análisis en tiempo real del código del proyecto. Autocompletado de código, refactorización asistida y sugerencias de optimización son algunas de las características en las que supera a Eclipse.
- El editor gráfico de interfaces gráficas de usuario, aunque presente en Eclipse de manera básica, es mucho más completo, rápido y preciso en Android Studio.
- El único aspecto en el que Eclipse supera a Android Studio es en el soporte para extensiones nativas de Android, escritas en C++, que todavía no ha sido implementado en Android Studio. Sin embargo, en este proyecto no se hace uso de dichas extensiones, por lo que no supone un problema.

2.2.2. Estructura del código

La estructura de la aplicación sigue la estructura estándar de proyecto Android que se sigue en el entorno de desarrollo Android Studio. Se distinguen cuatro grupos principales:

Manifiestos son archivos que presentan la aplicación al sistema operativo y proveen información básica sobre la misma. Esta información incluye el paquete Java de la aplicación, los componentes de la misma (actividades, servicios, eventos de emisión, proveedores de contenido...), los permisos requeridos por la aplicación, el nivel mínimo de API Android requerido y los permisos proporcionados por la aplicación.

Código fuente escrito en el lenguaje de programación Java, el cual será compilado posteriormente y convertido a código intermedio. Toda la lógica de la aplicación está implementada de esta manera. El código fuente a su vez se organiza dentro de paquetes, que son agrupaciones lógicas y funcionales de distintos archivos de código fuente.

Recursos que incluyen todo elemento gráfico, diseño de interfaces gráficas de usuario o componentes de las mismas, composición de menús contextuales y valores de variables. Es posible proveer distintas versiones de cada uno de los recursos enfocadas a características concretas de los dispositivos. Por ejemplo, es posible especificar un recurso para un idioma del teléfono, tamaño de pantalla, ratio, orientación del dispositivo, versión de la API Android e incluso si el dispositivo se encuentra en modo nocturno o no. Esta característica es aprovechada en el proyecto para proveer una aplicación bilingüe, que muestre un idioma u otro dependiendo del que tenga configurado el dispositivo en el que la aplicación ha sido instalada.

Instrucciones de compilación escritos en el Domain Specific Language, o lenguaje específico del dominio (DSL) de Gradle. Esto es una herramienta de automatización de proyectos que permite manejar de manera cómoda y eficiente tareas como gestión de dependencias, compilación, empaquetado y publicado de artefactos.

A su vez, el código fuente está dividido en subpaquetes. Se ha creado un paquete para las actividades, otro paquete para los fragmentos, otro paquete para los servicios y un último paquete para las clases de utilidad y misceláneas. Dicha estructura puede observarse en la figura 2.4.

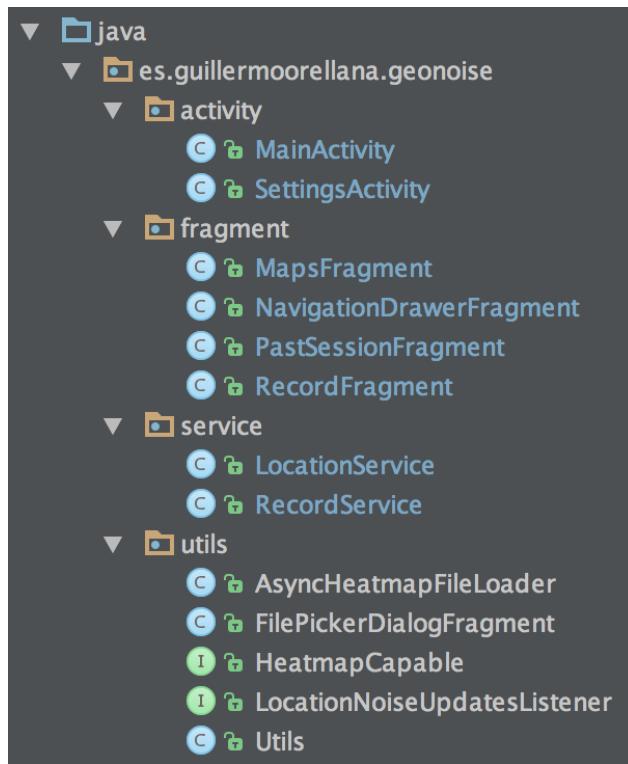


Figura 2.4: Organización del código fuente.

2.2.3. Interfaz gráfica

La interfaz gráfica de esta aplicación se ha desarrollado siguiendo las guías de diseño vigentes para la versión del sistema KitKat. Estas no son las más modernas, ya que recientemente han sido actualizadas a Lollipop. En esta última actualización, las guías de diseño cambiadas hacia el llamado “Material Design” (diseño material). Se ha desestimado seguir las dadas su novedad, que implica una nueva curva de aprendizaje y posible escasez de recursos de apoyo.

No obstante, la interfaz es sencilla e intuitiva, y resulta familiar para todo usuario del sistema operativo Android. Además se han incluido explicaciones y guías de usuario dentro de la aplicación, para mejorar su usabilidad y asegurar el correcto uso de la misma por parte del usuario.

En Android hay dos maneras de definir una interfaz gráfica: programáticamente o por archivos de recurso XML. Se ha optado por la segunda opción. Esto disminuye el acoplamiento en el código, aumenta la reusabilidad y la claridad del mismo. Por otra parte, el entorno de desarrollo permite previsualizar el resultado de dichos archivos XML con bastante fidelidad.

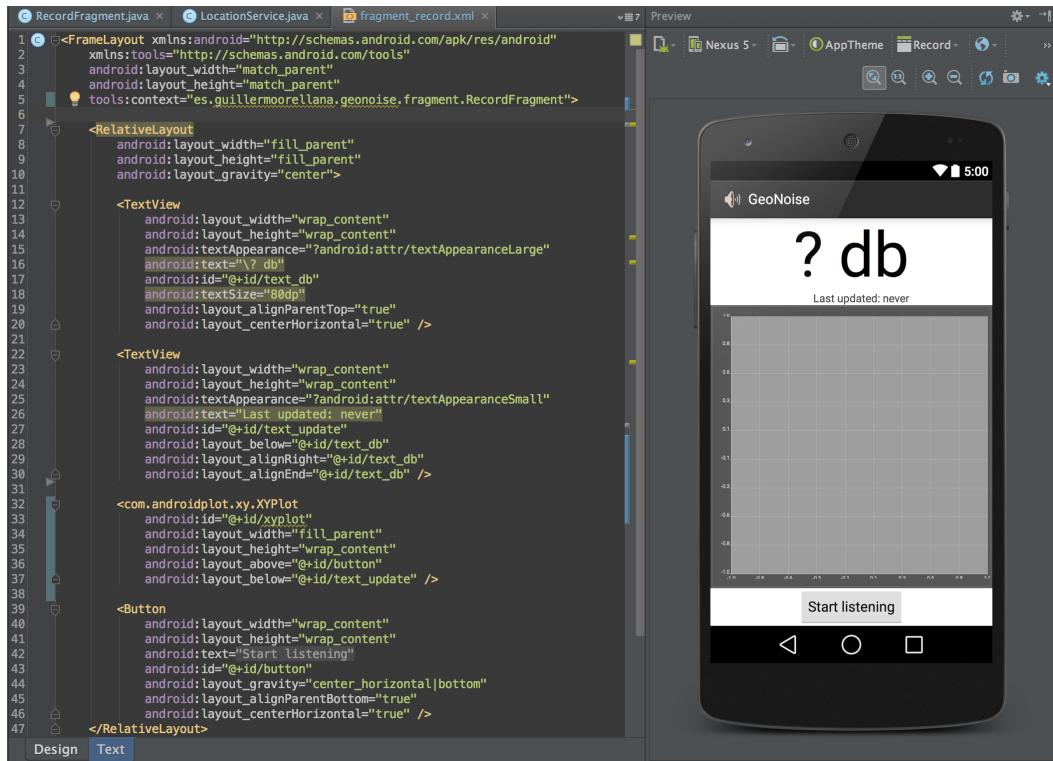


Figura 2.5: Editor de archivos de recurso XML con vista previa.

Además, en consonancia con las últimas recomendaciones de diseño de aplicaciones de Google, se han utilizado fragmentos cuando ha sido conveniente; incrementando la modularidad de los componentes de la interfaz gráfica y su potencial

reusabilidad.

Para el diseño de los archivos XML se ha hecho uso de la herramienta incorporada para tal propósito en el entorno de desarrollo Android Studio; la cual nos brinda la posibilidad de tener una vista previa del resultado de la descripción en XML que estamos realizando de la interfaz gráfica de usuario. Como ejemplo, en la figura 2.5 se puede observar la edición de la interfaz gráfica de usuario de la actividad engargada de grabar los sonidos.

Si así se desea, también cambie la posibilidad de diseñar las interfaces gráficas de usuario de la aplicación mediante la misma herramienta, pero de una manera completamente visual, tal y como se observa en la figura 2.6. De esta manera, se sacrifica precisión y control por comodidad y claridad.

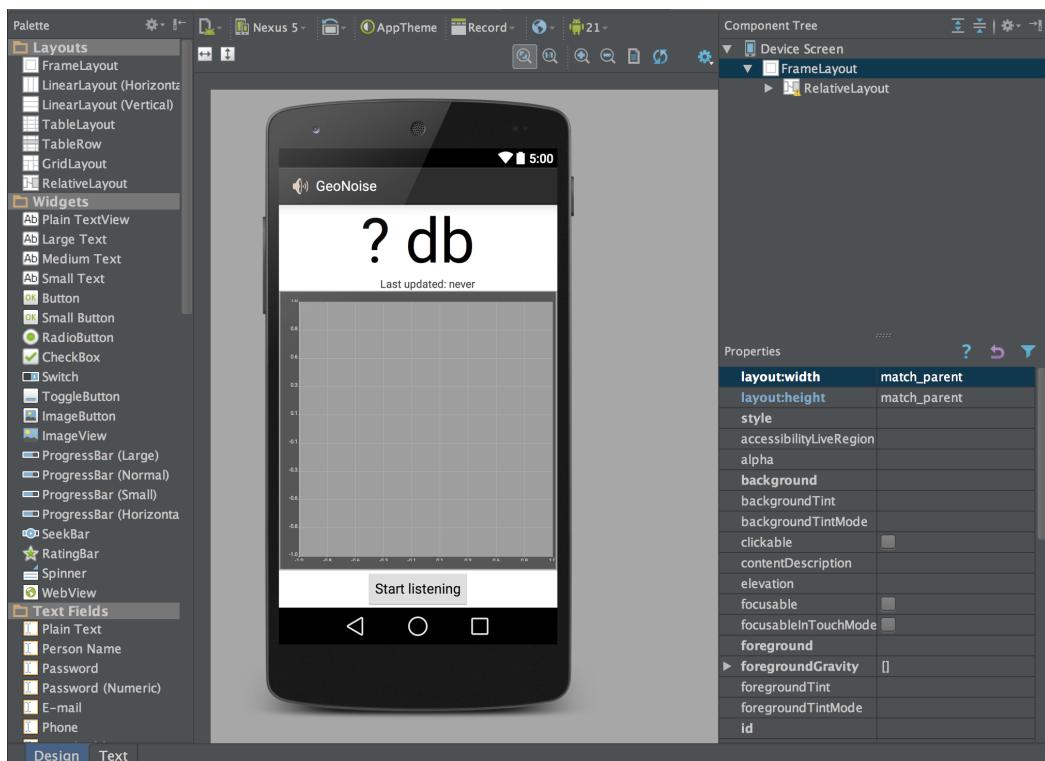


Figura 2.6: Edición completamente visual de la interfaz gráfica de usuario de la aplicación.

Dentro de cada pantalla, la estructuración de los elementos que se observan se realiza a través de layouts. Los layouts pueden definirse como contenedores de una o más vistas, que ayudan al posicionamiento de cada una de ellas dentro de la aplicación así como a controlar el comportamiento de las mismas. Este concepto encaja a la perfección con el anidado de etiquetas de XML. En el ejemplo de la figura 2.5, el código XML es el presente en el extracto 2.1. En dicho extracto se puede observar cómo el elemento **RelativeLayout** contiene a varios elementos a su vez, entre otros **TextView** y **Button**, y cómo estos elementos interiores definen su posición y tamaño en relación al resto de elementos.

```
1 <RelativeLayout
2     android:layout_width="fill_parent"
3     android:layout_height="fill_parent"
4     android:layout_gravity="center">
5     <TextView
6         android:layout_width="wrap_content"
7         android:layout_height="wrap_content"
8         android:text="\? db"
9         android:id="@+id/text_db"
10        android:textSize="80dp"
11        android:layout_alignParentTop="true"
12        android:layout_centerHorizontal="true" />
13     <TextView
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="Last updated: never"
17         android:id="@+id/text_update"
18         android:layout_below="@+id/text_db"
19         android:layout_alignRight="@+id/text_db"
20         android:layout_alignEnd="@+id/text_db" />
21     [...]
22 </RelativeLayout>
```

Fragmento de código 2.1: Descripción de la interfaz gráfica de usuario del fragmento **Medición**

Cajón de Navegación

El cajón de navegación, más conocido por su denominación inglesa Navigation Drawer, es un patrón de interfaz de usuario muy utilizado en Android. Consiste en un panel que transiciona desde el borde izquierdo de la pantalla y muestra las opciones principales de navegación de la aplicación. Se ha decidido incluir dicho patrón de diseño en la aplicación dado que permite una mejor utilización del espacio, al no bloquear ninguna parte de la pantalla mientras se encuentra sin desplegar.

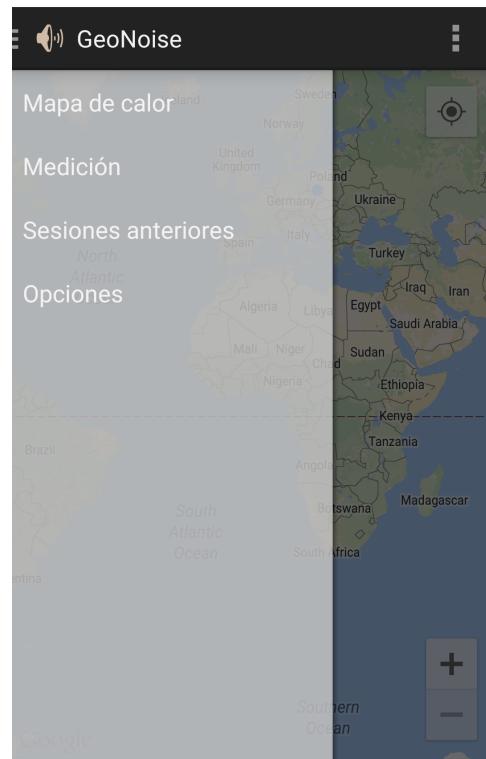


Figura 2.7: Patrón de cajón de navegación implementado en la aplicación.

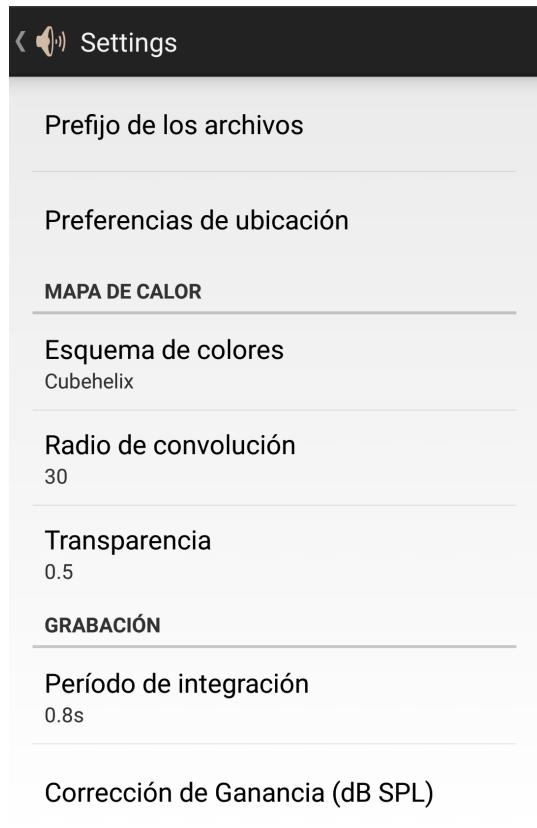


Figura 2.8: Actividad **Opciones** en un dispositivo con Castellano como lenguaje del sistema.

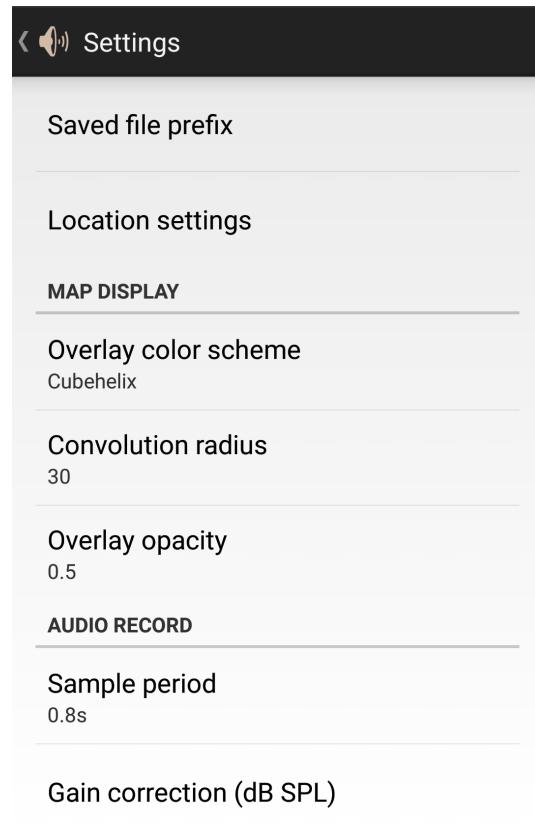


Figura 2.9: Actividad **Opciones** en un dispositivo con cualquier otro lenguaje del sistema.

Multilengüaje

Como se ha mencionado anteriormente en el apartado 2.2.2, en este proyecto se aprovecha que los archivos de recursos pueden ser discriminados según el idioma principal configurado en el dispositivo de instalación, para obtener una aplicación bilingüe. La manera de conseguirlo es, en lugar de incrustar cadenas de texto en código e interfaces gráficas de usuario, extraer las que se desee traducida a un archivo de recurso, normalmente `strings.xml`, y referenciar dichas cadenas de texto por un identificador dado. De esta manera, por cada versión adicional de `strings.xml` que

se cree, y que sea configurado para estar asociado a un lenguaje del sistema en concreto, se consigue una traducción de la aplicación.

En las figuras 2.8 y 2.9 se muestra la actividad `Opciones`. La diferencia es que en la figura 2.8 el dispositivo está configurado con el Castellano como lengua del sistema, por lo que las cadenas de texto son cargadas de la versión `strings-es.xml` asociada con dicho perfil. En la figura 2.9, el sistema no está configurado en ninguno de los casos especiales, por lo que las cadenas de texto se extraen del archivo por defecto, `strings.xml`, el cual se encuentra traducido al Inglés.

En caso de que fuera necesaria la traducción a cualquier idioma adicional, bastaría con crear una versión de `strings.xml` para dicho idioma, y asociarlo con el perfil correspondiente, de manera que sea preferida dicha versión en lugar de la versión por defecto.

2.2.4. Captura de audio

El sistema operativo Android provee una Hardware Abstraction Layer, o capa de abstracción del hardware (HAL) que conecta todas las API de alto nivel con los controladores y hardware subyacentes en cada dispositivo. La figura 2.10 muestra los distintos niveles de la arquitectura del audio en el sistema operativo Android y su interconexión.

En la aplicación se ha optado por la simplicidad, y utilizado las clases del paquete `android.media.*` que provee el framework, y que brindan un nivel de abstracción bastante cómodo.

Para garantizar que la operación de captura de audio no se ve interferida por otras funciones de la aplicación, se ha optado por crear un servicio que corra en segundo plano. Este responde a los comandos de la aplicación en primer plano, tal y como se explicó en el apartado 1.2.2.

El servicio hace uso de la clase `AudioRecord`, la cual requiere una preparación

previa a la grabación, tal y como se muestra en el fragmento de código 2.2.

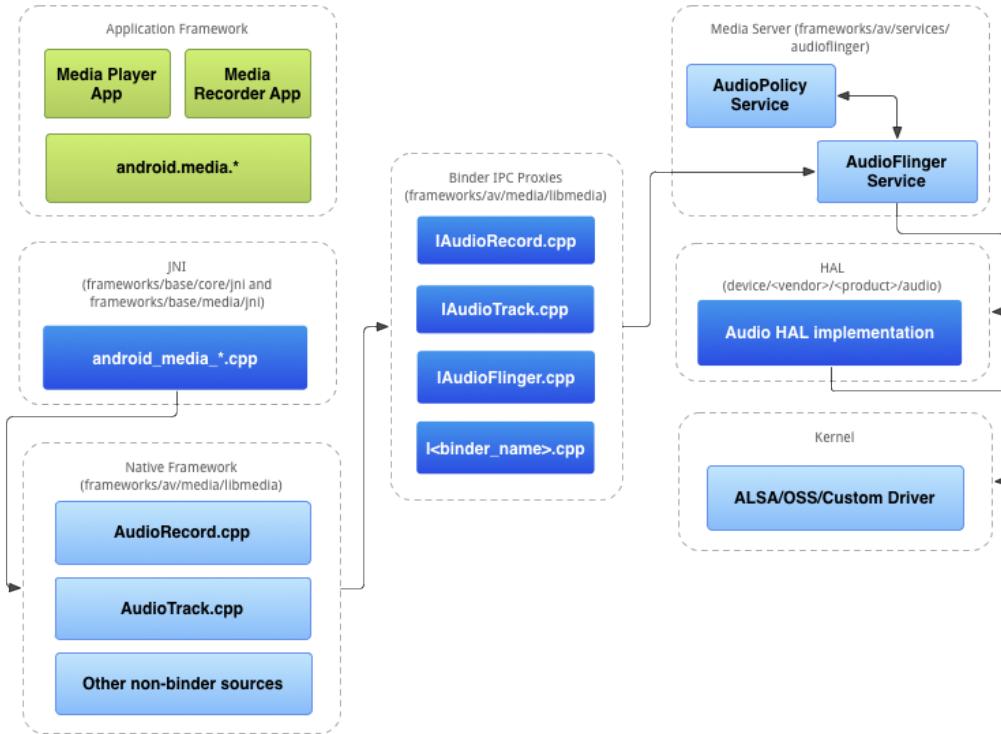


Figura 2.10: Arquitectura del audio en Android [Pro14]

Una vez la instancia de `AudioRecord` ha sido configurada, es posible comenzar la misma llamando al método `startRecording()`. De esta manera, el sistema operativo empezará a llenar el búfer interno de la instancia de `AudioRecord` según los parámetros que han sido indicados. Los datos del búfer son obtenidos bajo demanda, llamando al método `read(short[] audioData, int offsetInShorts, int sizeInShorts)` que introducirá en el búfer referenciado las muestras obtenidas hasta el momento. Una muestra de cómo es realizado en la aplicación está presente en la figura 2.3.

```

1  private void prepareAudio() {
2      try {
3          bufferSize = 10 * AudioRecord.getMinBufferSize(sampleRate,
4              ↳ AudioFormat.CHANNEL_IN_MONO,
5              ↳ AudioFormat.ENCODING_PCM_16BIT);
6          samplePeriod = 1.0f / ((float) sampleRate / (float)
7              ↳ bufferSize);
8          audio = new
9              ↳ AudioRecord(MediaRecorder.AudioSource.VOICE_RECOGNITION,
10                 ↳ sampleRate, AudioFormat.CHANNEL_IN_MONO,
11                 ↳ AudioFormat.ENCODING_PCM_16BIT, bufferSize);
12     } catch (Exception e) {
13         Log.d(TAG, "Error creating audioRecorder");
14     }
15 }
```

Fragmento de código 2.2: Preparación de AudioRecord

Las muestras están codificadas en formato PCM de 16 bits con signo. Por tanto, el límite superior del valor de las muestras es $2^{15} - 1 = 32767$. Los teléfonos móviles inteligentes suelen incorporar micrófonos de tecnología Microelectromechanical Systems, o Sistemas Microelectromecánicos (MEMS). El valor típico de saturación de estos micrófonos es de 90dB SPL. Asumiremos pues, que el valor máximo 32767 corresponde a 90dB SPL, que corresponden con $P = 0,6325\text{Pa}$ en el micrófono. Asumiremos también que para el valor $P_0 = 0,00002\text{Pa}$ se obtiene una muestra PCM con valor 1. En consecuencia, a las muestras obtenidas les es aplicado un factor $\frac{32767}{0,6325\text{Pa}} \approx 51805,5336\text{Pa}^{-1}$.

```

1 short[] buffer = new short[bufferSize];
2 int resultSize = -1;
3 resultSize = audio.read(buffer, 0, bufferSize);
4 double sum = 0;
5 //  $p_{rms} = \left[ \frac{1}{T} \int_0^T p^2(t)dt \right]^{\frac{1}{2}} \rightarrow \left( \frac{1}{T} \sum_0^T p^2[n] \right)^{\frac{1}{2}}$ 
6 for (int i = 0; i < resultSize; i++) {
7     sum += buffer[i] / 51805.5336 * buffer[i] / 51805.5336;
8 }
9 double p_rms = Math.sqrt(sum / resultSize);

```

Fragmento de código 2.3: Captura de muestras con AudioRecord y cálculo de la p_{rms}

Al mismo tiempo que se realiza la captura de sonido, se solicita al servicio de geolocalización la posición actual, para poder relacionar ambas medidas. Dicho servicio de geolocalización se trata en detalle en la sección 2.2.5.

Una vez obtenido el contenido del búfer de muestras, realizamos el procesado de las mismas. Según la definición de p_{rms} vista en la ecuación 1.2, y teniendo en cuenta que para muestras discretas la integral pasa a ser un sumatorio, se implementa el cálculo de p_{rms} tal y como en 2.3.

El siguiente paso es obtener el SPL correspondiente a la p_{rms} calculada, implementado en el fragmento de código 2.4 según la ecuación 1.1. Se ha añadido una constante `splAdjustment`, la cual se utiliza para calibrar previsibles diferencias en el sistema captador de audio de distintos teléfonos.

```

1 public double getDecibels(double level) {
2     return Math.abs(20.0 * Math.log10(level / 0.00002)) +
3         splAdjustment;

```

Fragmento de código 2.4: Cálculo del SPL.

2.2.5. Librerías externas

Ubicación

Para obtener la ubicación absoluta del dispositivo móvil, la aplicación se comunica con los GMS. Estos no sólo actúan como una capa de abstracción del hardware GPS del dispositivo, sino que además proveen servicios de valor añadido. Algunos de los beneficios incluyen un menor tiempo de precalentamiento, localización por redes WiFi y optimización del consumo energético.

No obstante, antes de hacer uso de los GMS, es necesario configurar el proyecto para que obtenga como dependencias las librerías pertinentes. Gracias al sistema de compilación de Gradle, este paso es sumamente sencillo.

```
1 dependencies {  
2     compile fileTree(dir: 'libs', include: ['*.jar'])  
3     compile 'com.google.android.gms:play-services:6.5.87'  
4     compile 'com.google.android.gms:play-services-maps:6.5.87'  
5     compile 'com.android.support:support-v4:20.0.0'  
6     compile 'com.android.support:appcompat-v7:20.0.0'  
7     compile 'com.google.maps.android:android-maps-utils:0.5.+'  
8     compile 'net.sf.opencsv:opencsv:2.0.+'  
9     compile 'com.androidplot:androidplot-core:0.6.+'  
10 }
```

Fragmento de código 2.5: Sección de dependencias dentro del archivo `build.gradle`

Los GMS en su versión más reciente se han fragmentado, así que solo es necesario declarar como dependencias las partes que sean necesarias en el proyecto, para evitar sobrecargar la aplicación. Las dependencias se añaden editando el archivo `build.gradle`. Para utilizar la parte de ubicación de los GMS, se añade la línea 3 del fragmento 2.5, dado que ubicación se encuentra en la parte general de las libre-

rías. Las dependencias son añadidas con el descriptor Maven¹ del paquete, siguiendo el esquema **paquete-padre:librería:versión**, y son descargadas del repositorio central de Maven.

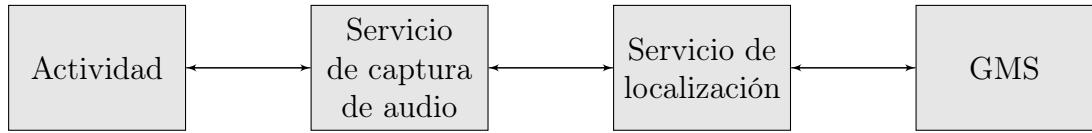


Figura 2.11: Diagrama de interacción de la aplicación y los servicios.

Para comunicarse con los GMS, es utilizado un servicio en segundo plano por los mismos motivos que en el apartado de Captura de Audio. De hecho, es un servicio intermedio, ya que la obtención en sí de la posición la realiza el servicio GMS. La interacción de la aplicación con los distintos servicios se puede observar en el diagrama 2.11. El servicio creado en la aplicación se conecta a los GMS, solicita actualizaciones y las pone a disposición del resto de la aplicación de la manera mostrada en 2.6.

```

1 private GoogleApiClient locationClient;
2 locationClient = new GoogleApiClient.Builder(this)
3         .addApi(LocationServices.API)
4         .addConnectionCallbacks(this)
5         .addOnConnectionFailedListener(this)
6         .build();
7 locationClient.connect();
  
```

Fragmento de código 2.6: Solicitud de conexión a los GMS.

Mapas

En el proyecto, se hace uso de la API Google Maps Android versión 2, la más moderna disponible, y con mejor soporte de usuario. Una aplicación que utilice los

¹El funcionamiento de la herramienta Maven y el sistema de gestión de proyectos de software asociado quedan fuera del ámbito de este proyecto. No obstante, se invita al lector a visitar maven.apache.org para conocer más acerca de los mismos.

servicios de la API Google Maps Android necesita elementos adicionales a parte de los requeridos por cualquier aplicación Android. Esto son, al igual que para utilizar los servicios de ubicación, la inclusión de las dependencias pertinentes (en caso de los mapas, existen como una dependencia separada), y adicionalmente, se necesita una clave para identificarse como usuario autorizado de la API. Esta identificación es necesaria, dado que la API de Google Maps es sólo de libre uso bajo ciertos límites. En la actualidad, el límite está fijado en los 25.000 accesos diarios a la API, a partir de los cuales se considera que se hace un uso empresarial de la misma, por el cual hay que pagar. Por tanto, al utilizar una clave de API, se identifica la aplicación como consumidor de dichos datos, y Google puede verificar que no se hace uso de esta por encima de los límites impuestos.

```
1 <application
2     android:icon="@drawable/ic_launcher"
3     android:label="@string/app_name"
4     android:theme="@style/AppTheme">
5
6     ...
7
8     <meta-data
9         android:name="com.google.android.maps.v2.API_KEY"
10        android:value="AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0"/>
11
12     ...
13
14 </application>
```

Fragmento de código 2.7: Ubicación de la clave de API de Google Maps en el manifiesto principal.

Para obtener una clave de la API de Google Maps, es necesario indicar en la consola de APIs de Google el nombre del paquete de la aplicación junto con la huella SHA-1 de la clave con la que se firma el APK final, garantizando una relación única entre una clave de API y una aplicación dadas. Por ejemplo, la huella

SHA-1 BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75 y el nombre de paquete com.example.android.mapexample darán como resultado la clave AIzaSyBdV1-cTICSwYKrZ95SuvNw7dbMuDt1KG0. La clave se indica en el manifiesto de la aplicación en una etiqueta `meta-data`, tal y como se muestra en el fragmento de código 2.7. Los detalles de cómo obtener una clave de API de Google Maps se describen en el apéndice A.

Una vez que la aplicación está autorizada a acceder a la API de Google Maps, y las dependencias de librerías han sido configuradas, mostrar un mapa en la aplicación es tarea sencilla. Los parámetros de configuración de dependencias para este componente de los GMS son los mostrados en la línea 4 del fragmento de código 2.5. Los GMS proveen un fragmento `MapFragment` listo para ser incluído en una aplicación, y con el que el resto de la aplicación puede interactuar. La manera más simple de integrarlo es incluir el fragmento en alguno de los archivos de diseño de interfaz gráfica de usuario mediante la etiqueta `fragment`, tal y como se demuestra en el fragmento de código 2.8.

```
1 <fragment xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   android:id="@+id/map"
6   android:name="com.google.android.gms.maps.MapFragment"/>
```

Fragmento de código 2.8: Inclusión del fragmento `MapFragment` en un archivo de diseño de interfaz gráfica de usuario.

Mapa de calor

Los mapas de calor se generan con la ayuda de una pequeña librería de utilidades adicional construida sobre la API de Google Maps. La configuración de dependencias para dicha librería se muestra en la línea 7 del fragmento de código 2.5. Esta no es

más que un conjunto de clases de utilidad, poco documentadas, que facilitan tareas como la coloración de partes del mapa o generación de esquemas y gradientes de colores. Una de las clases que provee es una derivación de la clase `LatLng` (que representa una latitud y longitud concretas) en la clase `WeighedLatLng`, que como la traducción de su nombre indica, no es más que asociar un peso o una intensidad a dicho par latitud-longitud. Una vez que los datos han sido extraídos del medio de almacenamiento pertinente, y posterior a cualquier procesado que se decida hacer, cada trío nivel-latitud-longitud se almacena en una instancia de `WeighedLatLng`.

```
1 HeatmapTileProvider mProvider = new HeatmapTileProvider.Builder()
2     .weightedData(weighedLatLngList)
3     .build();
4 mMap.addTileOverlay(new TileOverlayOptions()
5     .tileProvider(mProvider)
6 );
```

Fragmento de código 2.9: Inclusión del fragmento MapFragment en un archivo de diseño de interfaz gráfica de usuario.

Una vez obtenido el conjunto de `WeighedLatLng`, se hace uso del patrón de diseño *provider* `HeatmapTileProvider` para generar el gráfico que irá superpuesto, y se aplica mediante el método `addTileOverlay`. En el fragmento de código 2.9, se observa cómo se ha implementado en la aplicación. En caso de querer modificar el gradiente con el que se muestra el mapa de calor, es posible hacerlo en el proveedor, añadiendo una llamada a `gradient()` antes de la llamada al `build()` final.

Gráficos

A la hora de dibujar gráficos arbitrarios en las interfaces gráficas de usuario utilizando la API que Android provee, no hay término medio. O bien se utilizan las formas básicas predefinidas en el sistema, opción muy limitada, o bien se utiliza el soporte incluído para gráficos 2D y 3D mediante la API OpenGL, lo cual supone una

exageración en términos de complejidad. Como término medio, se ha decidido utilizar la librería *AndroidPlot*, disponible de manera libre en licencias de código abierto, la cual provee buen soporte para dibujar gráficas estáticas y dinámicas sin perderse en la complejidad que supone utilizar OpenGL. La información de dependencias para utilizar AndroidPlot se muestra en la línea 9 del fragmento de código 2.5. Una vez incluída la dependencia, el uso de la librería AndroidPlot es muy similar a los mapas de Google, aunque sin la necesidad de utilizar una clave para la API.

```

1 <com.androidplot.xy.XYPlot
2     android:id="@+id/xyplot"
3     android:layout_width="fill_parent"
4     android:layout_height="wrap_content"
5     android:layout_above="@+id/button"
6     android:layout_below="@+id/text_update" />

```

Fragmento de código 2.10: Inclusión del fragmento XYPlot en un archivo de diseño de interfaz gráfica de usuario.

AndroidPlot es una librería con multitud de funcionalidades dentro del dominio del dibujo de gráficas, desde simples gráficas de barras hasta complejas gráficas de sectores. Sin embargo, en el proyecto es utilizado un componente relativamente simple, el **XYPlot**, que tal y como el nombre indica sirve para representar un conjunto de puntos (X, Y).

Dicho componente es utilizado en el fragmento **Medición** para, cuando hay una medición en curso, mostrar una representación de la magnitud de las últimas muestras. La inclusión del componente en el fragmento se hace de manera parecida al mapa, si bien en el caso el mapa se nos proveía de un fragmento completo, en este caso **XYPlot** constituye una vista que puede ser incrustada en cualquier parte de un diseño de interfaz gráfica de usuario.

En el fragmento de código 2.10, se muestra la inclusión del componente en el diseño de la interfaz gráfica de usuario del fragmento **Medición**, y el fragmento

```
1 XYPlot xyPlot = (XYPlot) view.findViewById(R.id.xyplot);
2 // Límites del dominio (x)
3 xyPlot.setDomainBoundaries(0, HISTORY_SIZE, BoundaryMode.FIXED);
4 // Límites del rango (y)
5 xyPlot.setRangeBoundaries(0, 140, BoundaryMode.FIXED);
6 LineAndPointFormatter formatter = new LineAndPointFormatter(
7     Color.rgb(0, 0, 0), null, null, null);
8 formatter.getLinePaint().setStrokeJoin(Paint.Join.ROUND);
9 formatter.getLinePaint().setStrokeWidth(10);
10 SimpleXYSeries mySeries = new SimpleXYSeries("Noise Level");
11 mySeries.useImplicitXVals();
12 xyPlot.addSeries(mySeries, formatter);
```

Fragmento de código 2.11: Configuración y puesta en marcha del componente XYPlot de la librería AndroidPlot.

de código 2.11 se muestra la configuración realizada en el código correspondiente al mismo fragmento. En código se configuran aspectos tales como los límites del dominio y del rango, si estos son fijos o crecen con los valores, el tipo y color de las líneas, y el tipo de valores a representar. En el caso del proyecto, se solicita a la librería que sean utilizados valores implícitos en el eje de abscisas.

Formato CSV

El formato CSV ha sido elegido por su simplicidad, y gracias a ello cabría la posibilidad de implementar su escritura y lectura por completo. Sin embargo, se ha optado por utilizar una librería externa, OpenCSV, para realizar dichas operaciones. Esto permite, a parte de un menor acople y una mayor modularidad en el código del programa, centrar los esfuerzos en partes más significativas del proyecto. La información de dependencias para utilizar OpenCSV se muestra en la línea 8 del fragmento de código 2.5.

En el fragmento de código 2.12, se muestra la operación de escritura, y en el fragmento de código 2.13 se muestra la operación de lectura.

```
1 CSVWriter wr;
2 // Obtener la ruta y el nombre de archivo
3 String path = getFilePathForSession();
4 // Crear la ruta en caso de que no exista en el árbol de archivos
5 new File(Utils.getSaveDirPath()).mkdirs();
6 // Crea la instancia del escritor CSV
7 wr = new CSVWriter(new FileWriter(path));
8 // Escribir las cabeceras del archivo
9 wr.writeNext(new String[]{"magnitude", "latitude", "longitude",
10    ↵ "accuracy", "timestamp"});
11 for(Item item: measures) {
12     // Escribir una línea por cada medición
13     wr.writeNext(new String[]{String.valueOf(item.db),
14        ↵ String.valueOf(item.location.getLatitude()),
15        String.valueOf(item.location.getLongitude()),
16        ↵ String.valueOf(item.location.getAccuracy()),
17        new SimpleDateFormat("dd/MM/yyyy hh:mm:ss").format(new
18          ↵ Date(item.location.getTime()))});
19 }

```

Fragmento de código 2.12: Uso del componente **CSVWriter** de la librería OpenCSV

```
1 CSVReader reader = new CSVReader(new FileReader(string));
2 // Lista de líneas, cada línea es un array de elementos que estaban
3 // separados por una coma
4 List<String[]> lines = reader.readAll();
5 for (String[] line : lines) {
6     if (line[0].equals("magnitude")) // Cabecera
7         continue;
8     WeightedLatLng item = new WeightedLatLng(
9         new LatLng(Double.parseDouble(line[1]),
10            ↵ Double.parseDouble(line[2])),
11        Double.parseDouble(line[0]) - 40
12    );
13     ret.add(item);
14 }

```

Fragmento de código 2.13: Uso del componente **CSVReader** de la librería OpenCSV.

Capítulo 3

Plan de pruebas y verificación

3.1. Funcionamiento de la aplicación

Una vez ejecutada la aplicación, se inicia la actividad del cajón de navegación, mostrando el fragmento **Mapa de Calor**. El mapa está centrado en la ubicación actual, tal y como muestra la figura 3.1. Esta permite utilizar el mapa de manera idéntica a la que lo haríamos en la aplicación Google Maps. Desde esta pantalla, se puede o bien **Cargar datos** de alguna sesión de grabación o abrir el cajón de navegación, ya sea pulsando sobre el ícono de la aplicación, o haciendo el gesto característico de este patrón de diseño de interfaz de usuario, que es arrastrar desde fuera del borde izquierdo hacia dentro.

Una vez abierto el cajón de navegación, se presentan los distintos elementos del menú. Apreciable en la figura 3.2, el cajón de navegación da acceso a los fragmentos **Mapa de calor**, **Medición**, **Sesiones anteriores** y a la actividad de **Opciones**.

Para realizar una nueva medición, se accede a **Medición** desde el cajón de navegación, lo que hace aparecer al fragmento. Al pulsar el botón **Empezar medición**, el proceso de medición comienza. Los servicios en segundo plano son iniciados, y tras un breve retardo empiezan a aparecer en pantalla valores de la medición.

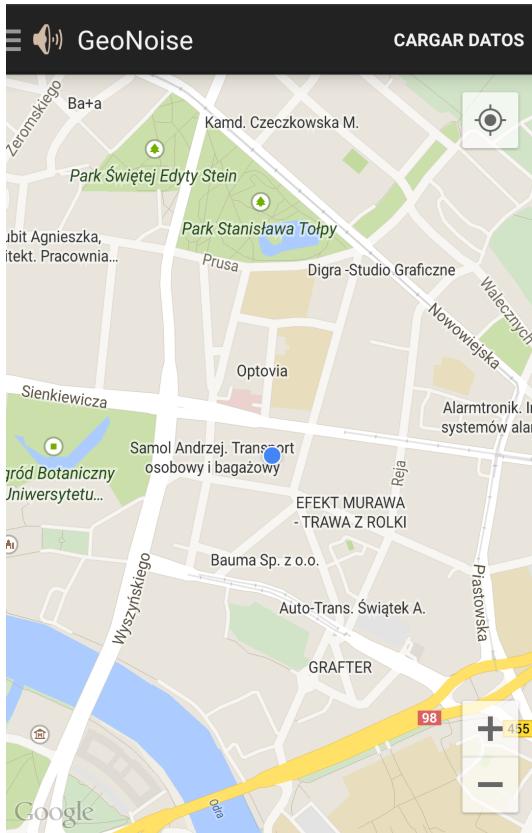


Figura 3.1: Fragmento Mapa de Calor.

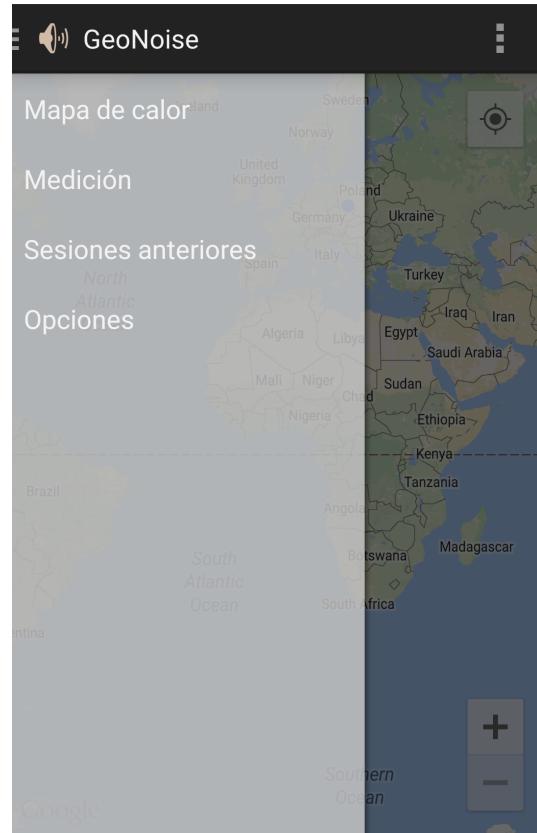


Figura 3.2: Cajón de navegación abierto.

Conforme los valores van siendo medidos y calculados, el histórico de las últimas muestras se dibuja en pantalla en el área dispuesta a tal efecto. Durante todo el proceso, los datos recabados son grabados a la memoria del dispositivo en segundo plano. Una vez finalizada la medida, se presiona el botón **Parar medición**, y los servicios en segundo plano de la aplicación vuelven al reposo. Abriendo de nuevo el cajón de navegación, es posible acceder al fragmento de **Sesiones anteriores**, donde se muestra una lista de todos los archivos producidos en grabaciones anteriores. De la misma manera, es posible volver al fragmento **Mapa de calor**.

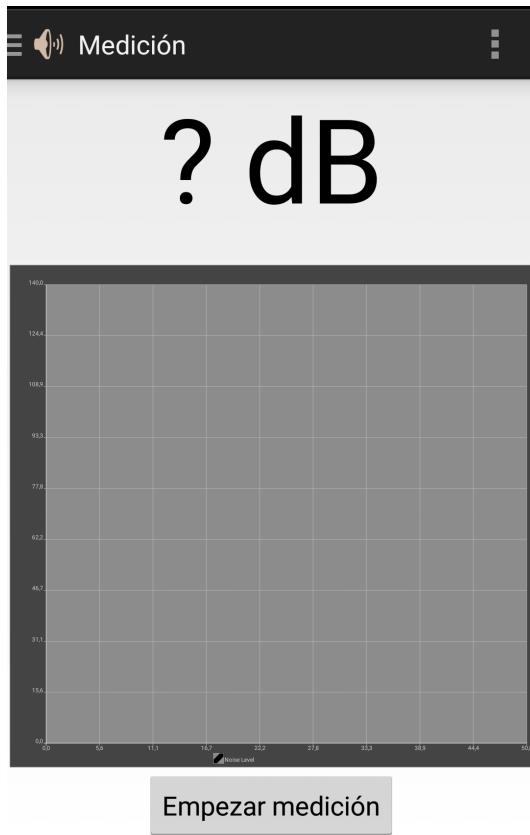


Figura 3.3: Fragmento **Medición** en reposo.



Figura 3.4: Fragmento **Medición** durante una medición.

De vuelta en el fragmento **Mapa de calor**, al pulsar el botón **Cargar datos**, se mostrará un diálogo con la lista de todos los archivos producidos en grabaciones anteriores. Al seleccionar uno de estos archivos, la aplicación calcula el mapa de calor resultante de los datos contenidos en dicho archivo, y lo superpone al mapa actual. En la figura 3.6 se aprecia un ejemplo de dicho comportamiento. Cambiando el nivel de acercamiento del mapa, se obtienen distintos niveles de agregamiento de los datos.

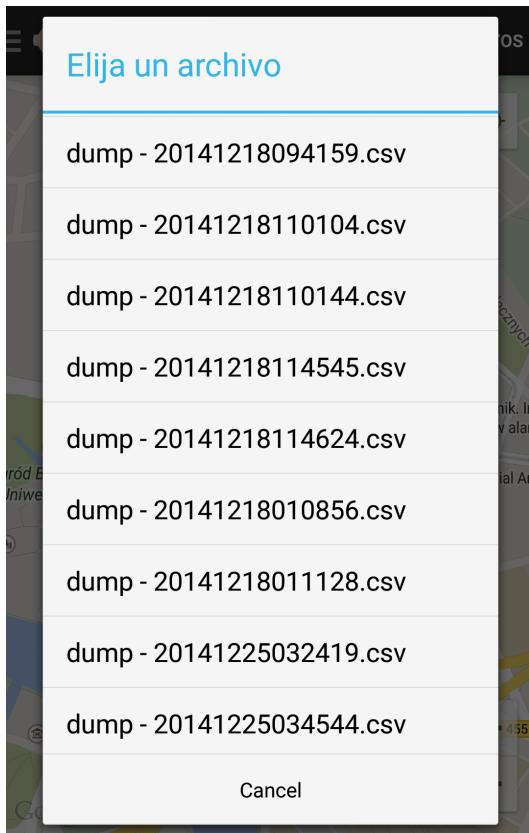


Figura 3.5: Diálogo para la elección del archivo de sesión.

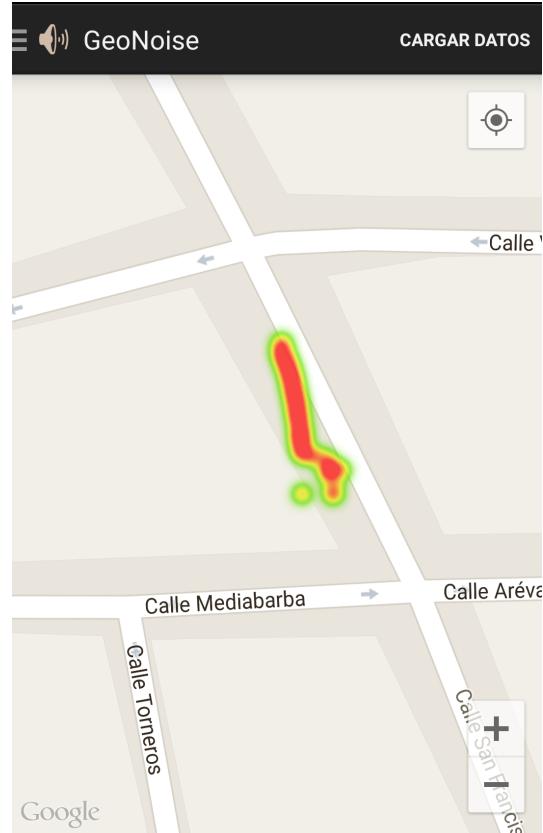


Figura 3.6: Fragmento Mapa de Calor mostrando datos cargados.

La aplicación dispone también de una actividad de **Opciones**, donde se pueden configurar varios aspectos del comportamiento de la misma. El acceso a dicha actividad, puede ser llevado a cabo de dos maneras. La primera es usando el mismo cajón de navegación que para acceder al resto de los fragmentos de la aplicación. La segunda es haciendo uso del menú contextual que aparece al tocar en la barra de acción cuando nos encontramos en cualquier fragmento que no sea **Mapa de calor**, ya que en este último el botón **Cargar datos** ocupa su lugar. En la figura 3.8 se observa el contenido de dicha actividad.

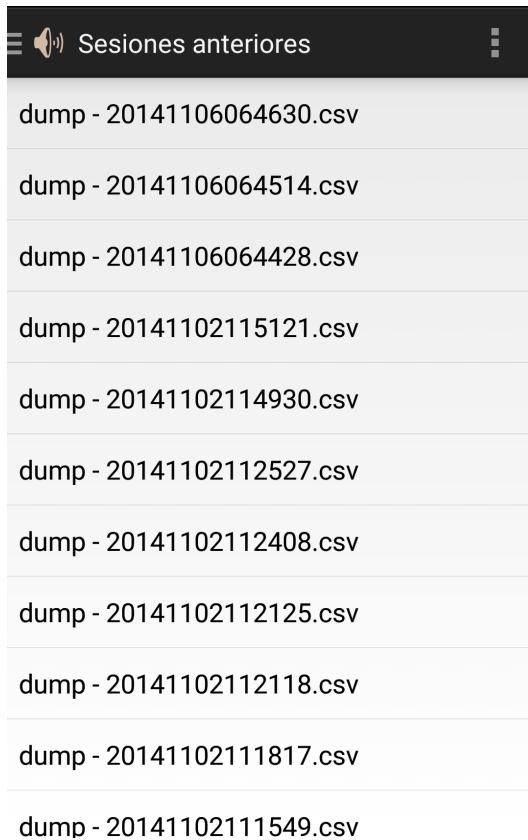


Figura 3.7: Fragmento Sesiones mostrando la lista de archivos de sesiones anteriores.

Dentro de la actividad **Opciones**, cabe la posibilidad de realizar múltiples configuraciones. Estas son:

- Renombrar el prefijo utilizado en el nombre de los archivos de guardado en el almacenamiento del dispositivo
- Acceder a las preferencias de ubicación del sistema, para poder habilitar cómodamente la medida de alta precisión

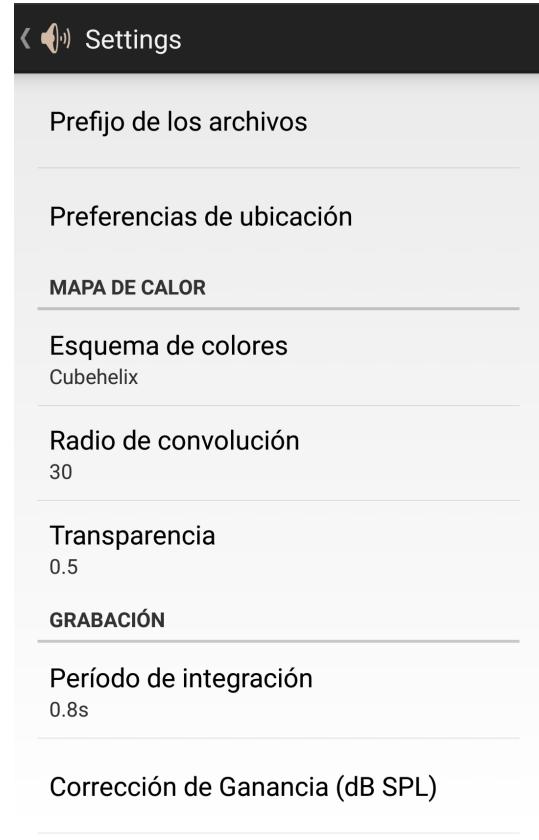


Figura 3.8: Actividad **Opciones**, donde se configuran distintos aspectos de la aplicación.

- Cambiar el esquema de colores utilizado en la generación del mapa de calor
- Alterar el radio de convolución que se utiliza en la generación del mapa de calor
- Modificar la transparencia con la que el mapa de calor se superpone al mapa geográfico
- Variar el período de integración con el que se realizan las medidas
- Aplicar una corrección de ganancia a las medidas

Por último, cabe destacar que, al guardarse los archivos en la memoria del dispositivo, estos pueden ser extraídos y utilizados posteriormente, ya sea para ser procesados, representados de diferente manera o agregados con otros datos. Dependiendo del modelo de teléfono inteligente utilizado, este dispondrá o no de tarjeta de almacenamiento externa. La aplicación preferirá guardar sus datos en la tarjeta de almacenamiento externa, y solo cuando esta no exista, se grabarán los datos en la memoria interna del teléfono. Un ejemplo del contenido de un archivo de sesión de captura de la aplicación está disponible en la figura 3.9.

```
"magnitude","latitude","longitude","accuracy","timestamp"
"62.09779","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"60.62046","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"57.23939","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"55.49906","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"56.60217","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"63.38877","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"64.22864","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"72.13268","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
"78.58884","51.1157184","17.0541213","12.618","31/01/2015 04:13:03"
```

Figura 3.9: Extracto de un archivo CSV producido por la aplicación

3.2. Precisión de las medidas

Para cerciorarse de que los valores de nivel de presión sonora mostrados en la aplicación se ajustan a la realidad, es necesario comparar los valores para una misma fuente con los obtenidos por un aparato de medición calibrado.

Tras realizar dichas medidas, la aplicación deberá de ser configurada conforme a los resultados obtenidos, introduciendo los parámetros pertinentes en la pantalla dispuesta a dicho efecto.

Dicho proceso es necesario para cada micrófono distinto que sea usado con la aplicación, ya sea por usarla en un teléfono distinto o por utilizar un micrófono externo, ya que las características de cada uno varían, y no puede garantizarse la fidelidad de los resultados de un micrófono con los parámetros de otro.

3.2.1. Realización de medidas en laboratorio

Para las pruebas de calibrado del nivel de presión sonora, se han realizado en el laboratorio medidas de nivel de presión sonora emitido por un monitor de estudio, primero por un sonómetro calibrado, y después por la aplicación.

El sonómetro utilizado ha sido el Svantek SVAN 977.

El micrófono utilizado por la aplicación ha sido el integrado en el teléfono, modelo LG Nexus 5. Según [LG], es un micrófono del tipo microelectromecánico, o MEMS según sus siglas en inglés, de la marca Goertek. Sin embargo, la hoja de características del micrófono no está disponible.

Los resultados de la batería de pruebas pueden observarse en la tabla 3.1. Es notoria la similitud de las mediciones para el tono puro, y un resultado del todo esperado dado que cae dentro del rango de frecuencias típicas de trabajo de un micrófono de un dispositivo móvil.

Aparato	Sonómetro	Nexus 5
Tono 440 Hz	68.7 dB	68.6 dB
Fuente sonora apagada	37.4 dB	43.2 dB
Ruido blanco	69.3 dB	62.6 dB
Ruido Rosa	69.2 dB	64.1 dB
Canción	63.1 dB	59.3 dB

Tabla 3.1: Tabla de comparacion de mediciones

Sin embargo, cuanto más nos alejamos de dicho funcionamiento típico del micrófono del dispositivo móvil, es posible observar mayores discrepancias. En el límite inferior, se observa cómo la sensibilidad de dicho tipo de micrófonos es limitada, y no es capaz de detectar niveles menores de alrededor de 43 dB.

Presentado ante un ruido blanco, la aplicación reporta un nivel menor que el observado en la lectura del sonómetro. Es de suponer que esto es debido al sonómetro siendo capaz de abarcar un espectro mayor de frecuencias en su medición, y por tanto recibiendo unas lecturas más altas.

3.2.2. Realización de medidas de campo

Calle concurrida

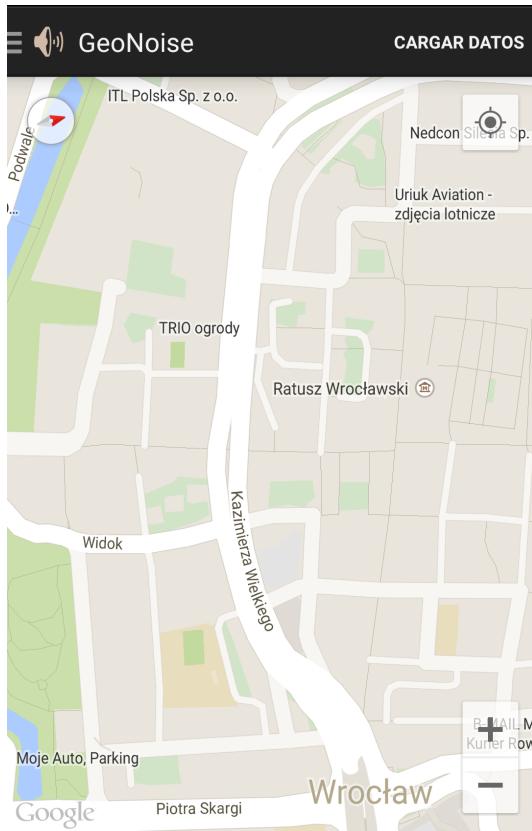


Figura 3.10: Vista del mapa de la calle antes de superponer la medida.

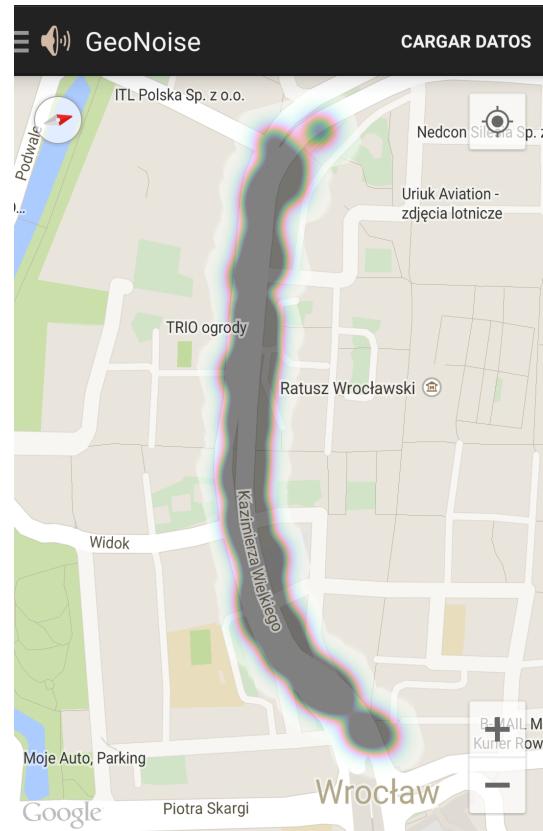


Figura 3.11: Vista del mapa de la calle con la medida superpuesta.

Para el escenario de calle concurrida, se ha elegido la calle Kazimierza Wielkiego de Wrocław, una de las calles principales de la ciudad, vertical en la figura 3.10. Se trata de una calle concurrida, con un volumen de tráfico considerable, y por tanto es de esperar un nivel de ruido elevado. Tras la realización de las medidas, es posible observar en la figura 3.11 cómo prácticamente la totalidad del terreno cubierto en

la medida se encuentra en el valor más alto.

Parque tranquilo



Figura 3.12: Vista del mapa del parque antes de superponer la medida.

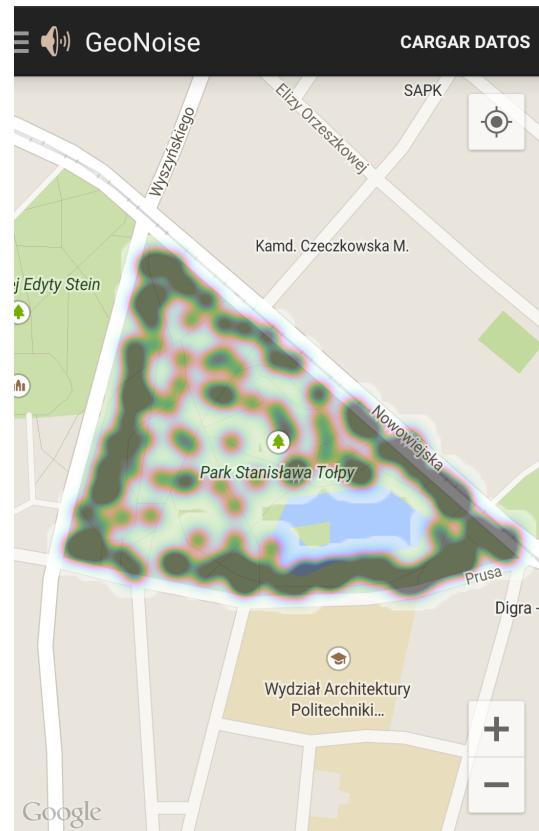


Figura 3.13: Vista del mapa del parque con la medida superpuesta.

Para el escenario de parque tranquilo, se ha elegido el parque Stanisława Toły de Wrocław, un parque de tamaño mediano con abundante vegetación, cuya disposición se observa en la figura 3.12. En este caso, se da una mayor diversidad de medidas. Las medidas de mayor intensidad se encuentran cerca de la carretera, y disminuyen de intensidad conforme se aproxima al centro del parque.

3.3. Realización de los objetivos

3.3.1. Capacidad de medir la magnitud del ruido ambiente

La aplicación es capaz de medir la magnitud de ruido ambiente en cualquier momento dado, tan solo hay que comenzar la captación y se muestra un valor estimado en decibelios. Es posible observar que dicho objetivo se cumple en la figura 3.4.

3.3.2. Capacidad de determinar la posición del dispositivo

La aplicación cumple con el objetivo de ser capaz de determinar la posición absoluta del dispositivo móvil, ya sea dentro del mapa utilizado para representar los datos obtenidos previamente, o durante la operación de recabado de datos. Es posible apreciar cómo el punto azul determina nuestra posición actual en la figura 3.1. Durante la medición no se muestra de forma directa la posición absoluta medida, pero es tenida en cuenta en todo momento de la misma.

3.3.3. Capacidad de asociar ambas mediciones

La aplicación vincula en cada operación de muestreo la magnitud medida y la posición absoluta medida, de manera que sea posible conocer a qué punto geográfico pertenece cada medida. Es posible observar dicha asociación en el contenido del archivo creado por cada sesión de grabación, como puede observarse en la figura 3.9.

3.3.4. Capacidad de almacenar los datos obtenidos

Los datos son almacenados inmediatamente después de la finalización de la sesión de grabado a la memoria del dispositivo. Estos pueden ser recuperados conectando el dispositivo móvil a un ordenador personal, o bien representados dentro de la propia

aplicación. Una muestra del contenido de un archivo de almacenamiento se puede observar en la figura 3.9.

3.3.5. Capacidad de mostrar los datos obtenidos sobre un mapa

Una vez finalizada una sesión de grabación, la aplicación está capacitada para mostrar un listado de las sesiones finalizadas. Una vez seleccionada la sesión que se quiere representar sobre el mapa, la pantalla se actualiza mostrando los datos pertinentes, tal y como se aprecia en la figura 3.6.

Capítulo 4

Conclusiones y líneas de trabajo futuras

4.1. Conclusiones

En este proyecto se ha investigado, planificado y desarrollado una aplicación Android que permite una primera aproximación al problema de realizar medidas de nivel de presión sonora de manera fácil, clara y sin equipamiento extra. Además, posibilita la visualización de las mismas superpuestas a un mapa, en forma de mapa de calor.

Durante el proyecto se han tenido en cuenta todos los conocimientos adquiridos durante el estudio de la titulación de Ingeniero Técnico de Telecomunicación, especialidad en Sonido e Imagen. Estos conocimientos han sido de crítica importancia para superar problemas que han ido apareciendo en el transcurso del proyecto, tales como el control automático de ganancia (CAG), o la percepción subjetiva de los esquemas de colores de los mapas de calor.

Sin embargo, uno de los mayores retos del proyecto no ha sido su complejidad técnica en el apartado acústico, que es más bien simple, sino su implementación

en el sistema operativo Android. Durante la titulación no se han cursado ninguna asignatura relativa a programación móvil o Java. Por tanto, el factor de mayor novedad y en el que más se ha aprendido durante la realización del proyecto, es el componente Android del mismo. Este valioso conjunto de conocimientos sienta las bases para, en un futuro, poder construir aplicaciones de mayor complejidad técnica y científica en la plataforma Android.

La aplicación generada como resultado de este Proyecto Fin de Carrera, aún con ciertas limitaciones, responde de manera satisfactoria a los objetivos autoimpuestos al principio del mismo; como se ha demostrado en el capítulo de pruebas. Por tanto, al haber obtenido un producto funcional, se considera el proyecto como exitoso.

4.2. Líneas de trabajo futuras

Durante el desarrollo del proyecto, y especialmente durante las pruebas finales del mismo, han surgido ideas y mejoras para el mismo, que son posibles de implementar en un futuro.

Almacenamiento remoto

El almacenamiento en un servidor externo de los datos obtenidos por la aplicación es una mejora que se presenta obvia, pero no trivial. A implementar quedaría toda la lógica de sincronización, junto con la no sencilla elección de la infraestructura externa a procura. Esto abre camino a toda una serie de mejoras futuras.

Plataforma web de visualizado y agregación

Como acompañante a la aplicación, y extendiendo la idea de almacenamiento remoto, se puede construir una plataforma web. En ella, con la entrada de las medidas de múltiples usuarios, se puedan agregar los datos. Uno de los

posibles resultados de dicha agregación es poder representar los datos de toda una ciudad.

Exploración de alternativas en el formato de almacenamiento

En este proyecto se ha utilizado CSV como formato de almacenamiento, pero cabe la posibilidad de que otros formatos brinden un valor añadido o incluso posibiliten nuevos usos para la aplicación. Formatos a considerar serían JavaScript Object Notation, o notación de objetos de Javascript, eXtensible Markup Language, o lenguaje de marcado extensible (XML).

Análisis en frecuencia de las medidas

Extendiendo las funcionalidades acústicas de la aplicación, sería de gran utilidad la inclusión de representación en frecuencia de las medidas, y aplicar mediciones tales como división por octavas o tercios de octava.

Asociación de colores y niveles en la vista de Mapa de Calor

El mapa de calor proporciona una manera muy directa de visualizar datos en un plano, pero en caso de necesitar una mayor exactitud, actualmente es necesario acudir a los archivos de sesión y buscar los valores. La presencia de una barra con el esquema de colores utilizado y la escala a la que corresponde fue estimada inicialmente como una característica a incluir, pero tras una investigación inicial, resultó de demasiada complejidad

Inclusión de curvas de ponderación isofónicas

Los datos recabados por la aplicación no reciben ningún tratamiento adicional aparte del cálculo del nivel de presión sonora. Como posible extensión a la aplicación en su estado actual, es interesante considerar la inclusión de las distintas curvas de ponderación para ser aplicadas sobre las medidas. Para esto es necesario implementar primero el análisis en frecuencia de las medidas.

Mediciones basadas en posición

Según el diseño actual, las mediciones se realizan periódicamente según un intervalo de tiempo configurable. Sin embargo, con ayuda de la API de localización de GMS, es posible llevar a cabo mediciones cada vez que nos encontramos a una distancia arbitraria de la última medición. De esta manera, es posible realizar mediciones en intervalos de espacio constantes, en oposición a las mediciones en intervalos de tiempo constantes.

Asistencia en mediciones complejas

Las normativas de contaminación acústica, mapas oficiales de ruido y mediciones acústicas en general presentan unos complejos y muy diversos requisitos a la hora de realizar mediciones acústicas. Cabe la posibilidad en la aplicación de añadir medidas asistidas, en las que se indique al usuario los pasos a realizar, y la aplicación se encargue de los tiempos de medida, ponderaciones, distancias, etc.

Guillermo Orellana Ruiz

10 de febrero de 2015

Apéndice A

Obtención de clave para API de Google Maps

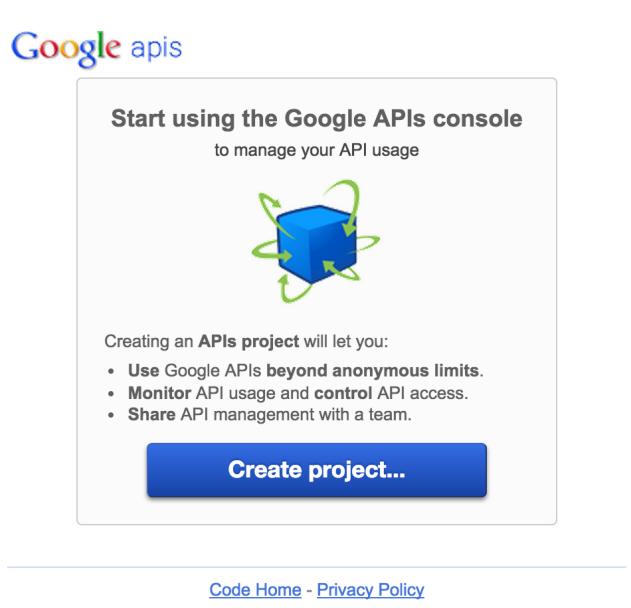


Figura A.1: Consola de APIs de Google sin proyectos.

El proceso para obtener una clave de usuario para la API pública de Google Maps es muy parecido a obtener una clave para cualquiera del resto de APIs

de Google, ya que todas pasan por la consola de APIs de Google, accesible en <https://code.google.com/apis/console/>. De ser la primera vez accediendo a la consola de APIs de Google, se muestra en pantalla la opción de crear un nuevo proyecto, tal y como se observa en la figura A.1.

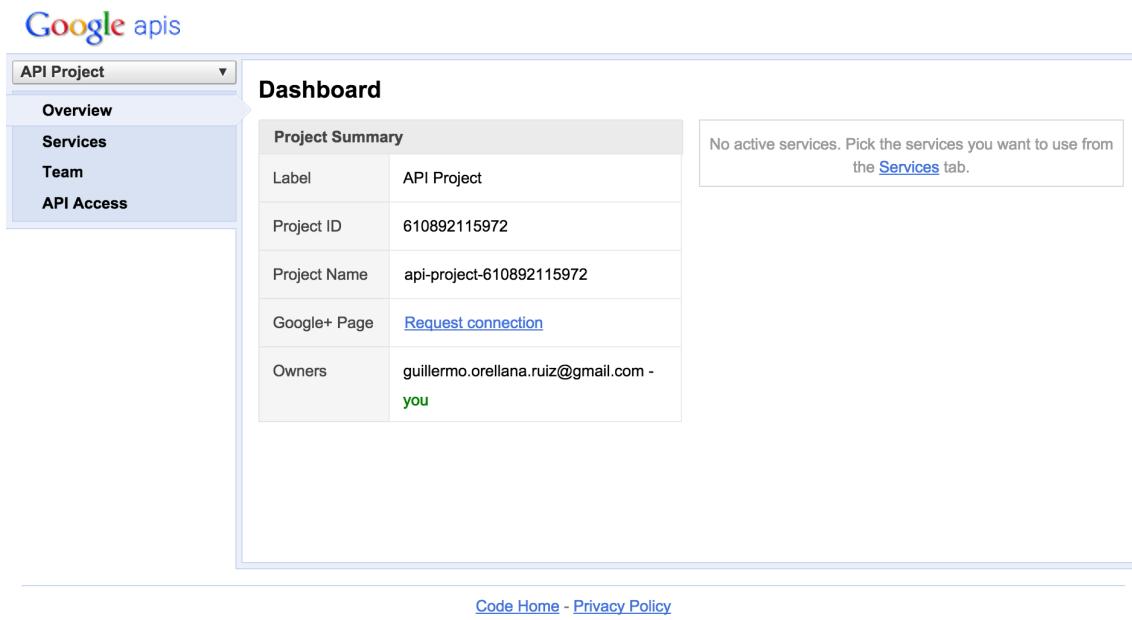


Figura A.2: Proyecto API Project creado por defecto y en blanco.

De no haber creado ningún proyecto con anterioridad, será creado el proyecto por defecto API Project. Es posible configurar el nombre del proyecto a posteriori, aunque no es de importancia para el propósito de obtener una clave de API de Google Maps. El proyecto por defecto se muestra en la figura A.2.

Google Contacts CardDAV API	<input type="button" value="OFF"/>	Courtesy limit: 20,000,000 requests/day
Google Container Engine API	<input type="button" value="OFF"/>	Courtesy limit: 1,000,000 requests/day
Google Maps Android API v2	<input checked="" type="button" value="ON"/>	
Google Maps Coordinate API	<input type="button" value="OFF"/>	Courtesy limit: 1,000 requests/day
Google Maps Embed API	<input type="button" value="OFF"/>	Courtesy limit: 2,000,000 requests/day
Google Maps Engine API	<input type="button" value="OFF"/>	Courtesy limit: 10,000 requests/day

Figura A.3: Servicio *Google Maps Android API v2* activado.

Una vez en la vista de proyecto, es necesario acceder a la sección **Services**, y localizar en la lista de servicios ofrecidos por Google el llamado *Google Maps Android API v2*, y activarlo. Para poder activarlo, es necesario leer y aceptar el acuerdo final de usuario, por el cual Google licencia un uso limitado de la API. Una vez que este servicio esté activado, se mostrará tal y como en la figuraA.3.

The screenshot shows the 'API Access' section of the Google API Console. On the left, there's a sidebar with options: API Project (selected), Overview, Services, Team, API Access (selected), Reports, and Quotas. The main content area has three sections:

- API Access**: A note about preventing abuse through OAuth tokens or API keys.
- Authorized API Access**: Information about OAuth 2.0, mentioning it allows users to share data while keeping their credentials private. It shows a single client ID entry:

API key:	AIzaSyA0O3fEDK0t89wgD0rtEvTji3wWIvQmBes	Generate new key...
Refers:	Any referer allowed	Edit allowed referrers...
Activated on:	Feb 7, 2015 7:43 AM	Delete key...
Activated by:	gullermo.orellana.ruiz@gmail.com – you	

 Buttons below this table allow creating new keys for Server, Browser, Android, or iOS.
- Simple API Access**: A note about using API keys for projects without user data access.
- Notification Endpoints**: A note about identifying domains for webhook notifications, with a table showing 'Allowed Domains: No domains allowed' and an 'Edit...' button.

Figura A.4: Sección API Access de la consola de APIs de Google.

Con el servicio activado, el siguiente paso consiste en acceder a la sección API Access, donde se gestionan los métodos de autentificación y las claves del proyecto seleccionado. Dicha sección está representada en la figura A.4. Para continuar, es necesario presionar sobre el botón Create new Android key....

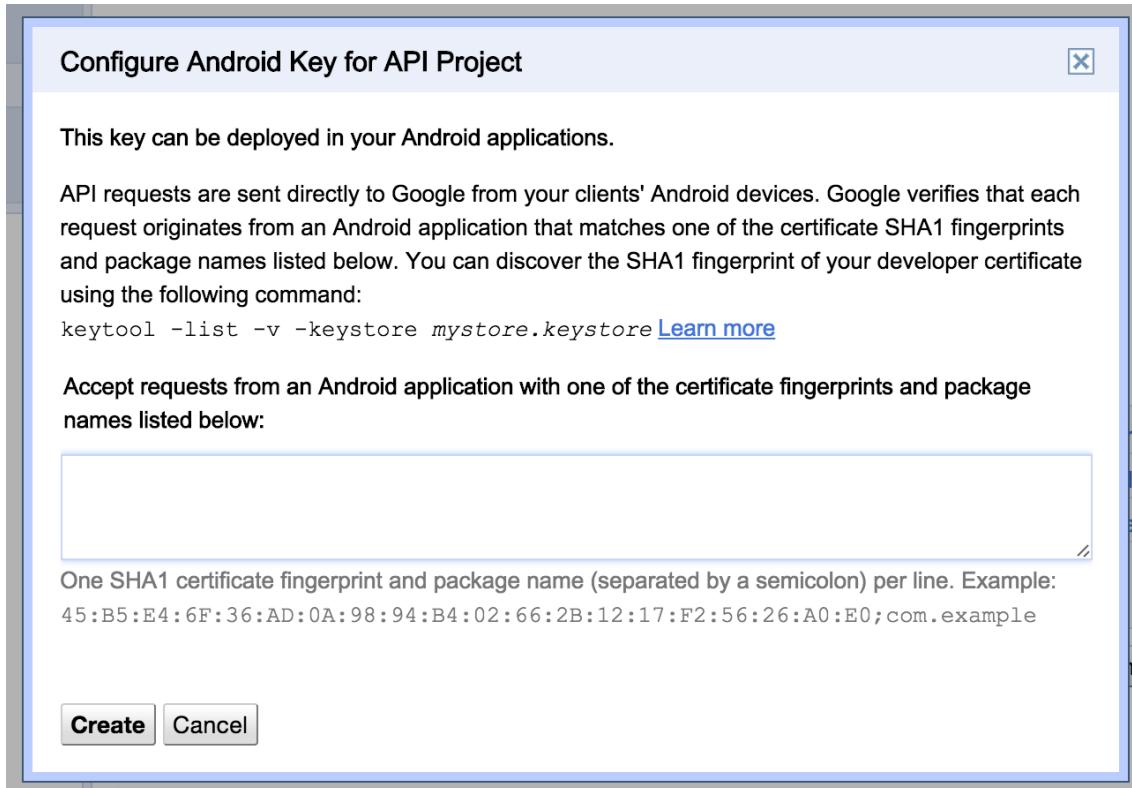


Figura A.5: Ventana de diálogo donde se introducen los datos necesarios para la clave.

Se presenta entonces una ventana de diálogo, representada en la figura A.5, en la que es necesario introducir la huella SHA-1 del certificado utilizado para firmar la aplicación, y el nombre de paquete Java de la misma. En caso de no conocer la huella SHA-1 del certificado, la misma ventana de diálogo indica la manera de obtenerla.

Simple API Access

Use API keys to identify your project when you do not need to access user data. [Learn more](#)

Key for Android apps (with certificates)	
API key:	AIzaSyChxDxt2sewhSZ26uN7oX_EAR0brR5DQQQ
Android apps:	45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F 2:56:26:A0:E0;com.example
Activated on:	Feb 7, 2015 7:45 AM
Activated by:	guillermo.orellana.ruiz@gmail.com – you

[Generate new key...](#)
[Edit allowed Android apps...](#)
[Delete key...](#)

Figura A.6: Sección de la consola mostrando los datos de acceso a la API.

Una vez finalizados todos los pasos, de vuelta en la sección **API Access**, habrá una nueva sección en la que estarán contenidos la clave de la API, la aplicación Android autorizada a utilizarla, la fecha de activación, y la identidad de quien la activó. Dicha sección puede ser apreciada en la figura A.6.

Bibliografía

- [eMa14] eMarketer. Smartphone users worldwide will total 1.75 billion in 2014. <http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536>, 2014.
- [Goo14] Google. Android developers guide. <http://developer.android.com/>, 2014.
- [Gre11] D. A. Green. A colour scheme for the display of astronomical intensity images. *Bulletin of the Astronomical Society of India*, 39:289–295, July 2011.
- [LG] LG. *Service Manual Nexus 5 LG D821*.
- [LS53] F. Langford-Smith. *Radiotron Designer’s Handbook*. RCA, 4th edition, 1953.
- [Pro14] Android Open Source Project. Android audio. <https://source.android.com/devices/audio/index.html>, 2014.
- [RL14] Kathy Nagamine Ramon Llamas, Ryan Reith. Smartphone market share q3 2014. Technical report, IDC Corporate USA, 2014.
- [Uni03] Unión Europea. *Directiva 2003/10/CE, de 6 de febrero de 2003, sobre las disposiciones mínimas de seguridad y de salud relativas a la exposición*

de los trabajadores a los riesgos derivados de los agentes físicos (ruido),
2003.

[Wik14a] Wikipedia. Pink noise. http://en.wikipedia.org/wiki/Pink_noise,
2014.

[Wik14b] Wikipedia. Pulse-code modulation. http://en.wikipedia.org/wiki/Pulse_code_modulation, 2014.

[Wik14c] Wikipedia. White noise. http://en.wikipedia.org/wiki/White_noise, 2014.

[Wik15] Wikipedia. Heat map. http://en.wikipedia.org/wiki/Heat_map,
2015.