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./Dª. _____

D./Dª. _____

D./Dª. _____

para juzgar el Proyecto Fin de Carrera titulado:

APLICACIÓN ANDROID PARA MEDICIÓN DE RUIDO AMBIENTAL Y SU REPRESENTACIÓN GEOLOCALIZADA

del alumno *D./Dª. Guillermo Orellana Ruiz*
dirigido por *D./Dª. 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, 5 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
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

Índice

Índice de figuras

Índice de Tablas

Introducción y visión general

Contenido

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, estamos hablando 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 múltiples y muy caros aparatos tales como sonómetro, calibrador, pantalla anti-viento, trípode, micrófono, 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 habiendo soluciones accesibles o de bajo presupuesto. El precio de un sonómetro profesional de clase 1, ronda entre 3 y cinco 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[eI14]. 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 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: Tabla usuarios de teléfonos móviles inteligentes [eI14]

Dentro del segmento de usuarios de teléfonos móviles inteligentes, existen varios sistemas operativos móviles, cada cual provee su propia plataforma de desarollo, 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: Tabla usuarios de teléfonos móviles inteligentes [RL14]

Herramientas y metodologías utilizadas

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 1.1, una medición logarítmica de la presión sonora efectiva relativa a un nivel de referencia. Por definición:

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

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

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

El valor de p_0 más comunmente utilizado es $p_0 = 20\mu Pa(RMS)$, 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 determinada1.3. 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.

$$P_a = f \cdot v = A \cdot p \cdot u \cdot v \tag{1.3}$$

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

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

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.

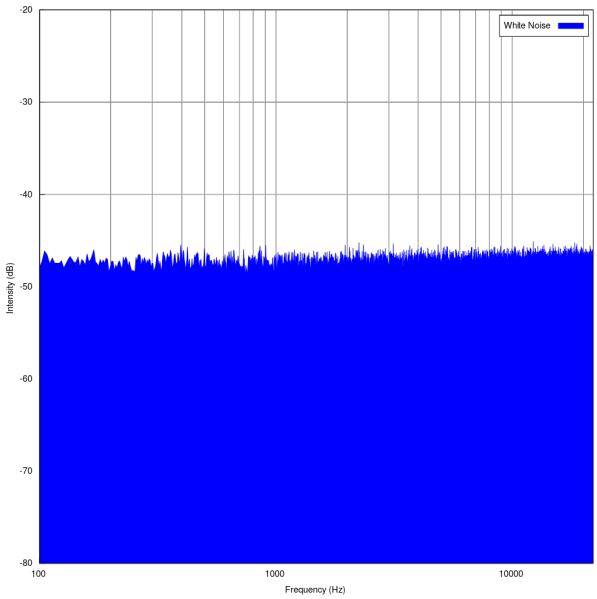

Figura 1.1: Espectro de un ruido blanco

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.

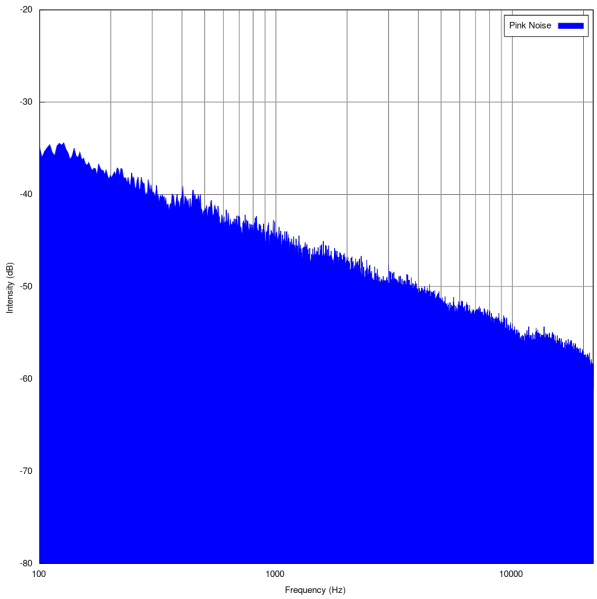

Figura 1.2: Espectro de un ruido rosa

1.1.4. 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.

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.

1.1.5. Modulación por impulsos codificados

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

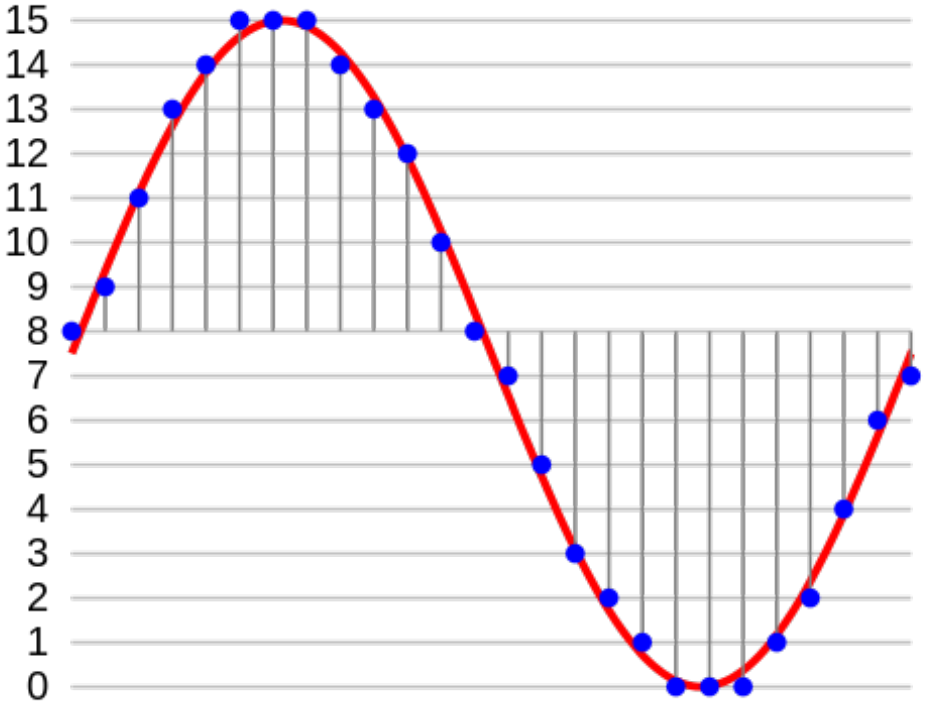

Figura 1.3: Muestreo y cuantización de una señal en PCM de 4 bits

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, en sus inicios concebido para dispositivos móviles con pantalla táctil, que ha evolucionado en una plataforma que actualmente también engloba relojes inteligentes, televisores, automóviles e incluso electrodomés-

ticos.

Está basado en el núcleo de Linux, sobre el que corre el entorno de ejecución propio de Android (ya sea Dalvik o ART), y este a su vez ejecuta el código de las aplicaciones, escritas mayoritariamente en Java, aunque se permiten extensiones en C/C++

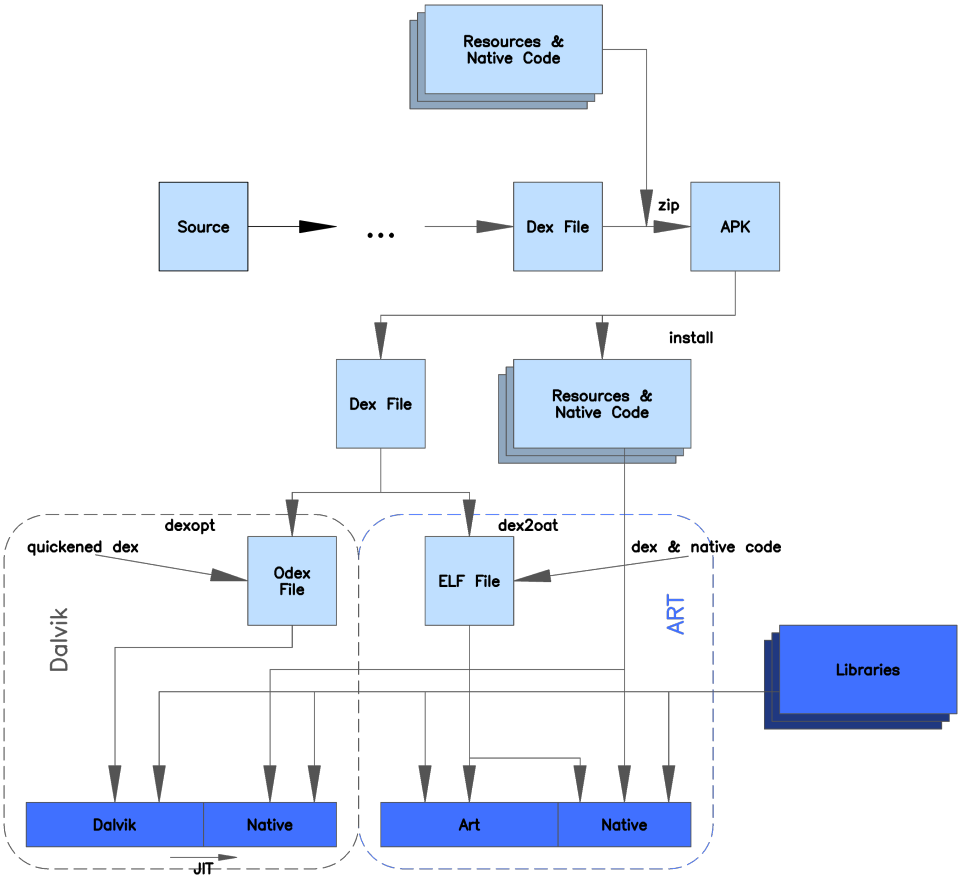

Figura 1.4: Diagrama de la estructura del sistema operativo Android

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.

Una parte importante de la interfaz general de Android son las notificaciones, presentes en la barra de estado

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 si 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, ya que un contexto permite acceso a recursos y clases específicos de la aplicación, y llamadas al sistema para operaciones a nivel de aplicación, tales como lanzar actividades, emitir mensajes de difusión o recibirlos.

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.

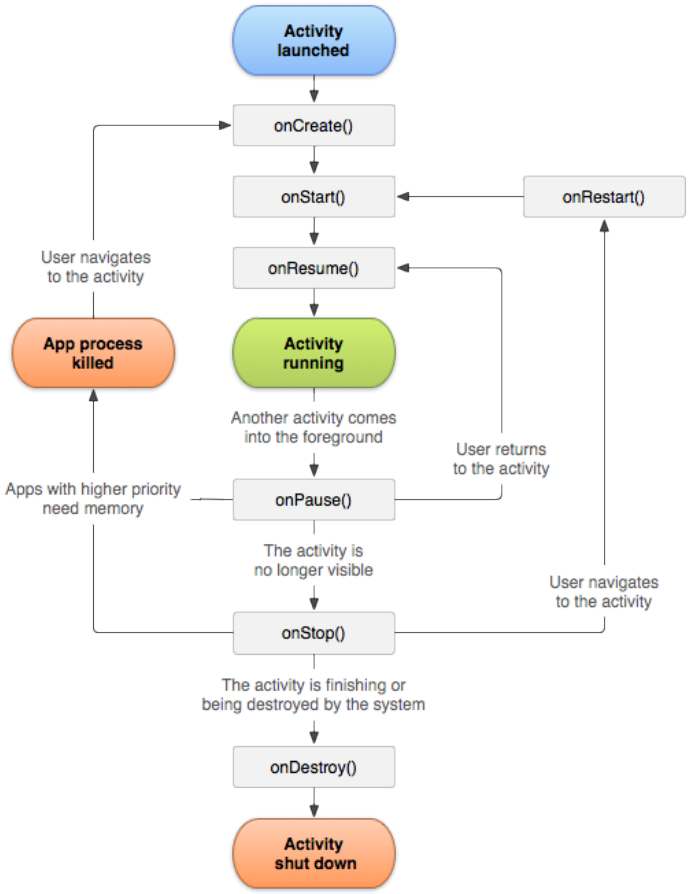

Figura 1.5: Diagrama del ciclo de vida de una actividad en Android. [Pro14b].

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 interacutar 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 destruído. 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 destruído.

Hay que 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.

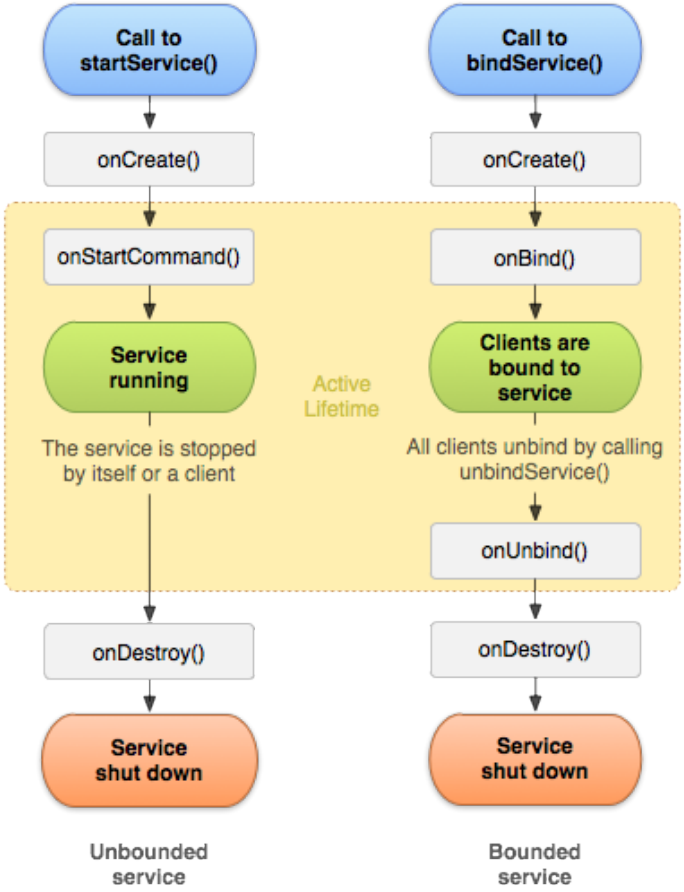

Figura 1.6: Diagrama del ciclo de vida de un servicio.androiddevguide

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 artilugios (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 aplicacones 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 porceso 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 qque debe de 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 de valores separados por comas, CSV por sus siglas en inglés. En este formato, como su nombre indica, cada valor de un registro está separdo 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.

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 órbida 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.

1.3. Otros

1.3.1. 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 1.7.

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.

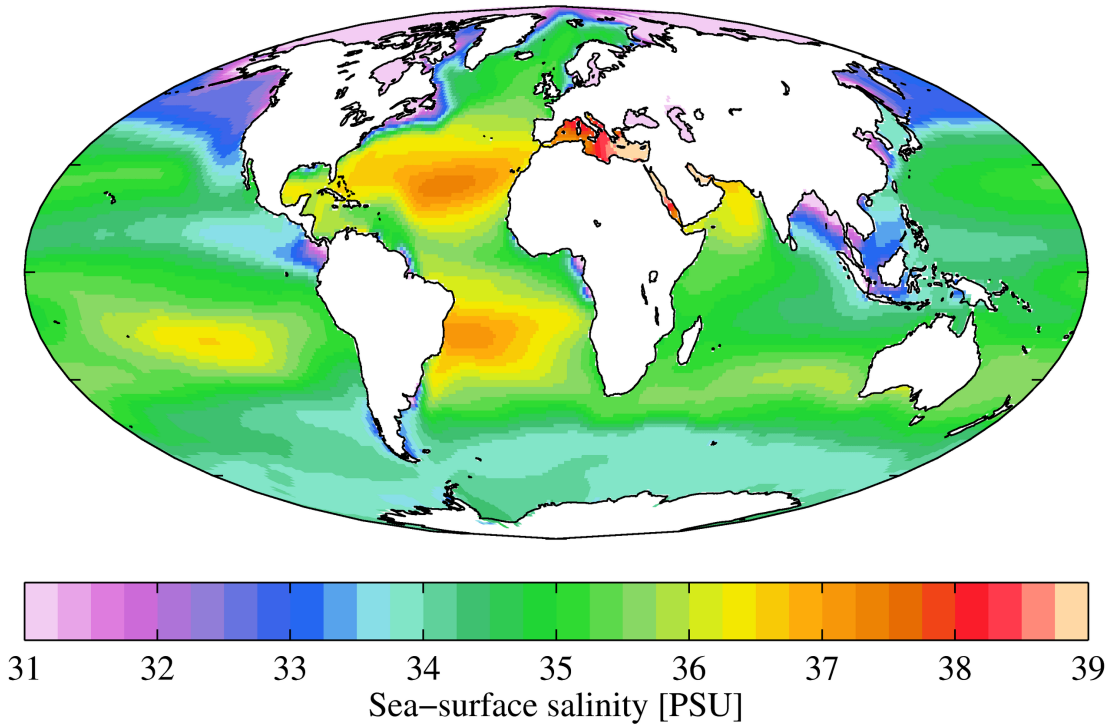

Figura 1.7: Mapa de calor mostrando el grado de salinidad de los océanos.[Wik15]

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 aplicaciónd 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 dificultoso 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 es la pantalla inicial, que muestra un mapa geográfico y nos permite superponer los datos previamente recabados

Grabar es la pantalla que interactúa con los sensores del dispositivo y comanda las acciones de grabación y guardado de archivos.

Sesiones presenta todos los datos recabados en pasadas grabaciones.

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

Figura 2.1: Primer borrador sobre 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 y el segundo 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 mostrando pérdida alguna de funcionalidad.

La versión mínima de Android requerida para el correcto funcionamiento de esta aplicación, es el nivel de Application Programming Interface, o interfaz de programación de aplicaciones (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 y marcado, que es la transmisión de voz vía redes celulares, y muestran 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, y no sólo trabajar correctamente con el dispositivo en el que fue desarrollada. Para ello, se habilita 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. Herramientas de desarrollo

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, tal como 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 en concreto de los dispositivos. Por ejemplo, es posible especificar un recurso para un idioma del télefono, tamaño de pantalla, ratio, orientación del dispositivo, versión de la API Android e incluso si el dispositivo se encuentra en modo nocturno o no.

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 2.2.

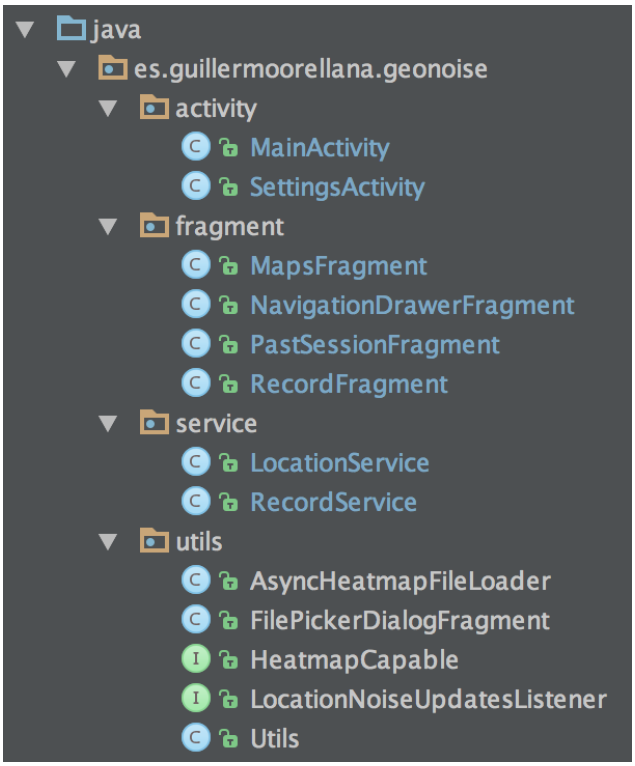

Figura 2.2: Organización del código fuente.

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, y las guías de diseño cambiadas hacia el llamado "Material Design" (diseño material), pero se ha desestimado seguirlas 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, la cual disminuye el acoplamiento en el código, aumenta la reusabilidad y la claridad del mismo, y por otra parte el entorno de desarrollo permite previsualizar el resultado de dichos archivos XML con bastante fidelidad.

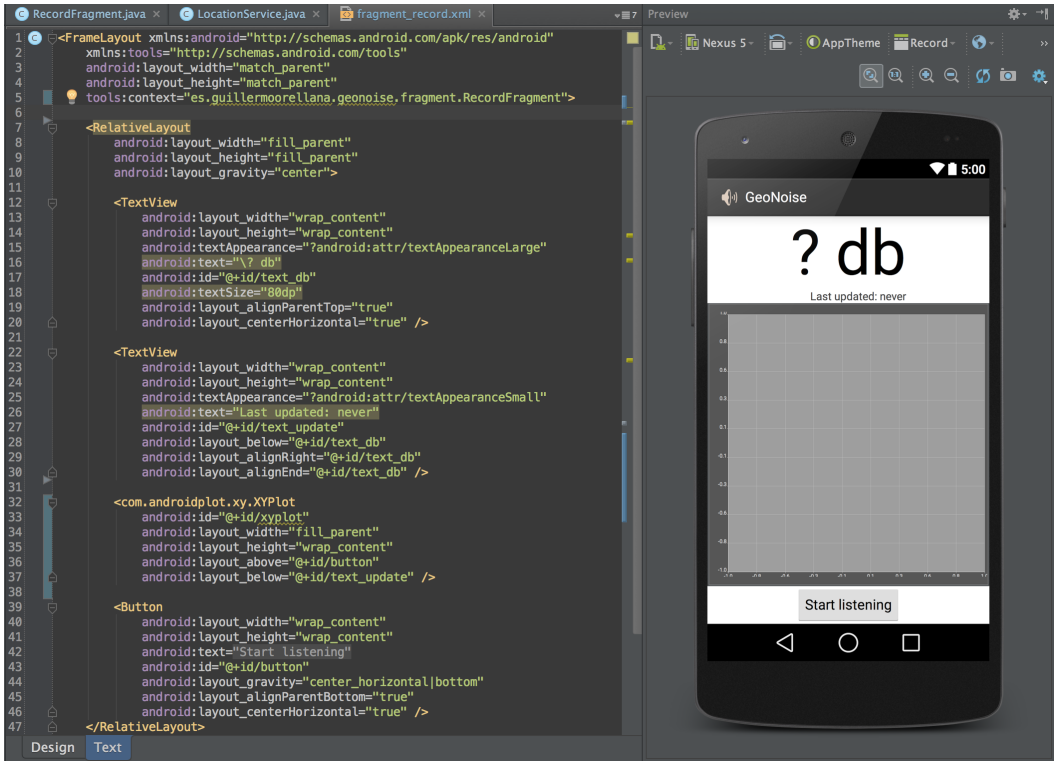

Figura 2.3: Editor de archivos de recurso XML con vista previa.

Además, en consonancia con las últimas recomendaciones de diseño de aplica-

ciones 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 dela interfaz gráfica de usuario. Como ejemplo, en la figura 2.3 se puede observar la edición de la interfaz gráfica de usuario de la actividad engargada de grabar los sonidos.

Si así se deseara, también cambe 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 2.4. De esta manera, se sacrifica precisión y control por comodidad y claridad.

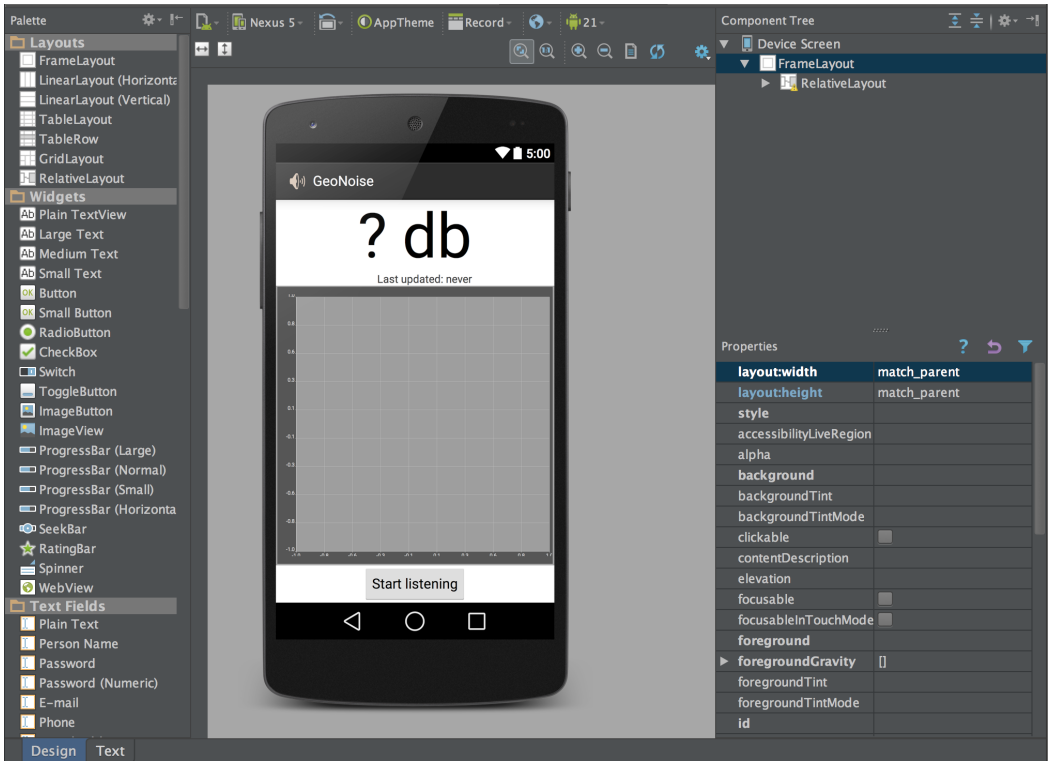

Figura 2.4: Edición completamente visual de la interfaz gráfica de usuario de la aplicación.

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 2.3, el código XML es el presente en el extracto 2.5. 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.

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="\? db"
        android:id="@+id/text_db"
        android:textSize="80dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Last updated: never"
        android:id="@+id/text_update"
        android:layout_below="@+id/text_db"
        android:layout_alignRight="@+id/text_db"
        android:layout_alignEnd="@+id/text_db" />
        [...]
</RelativeLayout>
```

Figura 2.5: Preparación de AudioRecord

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 región alguna de la pantalla mientras se encuentra sin desplegar.

Figura 2.6: Patrón de cajón de navegación implementado en la aplicación.

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 2.7 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

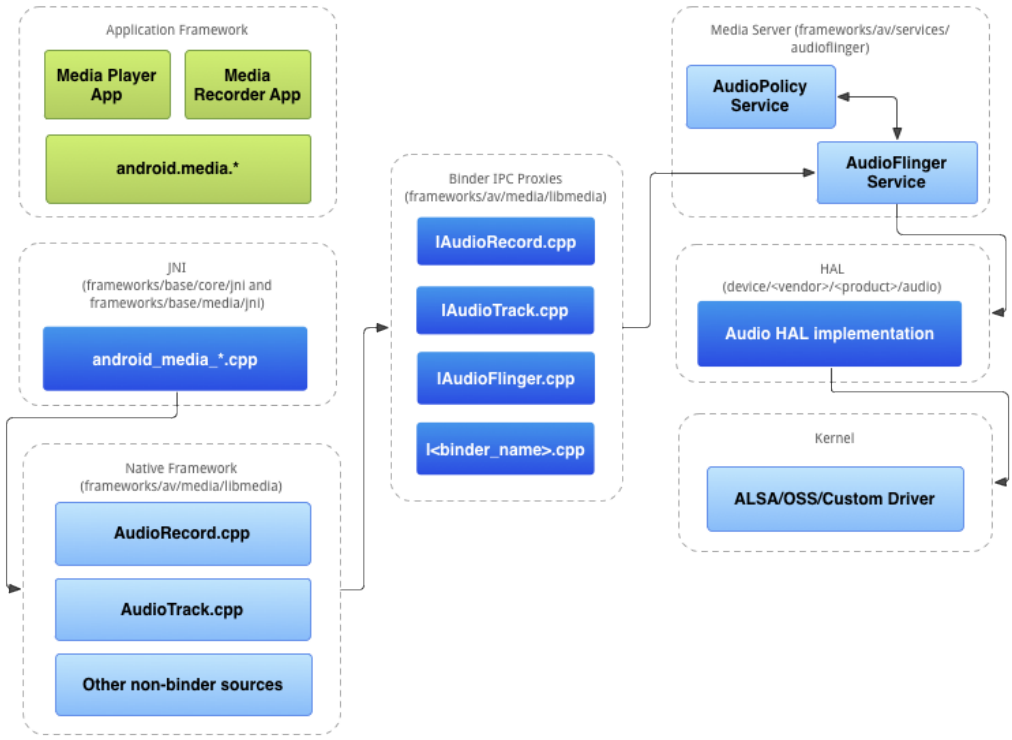

Figura 2.7: Arquitectura del audio en Android. Tomado de [Pro14a]

otras funciones de la aplicación, se ha optado por crear un servicio que corra en segundo plano y responda 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 extracto 2.8.

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 rellenar 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 2.9.

Las muestras están codificadas en formato PCM de 16 bits con signo. Por tanto,

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

Figura 2.8: Preparación de AudioRecord

el límite superior del valor de las muestras es $2^15 - 1 = 32767$. Los teléfonos móviles inteligentes suelen incorporar micrófonos de tecnología **MEMS!** (**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 $0{,}6325Pa$ en el micrófono. Asumiremos también que para el valor $P_0 = 0{,}00002Pa$ se obtiene una muestra PCM con valor 1. En consecuencia, a las muestras obtenidas les es aplicado un factor $\frac{32767}{0{,}6325Pa} = 51805{,}5336Pa^{-1}$.

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 2.9.

El siguiente paso es obtener el SPL correspondiente a la p_{rms} calculada, implementado en ka figura 2.10 según la ecuación 1.1. Se ha añadido una constante `splAdjustment`, la cual se utiliza para calibrar previsibles diferencias en el sistema

```
short[] buffer = new short[bufferSize];
int resultSize = -1;
resultSize = audio.read(buffer, 0, bufferSize);
double sum = 0;
// p_rms = [ 1/T ∫_0^T p^2(t)dt ]^1/2 = ( 1/T Σ_0^T p^2[n] )^1/2
for (int i = 0; i < resultSize; i++) {
    sum += buffer[i] / 51805.5336 * buffer[i] / 51805.5336;
}
double p_rms = Math.sqrt(sum / resultSize);
```

Figura 2.9: Captura de muestras con AudioRecord y cálculo de la p_{rms}

captador de audio de distintos teléfonos.

```
public double getDecibels(double level) {
    return Math.abs(20.0 * Math.log10(level /  0.00002)) +
     ↪ splAdjustment;
}
```

Figura 2.10: Cálculo del SPL.

2.2.5. Geolocalización

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

Se utiliza 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 hace 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 disponibilidad del resto de la aplicación de la manera mostrada en 2.12.

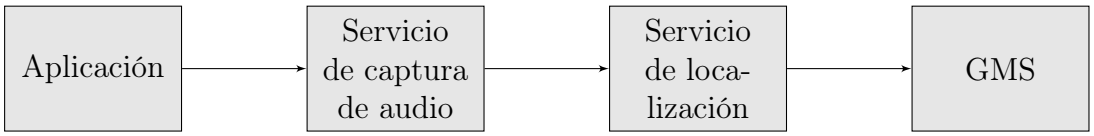

Figura 2.11: Diagrama de interacción de la aplicación y los servicios.

```
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();
```

Figura 2.12: Solicitud de conexión a los GMS.

Capítulo 3

Plan de pruebas y verificación

3.1. Funcionamiento de la aplicación

3.2. Precisión de las medidas

Para cercioranse 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.

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.

Aparato	**Sonómetro**	**Teléfono**
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

3.3. Realización de los objectivos

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. Podemos observar que dicho objetivo se cumple en la figura 3.1.

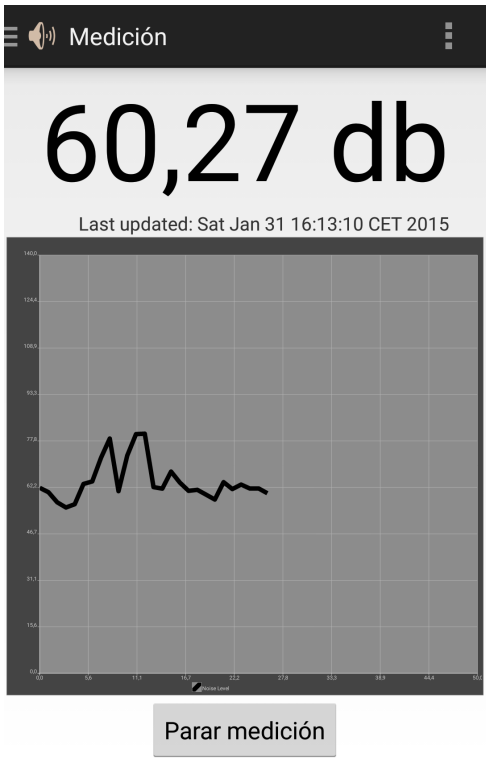

Figura 3.1: Captura de pantalla de la aplicación mostrando la estimación actual.

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 3.2. 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 3.3.

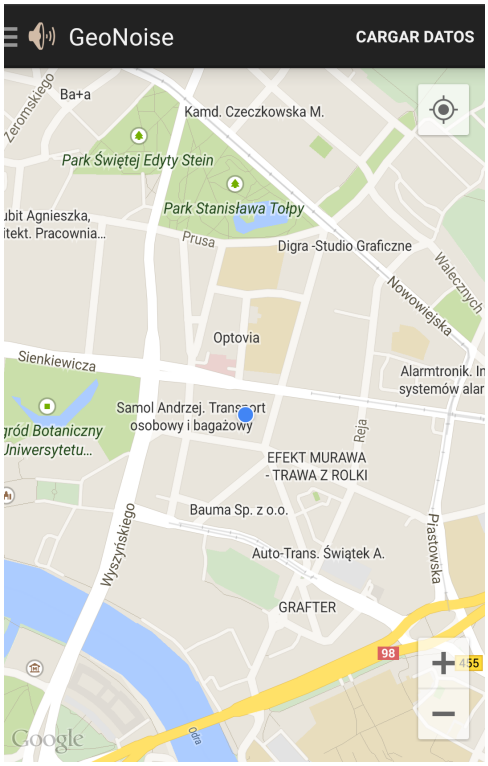

Figura 3.2: Captura de pantalla de la aplicación mostrando la localización actual del dispositivo móvil.

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 3.3.

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

```
"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 3.3: Extracto de un archivo CSV producido por la aplicación

pertinentes, tal y como se aprecia en la figura 3.4.

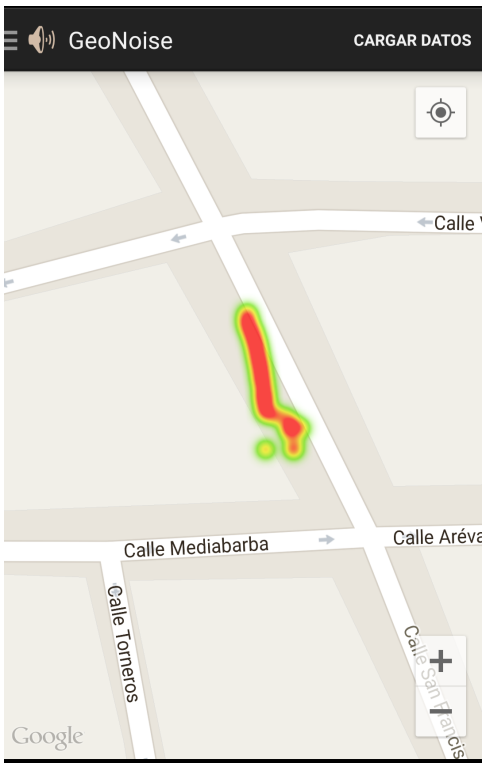

Figura 3.4: Captura de pantalla de la aplicación mostrando la representación de una sesión sencilla de grabación sobre un mapa.

Capítulo 4

Conclusiones y líneas de trabajo futuras

Son varias las mejoras directas que se pueden realizar en la aplicación.

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 la que, con la entrada de las medidas de múltiples usuarios, se puedan agregar los datos y, por ejemplo, representar los datos de toda una ciudad.

Exploración de alternativas en el formato de almacenamiento

En este proyecto se ha utilizado Comma Separated Values, o valores separados por coma (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.

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 encontremos 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.

Nombre del autor

5 de febrero de 2015

Bibliografía

[CPCA13] Eduardo Casilari Pérez and José Antonio Cortés Arrabal. *Breves notas de estilo para la redacción de Proyectos Fin de Carrera y Trabajos Fin de Grado*. ETSIT, Universidad de Málaga, 2013. `http://www.uma.es/media/files/Manual_de_Estilo_TFG_ETSIT.pdf`.

[eI14] eMarketer Inc. Smartphone users worldwide will total 1.75 billion in 2014. *eMarketer*, 2014.

[LG] LG. *Service Manual Nexus 5 LG D821*.

[min14] Minted: a highlighted source code for latex. `https://code.google.com/p/minted/`, 2014.

[Pro14a] Android Open Source Project. Android audio. `https://source.android.com/devices/audio/index.html`, 2014.

[Pro14b] Android Open Source Project. Android developers guide. `http://developer.android.com/`, 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.

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