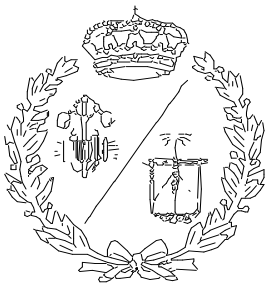
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Trabajo Fin de Grado

FONENDOSCOPIO ELECTRÓNICO: PROCESADO Y PRESENTACIÓN DE SEÑALES CARDÍACAS EN ENTORNO ANDROID

Electronic Phonendoscope: Processing and Display of Cardiac Signals in an Android Environment

Para acceder al Título de

GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: José María Asensio Delgado

Julio - 2019

Índice

Tabla de Contenido

Tabla de Figuras

Memoria

1 Introducción

Los fonendoscopios son herramientas utilizadas para la auscultación de los sonidos cardíacos y respiratorios. Este trabajo constituye una continuación de un proyecto en el que se plantea el desarrollo de un fonendoscopio electrónico. Se parte de un dispositivo ya creado desarrollado por el ingeniero César Campuzano en su Proyecto de Fin de Máster “FONENDOSCOPIO DIGITAL INALÁMBRICO CON PROCESADO DE SEÑAL SELECTIVO”.

El objetivo de este trabajo es realizar unas aplicaciones Android para el tratamiento y representación de las señales cardíacas. Para ello se ha comenzado por llevar a cabo un análisis de las opciones existentes para realizar un desarrollo Android y decidir cuál es la opción idónea para este tipo de proyecto. A continuación, se han realizado dos aplicaciones en ese entorno que realizan un primer procesado de datos, decidiendo el formato en el que se almacenará la información y mostrándola en forma de gráfica donde se representa la amplitud de la señal en cada instante, al mismo tiempo que se reproduce de forma sincrónica el audio original.

También se pretende introducir en el departamento la disciplina del desarrollo en entorno Android para futuros desarrollos.

Las ventajas que aporta este sistema de auscultación sobre el método tradicional donde el médico escucha directamente al paciente son varias. La capacidad de almacenar la información obtenida, transmitirla y compartirla libremente, permitiendo que se realicen diagnósticos a distancia.

La posible incorporación de análisis de datos que sean capaces de monitorizar al paciente de manera continuada y enviar una alerta cuando se produzcan actividades sospechosas, siendo capaz de ofrecer una grabación de los hechos. Esto podría ser utilizado para cuadros como arritmias cardiacas, donde el paciente podría dormir en su casa tranquilamente con un pequeño dispositivo pegado a su pecho y su móvil estaría grabando y enviando la información pertinente a la organización médica. Esto no solo es útil para el tratamiento, sino que podría ayudar

a la investigación, ya que se podría configurar para generar depósitos de datos de manera automática que podrían ser de gran interés ante futuras teorías o descubrimientos sobre enfermedades cardíacas.

Se podrá facilitar el aprendizaje en el centro médico. Los residentes tendrán una herramienta con la que podrán grabar los sonidos extraños y preguntar sobre ellos a los médicos más experimentados

Además de estas ventajas, hay otras como la posibilidad de aumentar el volumen de los sonidos cardiacos para facilitar su estudio.

2 Estudios Previos

El mercado nos ofrece diferentes softwares de desarrollo de aplicaciones móviles, tanto gratuitas como de pago. Para el desarrollo se precisa de una selección previa de la óptima o, al menos, de una adecuada para el trabajo que se va a realizar.

Se considerarán en profundidad tres opciones: “Android Studio”, “MIT App Inventor” y “AppSheet”.

2.1 Android Studio

Figura 1. Logo de Android Studio

Android Studio [ref. 1] es el entorno de desarrollo integral (IDE: Integral Development Environment) oficial de Android. Fue anunciado el 16 de mayo de 2013 en la “Google I/O” [ref. 2], un congreso organizado por la empresa tecnológica para presentar y discutir sobre los productos y aplicaciones de actualidad. El entonces nuevo sistema se plantea como un remplazo de Eclipse y aporta al desarrollador herramientas que facilitan la escritura de código y la detección de errores. Está basado en el software IntelliJ IDEA [ref.3], un IDE de Java desarrollado por JetBrains, una empresa de desarrollo de software con base en Praga, República Checa.

El software permite la introducción de código en Java, Kotlin y C++, siendo este último muy poco utilizado. Pero, ¿qué es Kotlin? Kotlin es un lenguaje desarrollado por JetBrains basado en Java y que puede ser compilado en JavaScript, su objetivo es ser una opción más conciso, seguro y robusto que pueda sustituir a Java. Google está apostando por la lenta suplantación de Java en Android Studio debido a estas ventajas.

Estructuralmente, el programa es muy completo y complejo y permite al desarrollador controlarlo todo. En lo que se refiere a sentido de posibilidades es el más completo, aunque esto lleva también a que se precise en muchas ocasiones de un mayor trabajo.

Esta herramienta es utilizada por los desarrolladores profesionales de aplicaciones móviles.

2.2 MIT App Inventor

Este entorno [ref. 4] fue desarrollado por Google Labs y el MIT CSAIL (Massachusetts Institute of Technology - Computer Science and Artificial Intelligence Laboratory). La versión actual, MIT App Inventor 2 salió en 2013, dos años después del cierre de Google Labs y desde entonces todo el desarrollo y mantenimiento ha sido llevado por el MIT.

Figura 2. Logo de MIT App Inventor

Está basado en programación con bloques, de manera que una persona alejada de la programación sea capaz de desarrollar sus propias aplicaciones. Por su formato accesible, es utilizado para introducir a niños y jóvenes en la estructura computacional y en el conocimiento de programación básica a partir de un entorno visual.

Se puede ver el aspecto general de esta aplicación web en la Figura 3.

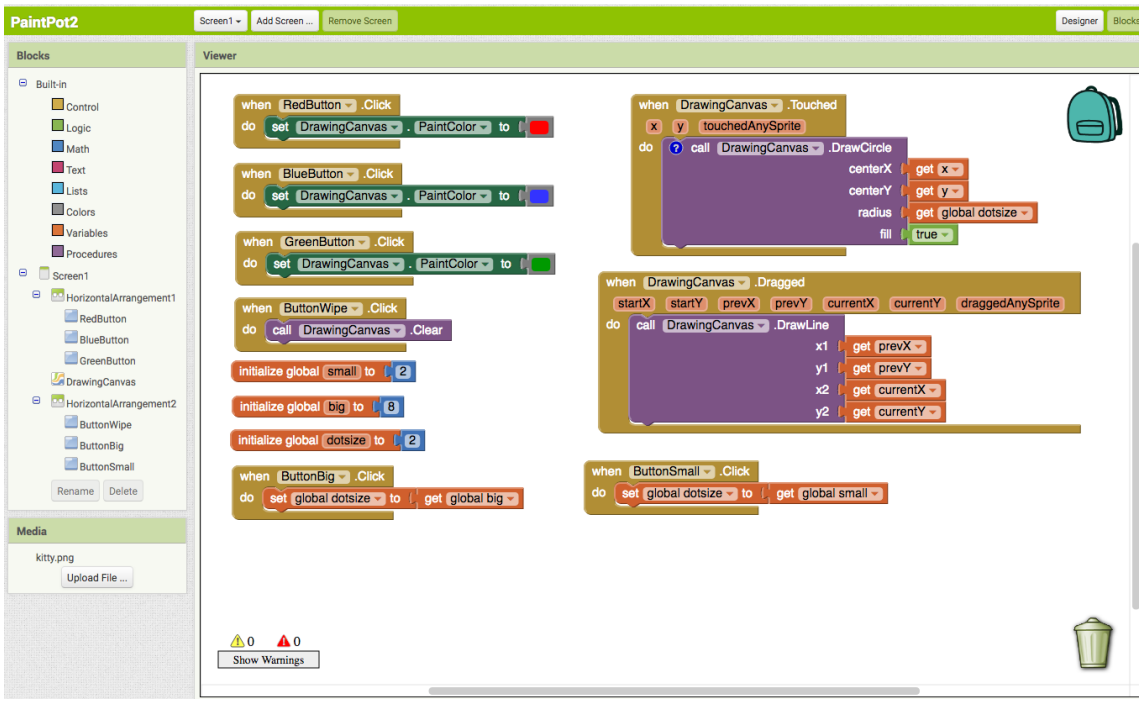


Figura 3. Página Web del MIT App Inspector

Como se puede ver se van juntando sistemas, cada uno con órdenes propias con el objetivo de crear una estructura global que realiza la tarea deseada.

2.3 AppSheet

Figura 4. Logo de AppSheet

Appsheet [ref. 7] es una aplicación creada por la empresa del mismo nombre. La empresa estadounidense de origen en Seattle fue creada en 2014 por Praveen Seshadri [ref. 8, 9 y 10], quien actualmente continúa en el proyecto a la vez que trabaja para Microsoft y colabora con la Universidad de Cornell como Assistant Professor en el Computer Science Department.

La compañía ofrece al cliente un entorno sencillo que opera sin la utilización de código en el que se puede trabajar conectado con diferentes plataformas como Google Drive, Office 365, Dropbox, SQL entre otras plataformas. De esta manera ofrece conexiones fáciles y automáticas entres los dispositivos que tengan sus programas y la red.

Incluye herramientas como localizador GPS apoyado en Google Maps, notificaciones e-mail, gráficas, grabador de firmas o escáner de códigos de barras. Lo cual lo hace un sistema muy completo para el posible usuario.

Sin embargo, difiere de las dos opciones anteriores en que es de pago, incluye una tarifa mensual de precios desde 5$ mensuales para un grupo de 10 usuarios.

El público que buscan alcanzar son empresas de tamaños pequeños o medianos que tengan trabajadores que se muevan a menudo y precisen de un sistema centralizado de logística para facilitar las operaciones.

2.4 Discusión

Podemos descartar de manera directa Appsheet como opción ya que, a pesar de ser una herramienta útil, está ideada para gestiones de bases de datos y de comunicaciones entre usuarios, mientras que nuestra tarea es obtener y procesar señales de audio.

Tenemos dos opciones viables: Android Studio y App Inventor. Para un usuario novato que no haya utilizado ninguna ni tenga conocimientos de programación en Java o Kotlin, la opción proporcionada por el MIT es más atractiva, debido a la

sencillez en su programación y el aporte de herramientas como programación en línea, lo que permite comenzar a hacer pruebas rápidamente. Por estas razones, opino que App Inventor es el mejor entorno para hacer una aplicación clara y definida de una manera rápida y eficaz.

No obstante, no me parece que App Inventor sea una opción recomendable para este trabajo por diversas razones que enumero a continuación.

Este trabajo se integra en la línea de investigación del Departamento TEISA dedicada a la creación de aplicaciones de Sistemas Automáticos para Biomedicina y es el primero de un nuevo proyecto en el que se utilizarán aplicaciones móviles.

Además, el trabajo se ha diseñado de forma que se pueda prolongar añadiendo en el futuro detectores o analizadores que, a partir del registro de los latidos cardiacos, sean capaces de detectar de manera autónoma la existencia de patologías cardiacas. Con estas aplicaciones se podría monitorizar a un paciente de manera autónoma y avisar al responsable médico cuando se detecte un comportamiento anómalo.

Si se utiliza Android Studio, estos analizadores se podrían desarrollar de manera totalmente independiente mediante rutinas escritas en Java e introducirlas de una manera relativamente sencilla a una aplicación móvil.

Por estas razones, se ha decidido optar finalmente por Android Studio como el entorno que se va a utilizar.

3 Android Studio

Como se ha dicho, Android Studio es un IDE que proporciona un entrono completo para el desarrollo de aplicaciones, permitiendo la máxima libertad al desarrollador.

La primera decisión que hay que tomar es en qué lenguajes se va a querer desarrollar el proyecto, en Java o en Kotlin. En un principio, intenté desarrollar todo este trabajo sobre Kotlin, pero un problema fundamental se interpuso en mi camino. Los tutoriales, guías y recursos en YouTube que se encuentran están en Java y los cursos más recomendados de Kotlin no suelen enseñar Android de raíz, sino que están orientados a público que sabe programar en Java y quiere emigrar o ampliar sus conocimientos al nuevo lenguaje.

Como el trabajo en Kotlin resultaba incómodo, la única opción coherente fue desarrollarlo todo en Java, por esta razón, todas las descripciones que contiene este documento son sobre Java y no se hace más referencia a Kotlin.

En este apartado, se van a explicar los componentes, herramientas y lenguajes que se necesitan conocer para utilizar Android Studio, así como algunos consejo y problemas encontrados a lo largo del proceso de desarrollo de una aplicación móvil.

3.1 Componentes de Android Studio

Vamos a analizar los diferentes componentes que hay en el entorno.

- *Activities* (actividades): en este trabajo es el elemento que más se utiliza. Es cada pantalla con interfaz de usuario. Con las actividades se configura la relación del usuario y el dispositivo con la aplicación. También se define en ellas la lógica de la aplicación. Pueden realizarse varias actividades al mismo tiempo que trabajen a la vez en la misma pantalla. Esta última afirmación puede resultar confusa, por tanto, introduciré un par ejemplos de aplicaciones en las que se utilizan varias actividades de manera simultánea para explicarlo:
    - Anuncios en una aplicación. Un caso habitual en aplicaciones gratuitas es la existencia de anuncios en la pantalla. Estos anuncios no forman

parte de la lógica de la aplicación, sino que son un sistema aparte que es invocado (ya sea de una memoria interna o de la web) y son mostrados al usuario.

  - Reproductor de vídeo. Como ejemplo YouTube, Twitch, Mixer o la que se desee. En la aplicación, se puede reproducir un vídeo mientras se navega por la variedad de contenido que ofrece la plataforma. Ahí se pueden apreciar claramente al menos dos actividades fundamentales, la primera la reproducción del vídeo y la segunda el navegador.
  - PlayStore. Este ejemplo es menos claro, pero me parece que puede ser útil para comprender el funcionamiento de las grandes aplicaciones. En PlayStore (o en cualquier plataforma de distribución análoga de otra compañía) suele haber una herramienta de búsqueda en la parte superior. Ese objeto trabaja de manera independiente a las cajas de recomendaciones que te va ofreciendo según la zona en la que estés o donde trabajes. A ambos objetos o actividades accede una actividad principal que va obteniendo estos recursos e introduciéndolos en la forma que tiene la aplicación para mostrarlos al usuario. Al final, en aplicaciones de este tipo, lo que se hace es ir "conglomerando" las distintas herramientas que se han desarrollado (actividades) para que el usuario pueda usarlas. El producto final de este trabajo es una actividad que puede ser modificada para trabajar con otras al mismo tiempo (analizadores) para que aporte más utilidad al usuario final.

- *Services* (servicios): son los componentes que se ejecutan en segundo plano sin la necesidad de que el usuario esté dentro del entorno de la aplicación. Los ejemplos más claros son la reproducción de música mientras se está en otra aplicación, la descarga de aplicaciones que realiza por ejemplo PlayStore o el rastreo de posición.
- *Broadcast receivers* (receptores de mensajes): permiten elevar o hacer aflorar eventos fuera del flujo normal de la aplicación. Estos eventos pueden aparecer incluso con la aplicación apagada. Son las notificaciones del móvil: batería baja, alarmas temporales, avisos de descargas disponibles…

- *Content providers* (proveedores de contenidos): es el componente que administra los datos de la aplicación. Almacena la información en una base de datos, en la web o en alguna carpeta del sistema. No se desarrollan totalmente en este trabajo, pero sí se recogen explicaciones de su funcionamiento.

3.2 Descargar Android Studio

Hay muchas páginas y guías donde se explica cómo realizar la descarga de Android Studio. Aun así, hemos querido recoger aquí una serie de informaciones para contar algunos aspectos que no se suelen mencionar como determinados fallos o recomendaciones de usuario, o consideraciones sobre los requerimientos.

Empezaremos este apartado analizando los requerimientos. Los requerimientos que nos pide para la instalación de Android Studio (para Windows) son [ref. 1]:

- Microsoft® Windows® 7/8/10 (32- or 64-bit)
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

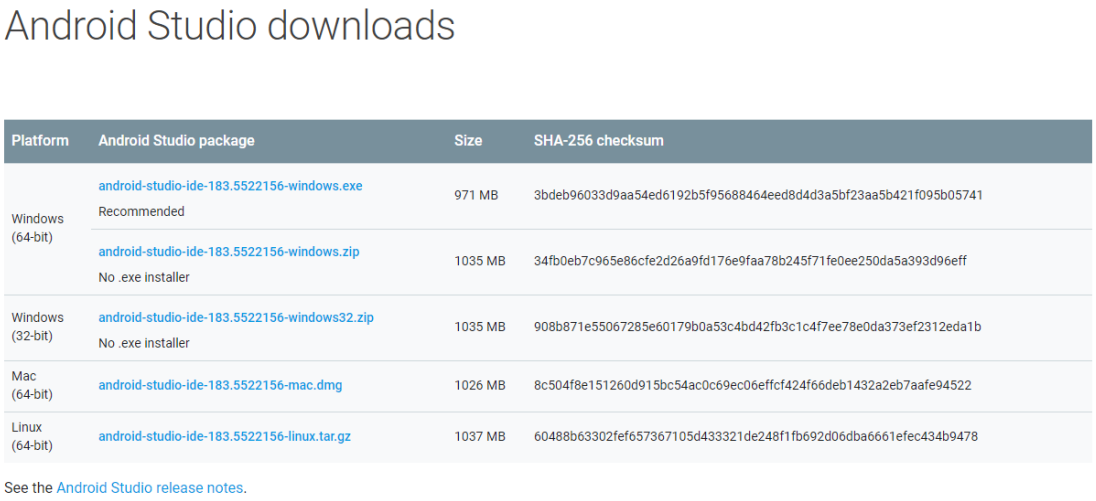
Los requisitos para otras plataformas (Linux, Mac y Chrome OS) son semejantes, siendo ligeramente más exigentes para los dispositivos de Chrome OS.

El dispositivo con el que se ha desarrollado este proyecto es un ordenador portátil HP Pavilion Convertible con las características:

- Procesador: Intel® Celeron® CPU N3050 @ 1.60 GHz
- RAM: 4,00 GB
- Tipo de OS: Microsoft® Windows® 10 de 64 bits, procesador x64
- Resolución: 1366 x 768

Como se puede ver, llega al mínimo de requerimientos, siendo ligeramente inferior la resolución de pantalla. La resolución mínima de pantalla es un requisito necesario para el dispositivo virtual y no para el resto de aplicaciones. Como se trabaja con la RAM mínima y no la recomendada, no es posible utilizar esta herramienta y la resolución de pantalla es suficiente para el resto de componentes.

Los archivos de descarga se encuentran en la plataforma de Android para desarrolladores [ref. 1], hacia el final de la página (ver Figura 5) En la misma plataforma existe una guía de descarga [ref. 12], sin embargo, en ella se incluye solo una pequeña descripción para apuntar los posibles errores encontrados y cómo solucionarlos.



Android Studio downloads

Platform	Android Studio package	Size	SHA-256 checksum
Windows (64-bit)	android-studio-ide-183.5522156-windows.exe Recommended	971 MB	3bdeb96033d9aa54ed6192b5f95688464eed8d4d3a5bf23aa5b421f095b05741
	android-studio-ide-183.5522156-windows.zip No .exe installer	1035 MB	34fb0eb7c965e86cfe2d26a9fd176e9faa78b245f71fe0ee250da5a393d96eff
Windows (32-bit)	android-studio-ide-183.5522156-windows32.zip No .exe installer	1035 MB	908b871e55067285e60179b0a53c4bd42fb3c1c4f7ee78e0da373ef2312eda1b
Mac (64-bit)	android-studio-ide-183.5522156-mac.dmg	1026 MB	8c504f8e151260d915bc54ac0c69ec06effcf424f66deb1432a2eb7aafe94522
Linux (64-bit)	android-studio-ide-183.5522156-linux.tar.gz	1037 MB	60488b63302fef657367105d433321de248f1fb692d06dba6661efec434b9478

See the Android Studio release notes.

Figura 5. Posibilidades de descarga.

Tras descargar y ejecutar, el programa instalador nos ofrece la posibilidad de abrir el programa. El programa debería estar ya preparado para funcionar, pero en algunos casos los kits de desarrollador pueden dar fallos.

Los kits de desarrollador (DK “Developer Kit” o SDK por “Software Developer Kit”) son conjuntos de herramientas destinadas a la creación de aplicaciones. Se encargan de la gestión de bibliotecas y paquetes, aportan en los sistemas de trabajo y se involucran en la transmisión del software al hardware que se vaya a utilizar.

Básicamente, en Android Studio se utilizan dos:

1. Android SDK. El kit de desarrollador de Android.
2. JDK. El kit de desarrollador Java.

El kit de Java es necesario, aunque se vaya a trabajar en Kotlin, debido a que comparten sistemas y llamadas.

Un problema que puede haber al usar el kit de Java es que las herramientas del desarrollador no estén disponibles en el sistema. En ese caso hay que acudir a las variables del entorno y añadir a la variable JAVA_HOME (si no existe se debe crear) la ruta hasta las herramientas (seguramente en archivos del sistema).

El Android SDK y otros recursos de Android Studio pueden dar problemas porque tiene actualizaciones frecuentes. En algunas temporadas, pide más de una actualización a la semana, aunque sean de pequeño tamaño. En mi caso, cuando inicié el programa por primera vez tenía errores para operar con las herramientas en su puesta a punto. Para solucionarlo, acabe descargándolas como un paquete aparte (las herramientas de Android pueden ser utilizadas en otros sistemas además de Android Studio) e indicando en el entorno que accediese a su archivo. Tras ello, en las siguientes actualizaciones volvió a dar problemas de nuevo. Al final, lo conseguí resolver abriendo el programa como administrador; de esta manera, cuando se producen actualizaciones no tiene problemas con el trabajo que hace sobre las carpetas.

Esta táctica la he acabado manteniendo en todo momento, debido a que no es posible saber cuándo va a haber una actualización, por lo que recomiendo que siempre que se trabaje con este programa se EJECUTE COMO ADMINISTRADOR.

3.3 Aplicación y proyectos

Las referencias suelen aludir a este apartado como “Build your first app” o “Prepara tu primera aplicación”, pero se va a realizar un enfoque algo distinto.

La pantalla inicial (Figura 6) nos ofrece las aplicaciones recientes a la izquierda y un menú de opciones en el resto[1].

[1] La pantalla presentada corresponde a la versión 3.4.1, en versiones anteriores era distinta y puede volver a cambiar

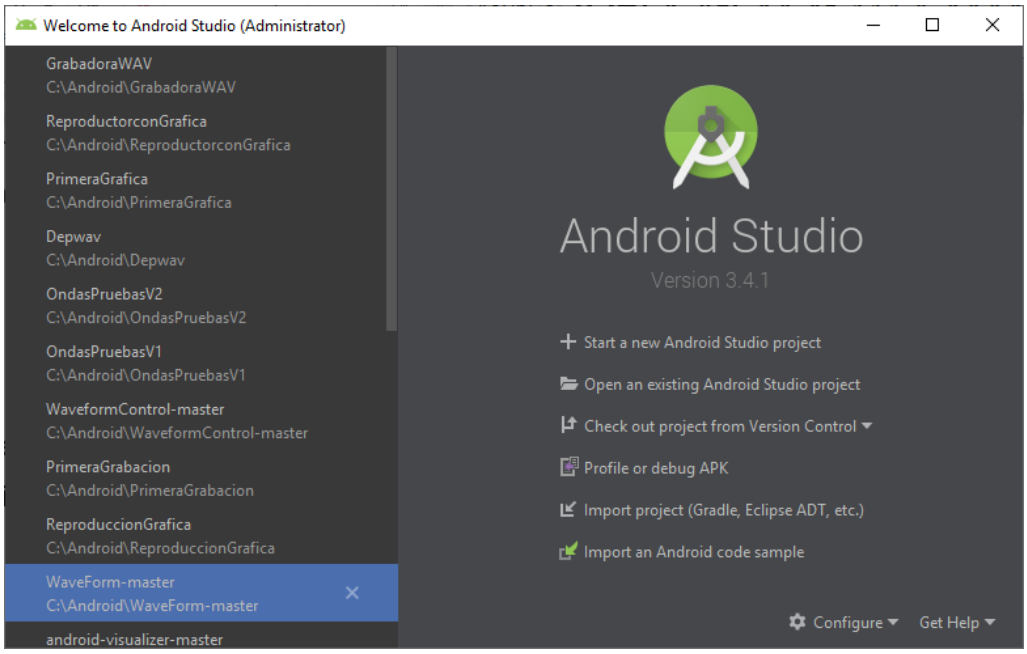


Figura 6. Pantalla inicial de Android Studio.

Cuando se selecciona crear un nuevo proyecto se muestran diferentes opciones como se ve en la Figura 7.

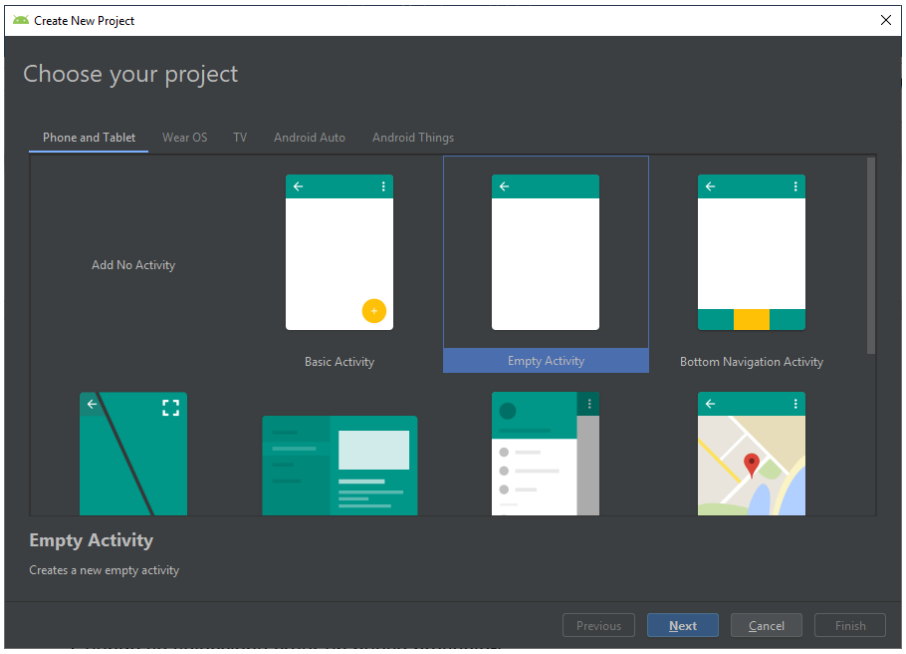


Figura 7. Opciones de nuevo Proyecto.

Se puede elegir el tipo de dispositivo sobre el que se va a trabajar y un modelo previo sobre el que escribir. La mayor parte de las fuentes recomiendan usar la actividad vacía (“Empty Activity") y, tras probar varias de las opciones, me parece una recomendación acertada. Es cierto que, en determinadas ocasiones, puede resultar más útil la actividad básica, que aporta un botón flotante, o la de panel de

navegación inferior, aunque ambos elementos pueden ser generados por el desarrollador.

Sin embargo, puede ser un buen planteamiento para un nuevo usuario que quiere usar ese tipo de elementos generar estas actividades para poder analizar cómo es el código para tener un buen ejemplo.

La ventana de configuración de proyecto viene a continuación (Figura 8. Además del nombre y la localización, se pueden encontrar tres campos importantes: el nombre del paquete, el lenguaje de programación y el nivel de API mínimo.

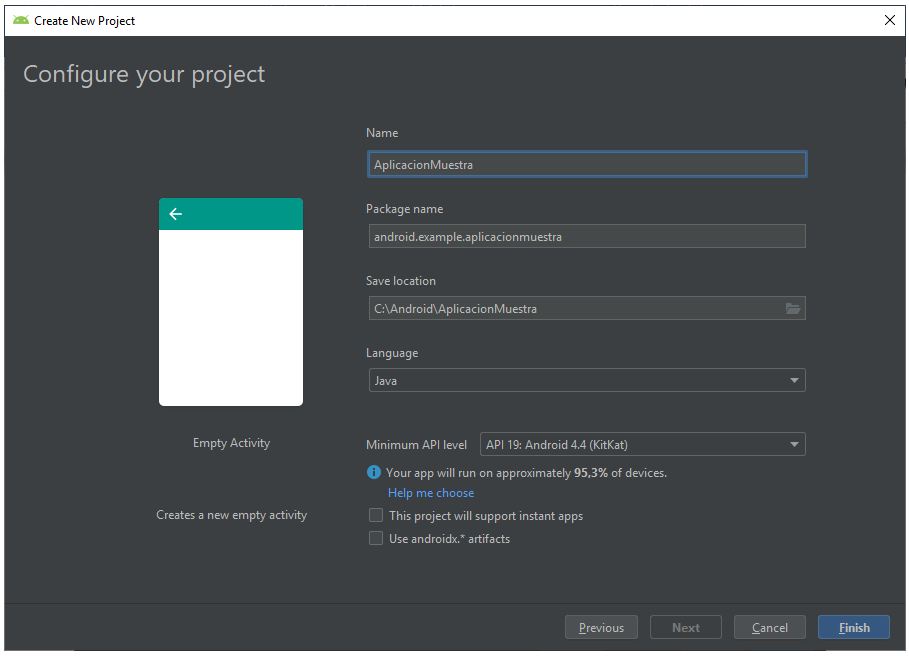


Figura 8. Pantalla de configuración del proyecto

El nombre del paquete es importante ya que hace referencia a la compañía a la que pertenece la aplicación. Además, es importante que sea único para que no pueda producirse la coincidencia del nombre de la aplicación con otro desarrollo. Los desarrolladores independientes suelen escribir ahí el nombre de su página web y otros organismos suelen poner su nombre o una forma abreviada.

El lenguaje puede ser Java o Kotlin. Posteriormente se puede cambiar durante el desarrollo del proyecto, pero es poco recomendable.

La API mínima requerida. API son las siglas de Application Programing Interface. Es la versión del móvil que se está utilizando. Para la elección de la API se atiende a dos factores: el primero es el alcance que puede tener la aplicación; los móviles no pueden, en general, trabajar con una versión mayor de la que tienen instaladas. El

segundo factor son las herramientas que vamos a utilizar. Si queremos usar implementaciones más nuevas tendremos poner una API mínima requerida mayor, pero eso limitará el número de dispositivos que podrán usar el producto.

3.4 Elementos del proyecto

Al abrir el nuevo proyecto, se nos abre el entorno como el de la Figura 9.

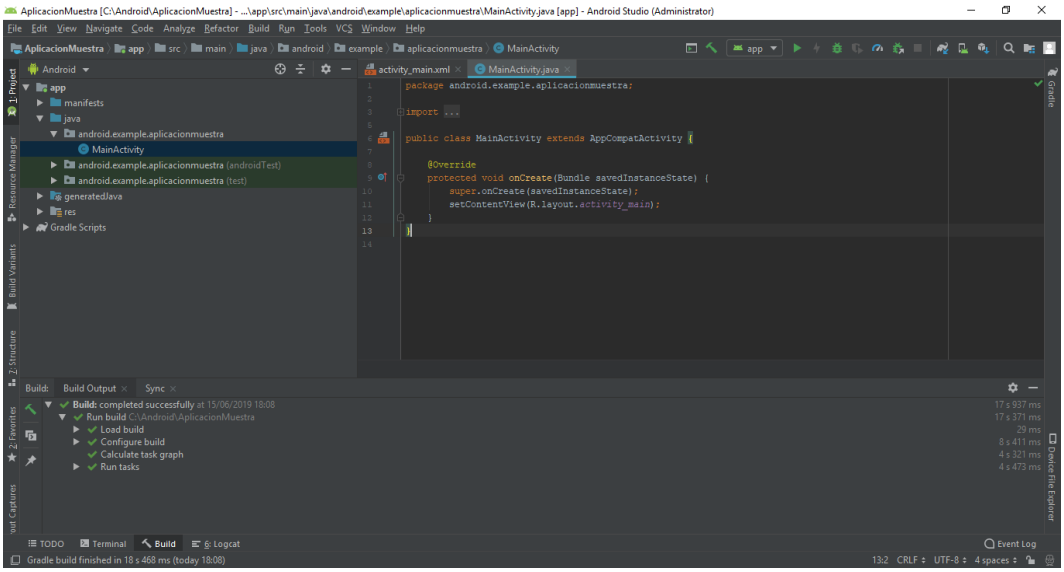


Figura 9. Entorno de Android Studio al abrir un nuevo proyecto con Actividad Vacía.

Tras la construcción inicial, se abren por omisión dos archivos: '*MainActivity.java*' y '*activity_main.xml*'.

En la barra de la izquierda, se encuentra la navegación (Figura 10). Tiene diferentes pestañas y, por omisión, nos muestra la de Android, donde se ven los recursos.

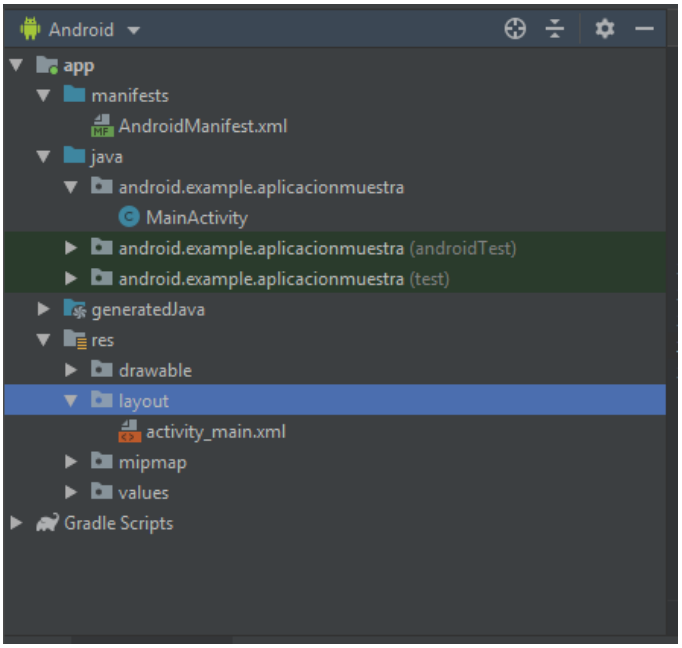


Figura 10. Barra de navegación lateral (1)

Hay cuatro carpetas:

1. '*manifests*': incluye la llamada de la aplicación. En el documento que contiene ('*AndroidManifest.xml*') se

recoge el nombre y directivas principales de la aplicación, incluyendo el icono de la aplicación, su orientación o las actividades con las que trabaja.

2. '*java*': además de los test que ejecuta para comprobar el funcionamiento correcto de la aplicación, contiene la carpeta donde se guarda la actividad principal y en la que se van guardando el resto de actividades o recursos java en caso de que sean utilizados. '*MainActivity*' o '*MainActivity.java*' es el archivo fundamental de funcionamiento de la aplicación.
3. '*generatedJava*': incluye configuraciones de la aplicación. No es necesario trabajar aquí en condiciones normales.
4. '*res*'. Incluye los recursos de la aplicación, incluyendo la distribución o '*layout*' con el archivo '*activity_main.xml*'. En esta carpeta suelen guardarse todos los archivos multimedia internos de la aplicación, desde sonidos de fondo, imágenes, logos, dibujos, etc. Cabe aclarar que pueden guardarse en otros directorios, pero se suele hacer aquí por sencillez, claridad y uniformidad.

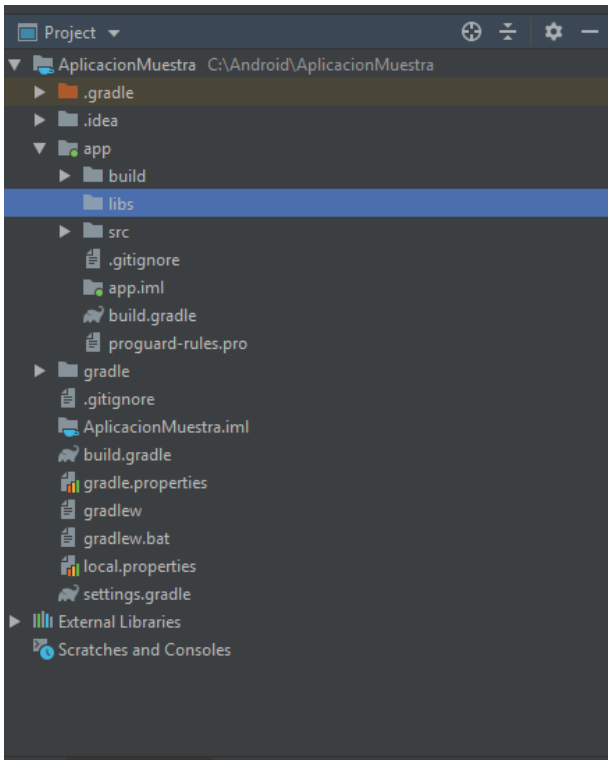


Figura 11. Barra de navegación lateral (2)

Los archivos iniciales del proyecto están configurados para que muestren un texto en el centro de la pantalla que dice "Hello World!".

Además de esta pestaña, se suele utilizar la de *project*. La pestaña *project* contiene la división en carpetas real del sistema. Una de las carpetas más interesantes es la carpeta '*lib*' dentro de '*app*'. Es la carpeta en la que se guardan las bibliotecas utilizadas en la aplicación.

Para acceder a ella se puede buscar en las carpetas o pulsar botón derecho y seleccionar '*Copy Path*'.

Una vez introducida la biblioteca en la carpeta, hay que introducirla '*as library*' para que el programa sepa a dónde acudir.

Se puede guardar la biblioteca en otra carpeta distinta a la mostrada, pero por orden y por facilitar el trabajo a personas que tengan que modificar el programa en el futuro, toda la comunidad lo hace así.

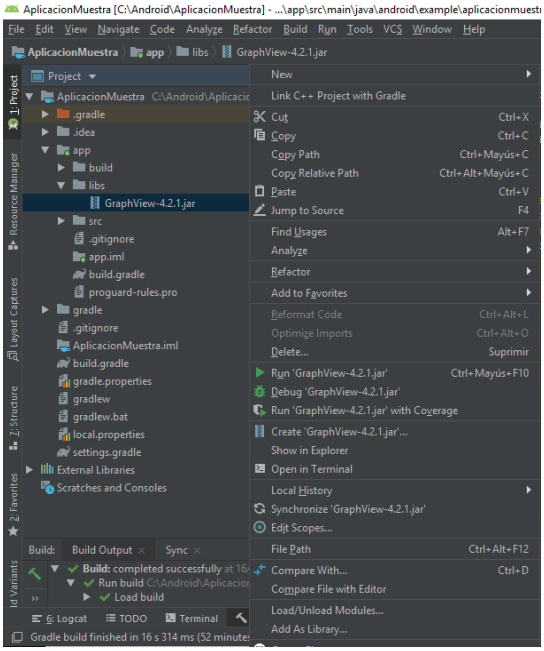


Figura 12. Exportar bibliotecas

3.5 Ejecutar la aplicación

Una vez que tenemos una aplicación completa, querremos ejecutarla para ver su funcionamiento. Existen dos opciones para ello, utilizar un emulador o ejecutarlo en un dispositivo.

Para lanzar la orden de ejecución o de depuración se usan los botones en la parte superior derecha (Figura 9). Cuando se hace eso se abre la pantalla que se muestra a continuación.

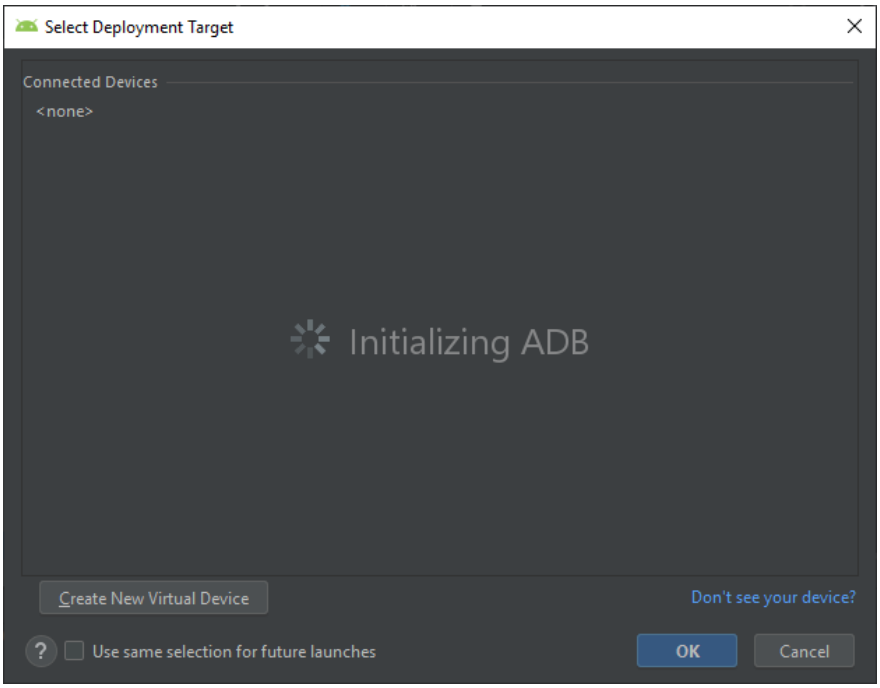


Figura 13. Selección de dispositivo para ejecución

ADB significa “Android Debug Bridge”. Es la herramienta con la que se establece la comunicación con los dispositivos. En la parte inferior se encuentra el botón con el que se genera el emulador. Una vez que se clica se seleccionan las características de dispositivo y versión deseadas para crearlo. Se supone que es sencillo y no da problemas, yo no puedo corroborarlo porque no he podido crearla por limitaciones del dispositivo.

Para conectarlo con un dispositivo móvil es necesario habilitar en dicho dispositivo las opciones de desarrollador y el sistema para modificaciones. Los pasos son:

1. Abrir ajustes.
2. Acceder a “Sobre el dispositivo” o “Información del teléfono” abajo.
3. Pulsar 7 veces sobre el “Número de compilación”.
4. Regresar a la pantalla anterior y entrar en “Opciones de desarrollo” que acaba de aparecer.
5. Bajar y activar “Depuración USB”.

Además, es necesario instalar el controlador del móvil utilizado, se hace sin complicaciones cuando se conecta el móvil al ordenador.

Aunque se decida hacer uso de un emulador, el desarrollador tiene que probar la aplicación en un dispositivo móvil para comprobar su funcionamiento final antes de ser utilizada o distribuida. Si es posible se recomienda probar la aplicación en dispositivos distintos, sobre todo para comprobar que no haya problemas de tamaño y resolución de pantalla. Sin embargo, en la práctica es algo inalcanzable a no ser que se trabaje en una empresa dedicada al desarrollo de aplicaciones móviles.

3.6 Diseño de interfaz de usuario

Para la interfaz de usuario se ha usado xml. El lenguaje *xml* (eXtensible Markup Language) es un lenguaje marcado muy semejante al *html*. Es de uso general y, en el entorno de Android Studio, se utiliza para la denominación de la aplicación (‘*AndroidManifest.xml*’) y para el diseño gráfico de la interfaz del usuario (‘*activity_main.xml*’). En este apartado se va a centrar la atención en la interfaz de usuario y los objetos más simples que existen en el archivo *.xml* correspondiente.

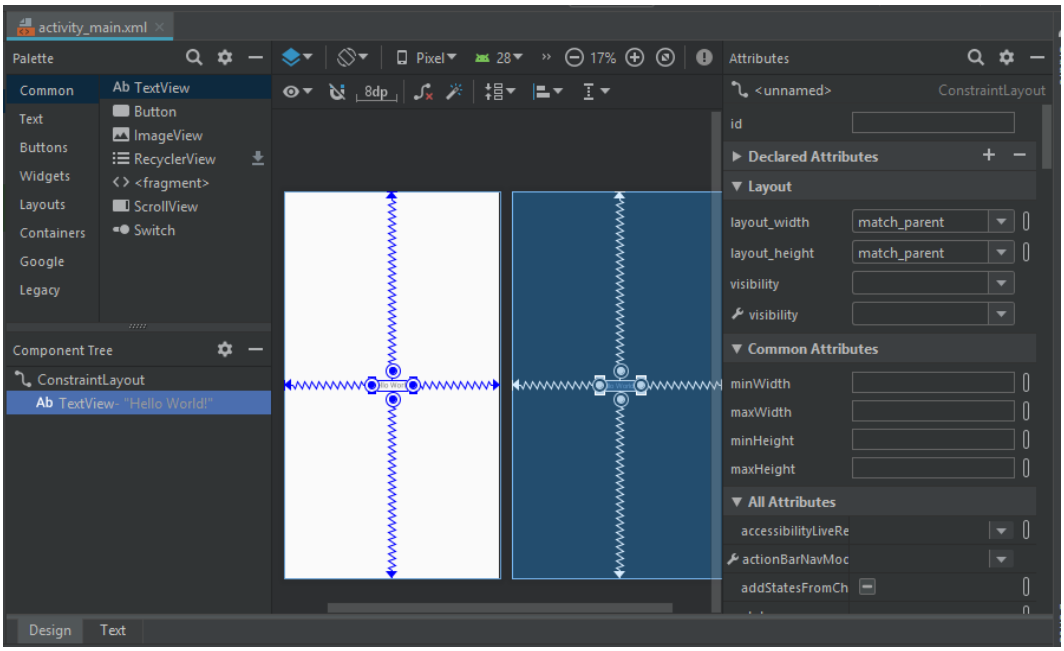


Figura 14. Diseño de interfaz de usuario. Design

La interfaz de usuario se puede configurar en una pantalla gráfica de diseño como la que se puede ver en la Figura 14, o escribiendo en código como se ve en la Figura 15.

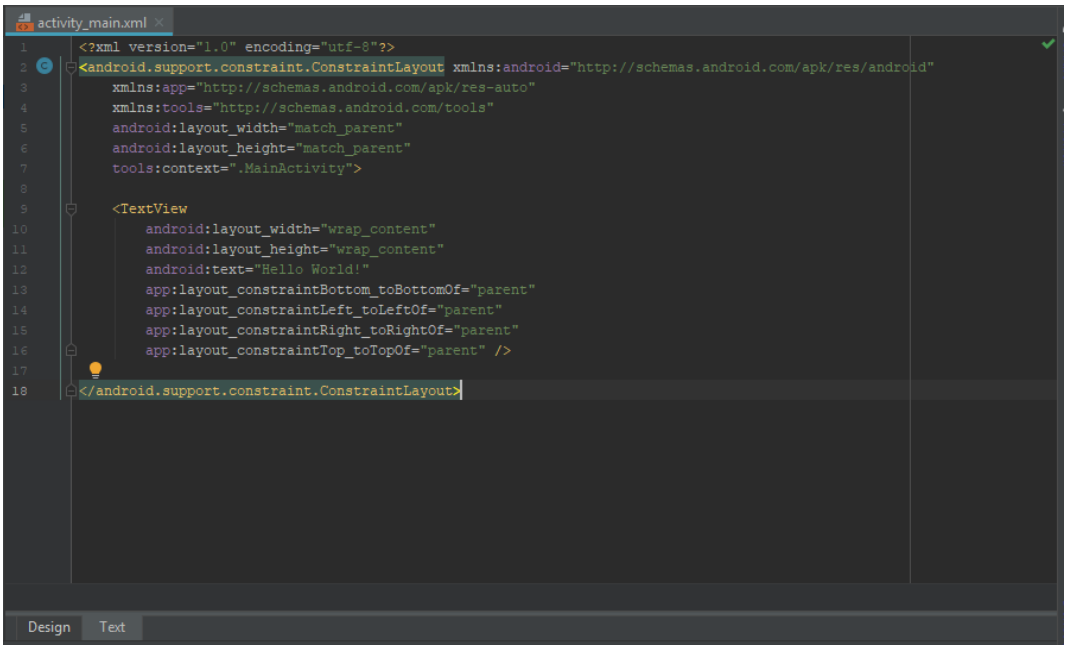


Figura 15. Diseño de interfaz de usuario. Text

En el modo '*Text*' está la opción de abrir una pantalla de vista preliminar para ver los cambios que vas haciendo.

El elemento fundamental de la interfaz de usuario son los '*layout*'. Estos objetos son las distribuciones sobre los que se asientan el resto de los objetos. El que se utiliza por omisión en el proyecto es el '*ConstraintLayout*'. Se basa en restricciones espaciales entre objetos. Personalmente, prefiero utilizar otros dos tipos de distribuciones para el desarrollo de aplicaciones, debido a que son más simples e igualmente de potentes. Los dos '*layouts*' que voy a describir son el '*LinearLayout*' y el '*RelativeLayout*'. Se comenzará la explicación con el '*LinearLayout*' y en este entorno se explicarán las relaciones y los componentes básicos mediante ejemplos.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textColor="@android:color/holo_orange_dark"
        android:textSize="24dp"
        android:background="@android:color/holo_blue_bright"
        />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón"/>

</LinearLayout>
```

La distribución lineal como se ve significa que los objetos van ocupando posiciones consecutivas unos detrás de otros. A cada objeto se le da un tamaño con '*layout_width*' y '*layout_height*'. Los valores más normales son '*match_parent*' que significa que el objeto se extiende por todo el espacio disponible del nivel superior. '*wrap_content*' significa que el objeto rodea el espacio de lo que tiene contenido. Véanse los dos objetos, tanto el botón como el texto (se ha puesto color de fondo al texto para ver su comportamiento. Además, se puede definir el tamaño como valor absoluto, es menos recomendable porque varía según el tamaño de la pantalla.

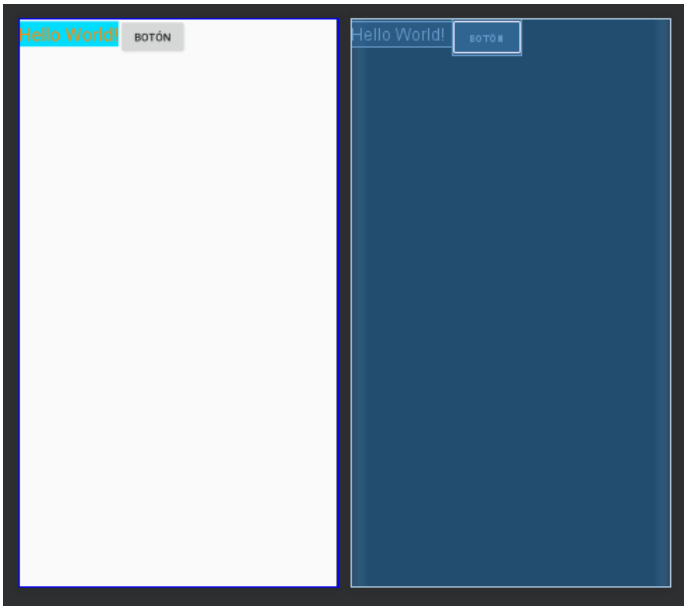


Figura 16. LinearLayout simple

Las dos unidades para definir tamaño en el móvil son '*sp*' y '*dp*'. '*dp*' significa *density-independent pixels* y '*sp*', *scalable pixels*. Básicamente la diferencia es que el segundo tipo está relacionado con la configuración del usuario. Por tanto, '*sp*' se utiliza en textos para que se ajuste la fuente al tamaño de letra del móvil y '*dp*' se utiliza en el resto.

Volviendo a la disposición lineal, a primera vista puede parecer limitada, pues pone los objetos totalmente juntos y linealmente. Sin embargo, hay algunos atributos con los que se puede modificar su comportamiento notablemente. Añadimos al '*LinearLayout*' los dos atributos siguientes:

```
android:gravity="center"
android:orientation="vertical"
```

Solo con esto el resultado pasa a ser:

Los objetos empiezan a distribuirse linealmente en vertical intentando quedarse centrados.

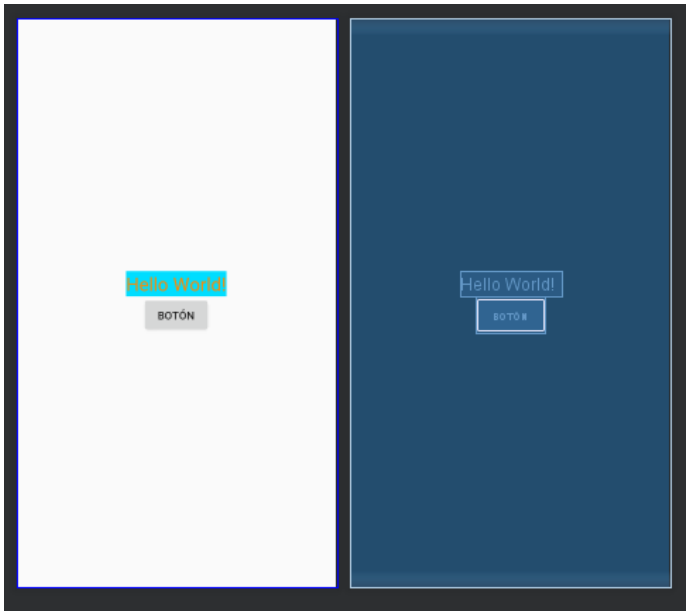


Figura 17. LinearLayout vertical centrado

Sin embargo, aún hay posibilidad de mejora. Los objetos como se ve están pegados, para separarlos tenemos dos opciones: '*padding*' y '*margin*'.

El '*layout_margin*' tiene la característica de que puede aplicarlo tanto la '*ViewGroup*' (es decir, el *Layout* en el que se está contenido) como el objeto. También hay una diferencia en la aplicación de los fondos. Para representarlo pongo este ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:layout_margin="5dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textColor="@android:color/holo_orange_dark"
        android:textSize="24dp"
        android:background="@android:color/holo_blue_bright"
        android:padding="30dp"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@android:color/black"
        android:text="Otro Texto"
        android:textSize="24dp"
        android:textColor="@android:color/white"
        android:layout_margin="30dp"
        />
    <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Botón"
        android:padding="30dp"
        />

</LinearLayout>
```

Como se puede observar hay un primer margen generado por el *layout* lineal y después cada uno de los componentes con sus márgenes. Aquí se puede ver la diferencia más importante entre el margen (“Otro Texto”) y el relleno (“Hello World!”). El relleno da su fondo al sistema, mientras que el margen no.

Figura 18. Muestra de Padding y Margin

Los *layouts*, pueden ser anidados en el interior de otros para aumentar el número de posibilidades.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello World!"
            android:textSize="24sp"
            android:textColor="@android:color/holo_orange_dark"
            android:background="@android:color/holo_blue_bright"
            />

        <TextView
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:background="@android:color/black"
            android:text="Otro Texto"
            android:textSize="24sp"
            android:textColor="@android:color/white"
            />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Botón"
            />

    </LinearLayout>

    <ImageView
        android:src="@drawable/Gato"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:scaleType="center"
        />

</LinearLayout>
```

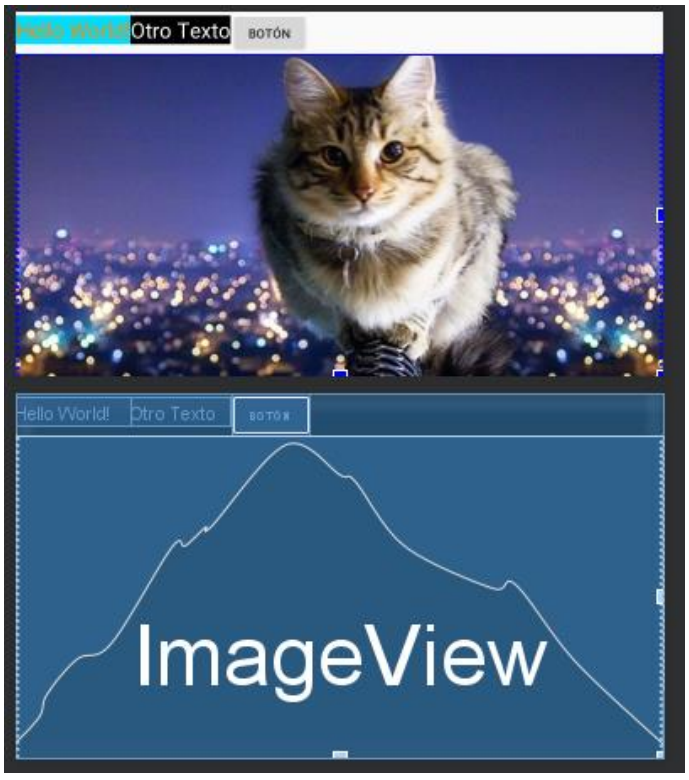


Figura 19. LinearLayout anidado e ImageView

Como se puede ver, se tiene por un lado el *layout* horizontal en donde están los dos textos y el botón dentro de un *layout* vertical que contiene en paralelo la imagen de un felino.

En este código introduzco dos conceptos extra. El primero, el código para introducir una foto. Para incluir una foto propia como es el caso hay que incluir en la biblioteca de "*drawable*" de recursos ("*res*") el archivo deseado. El otro concepto que se introduce es '*layout_weight*'. Este comando sirve para indicar que el objeto debe ocupar todo el espacio disponible en relación su entero con el resto de los pesos. Esto queda confuso, así que modificaré los dos textos y el botón para el siguiente ejemplo.

```
<TextView
    android:layout_width="0dp"
```

```
    android:layout_weight="2"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="24sp"
    android:textColor="@android:color/holo_orange_dark"
    android:background="@android:color/holo_blue_bright"
    />

<TextView
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="wrap_content"
    android:background="@android:color/black"
    android:text="Otro Texto"
    android:textSize="24sp"
    android:textColor="@android:color/white"
    />
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Botón"
    />
```

Figura 20. Linear Layout anidado con Weight e ImageView

Como se puede ver, el primer texto ocupa el doble que el segundo atendiendo a su relación de valores (2 a 1) y el botón ocupa únicamente su tamaño.

Se considera una buena práctica dejar el valor del tamaño que se va a rellenar vacío. Evita errores y configuraciones anómalas.

El *RelativeLayout* es el otro *layout* básico. En él, la posición se indica de manera, valga la redundancia, relativa. Pero, ¿relativa a qué? Pues al resto de los objetos y al entorno mismo. A continuación, se muestra un ejemplo completo en el que

mostraré bastantes de las posibilidades, ya que no quiero detenerme demasiado en este tema.

Una herramienta útil para trabajar con aplicaciones de mayor tamaño con formas que se repiten a menudo es *RecyclerView*. Esta vista nos permite generar formas rellenables y apilables de manera indefinida con facilidad. El ejemplo más simple de una *RecyclerView* es una aplicación de correo. En la bandeja de entrada cada uno de los correos tiene diferentes textos, imágenes e informaciones únicas que dependen del usuario, por tanto, lo que se hace es construir un formato único que es rellenado de manera iterativa. No he estudiado en profundidad esta herramienta, pero es muy potente y me parecía incorrecto no mencionarla ya que seguramente resulte útil al futuro desarrollador de Android.

Todas las herramientas que se han nombrado aquí son solo un mínimo reflejo de la inmensidad de opciones que existen, entre las que se pueden nombrar *ListView*, *GridLayout* o *ScrollButton*. Tras pensar el formato que se desea, simplemente hay que pensar en los nombres de los objetos, buscar cómo se nombran en Android para proceder a ir a su documentación y, leyendo sus atributos, aprender a utilizarlos.

Por último y como puente para unir con el apartado de java, mencionar un atributo de suma importancia: la identificación. El identificador es un nombre que le damos al objeto para poder encontrarlo desde nuestro código de Java o Kotlin y poder trabajar con ellos. La construcción es:

```
android:id="@+id/Boton"
```

Como en muchas construcciones, el sistema ayuda mucho con las recomendaciones para autocompletar.

3.7 Actividades en Java

Para explicar el funcionamiento de Java en Android partiré del esquema que nos aparece al generar un nuevo proyecto.

```
package android.example.aplicacionmuestra;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Analicemos las partes.

Tras una declaración del paquete, se indican las bibliotecas importadas y entonces se comienza la clase. Para aquel nuevo estudiante de Android (como yo mismo antes de empezar este trabajo final de grado) explicaré simplemente que Java está basado en objetos y clases, siendo las clases una clase en Java es un constructor o plano para crear objetos e interaccionar con ellos. En este caso, no se crea un sistema de cero, si no que se está partiendo ('*extends*') de un sistema ya existente y dándole un funcionamiento específico. En este caso, la clase de la que se parte es la existencia de la aplicación e infinidad de controles básicos como actuación del sistema al crearse, relación con el hardware… Tras esa clase, nos muestra un "*Override*" y una función que se aplica al crear el programa. Inicialmente el programa no hace nada cuando se crea, y es trabajo nuestro el incluir en este apartado las tares que se deseen hacer.

Escribiré un código sencillo en el que se consigue preparar un botón para ser pulsado.

```
package android.example.aplicacionmuestra;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button nombreBoton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nombreBoton = (Button) findViewById(R.id.Boton);

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

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

Como se puede ver se genera una variable '*Button*' para coger un botón que habíamos creado en el código xml y al que habíamos dado un identificado mediante el atributo '*id*'. Tras ello, se crea un "*Listener*", un sistema que actúa cuando sucede una condición, en este caso que el botón sea presionado. Hay muchos tipos de *Listener*, en botones que el botón se apriete o se suelte, pero en otros sistemas como audio o cámara, sistemas que actúan cuando una canción llega a su final o a un punto determinado.

Sin embargo, tener que conocer y escribir todo este código sin equivocarse al escribir un '*Override*' o la nomenclatura del sistema sería horrible. Por suerte, existe un autocompletar que nos introduce gran parte de este código del que se puede decir "no aporta lógica" sino solo define el sistema como es necesario. Este autocompletar es muy útil, porque no solo nos evita escribir código intermedio, sino que cuando se precisa de una nueva biblioteca nos la crea de manera automática.

Un tema importante en Java (a diferencia de otros lenguajes más "amables" como Python) es la designación de variables. Para crear una variable es necesario declarar previamente qué tipo de variable es. Se puede apreciar en la llamada al botón que primero invocamos la variable *Button* (que técnicamente no es de Java sino una adición de Android) y tras ello se puede elegir el nombre. Si se utiliza el autocompletar, el sistema ofrece como nombre de la variable el mismo nombre, pero con la primera letra en minúscula. Se puede utilizar si se desea o no, otros autores prefieren poner una "*m*" antes del objeto (es decir, "*Button mButton*"), pero todo esto son realmente costumbres o manías. Lo que sí es importante, es que la primera letra debe ser minúscula. O sea, el sistema permite que la primera letra sea mayúscula, pero por convenio está visto de una manera horrible y realmente tiene bastante sentido, si todas tus clases de objetos y variables empiezan con mayúscula y tus objetos y funciones con minúsculas, puedes saber de un simple vistazo con qué se está trabajando y queda un código mucho más claro para posteriores revisiones.

Pongamos el caso de que no se ha enunciado el importe de la biblioteca con la que se trabaja, en ese caso, el código saldrá en rojo declarando el error. Para resolverlo de una manera rápida, hay que colocarse sobre la zona cuestionada y usar el atajo de teclado Alt + Enter. Cuando se hace eso, sale un cuadro con opciones como el que se ve en la figura. Se selecciona importar la clase y añade la línea de código que resuelve el problema.

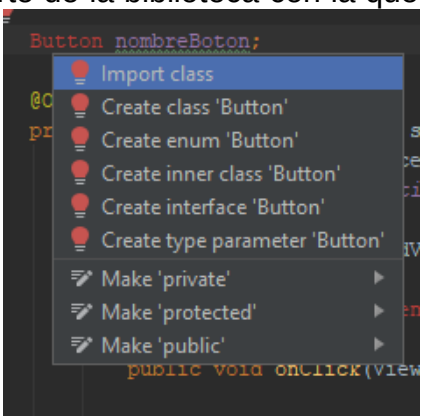


Figura 21. Correción rápida

Esto puede resultar extremadamente útil cuando se copian y pegan trozos de códigos de unos proyectos a otros.

Cuando se invocan algunos objetos, hay que designar cómo se quieren.

```
private int privado = '1';
public int publico = '1';
private final int numero_final = '1';
```

Se pueden declarar públicos o privados. En general, es mejor declararlos privados, porque es menos costoso computacionalmente, aunque si se va a trabajar con varios códigos puede ser necesario declararlos públicos. Al declararlos '*final*' se está haciendo que el objeto no pueda ser modificado. Con esto se consiguen dos cosas, asegurar que no puedas modificar un valor que quieres dejar constante por error (un código de una llamada, o la frecuencia de muestreo que vas a utilizar) y además reduces la memoria reservada para ese objeto.

Se puede declarar sin darle valor a la variable y también crear variables en medio del código en caso de que no se deseen que perduren en la aplicación. Por ejemplo:

```
int a = privado+publico+numero_final;
```

Que es lo mismo que:

```
int a;
a = privado+publico+numero_final;
```

Aquí se está creando un entero en el que se suman tres valores. Se puede utilizar en esta línea de comando, pero al introducirlo en sistemas que están en niveles más profundos (por ejemplo, un bucle dentro de un '*if*') puede ser imposible acceder a ellos.

En estos casos, la solución es declarar “a” al principio del código como una variable privada y actuar de manera normal.

Pongamos el caso de que queremos crear una función o una afirmación lógica. Partiendo del texto siguiente y haciendo Alt + Enter (a partir de ahora se llamará a este atajo de teclado “corrección rápida”) se nos ofrecen dos opciones, renombrar el elemento o definirlo, al elegir definir se nos plantean otras dos opciones, crear el nuevo elemento en ‘*onCreate()*’ o en *MainActivity*. Yo, por organización, prefiero la segunda opción pero, sinceramente, toda la importancia que tiene es la localización.

```
if (dime_booleano()){
    aplica_cuando_booleano();
}
```

Al hacer la corrección rápida con ambos elementos se nos generan dos códigos a rellenar:

```
private boolean dime_booleano() {
});
}
private void aplica_cuando_booleano() {
}
```

Ahora solo falta introducir el funcionamiento que se desee en cada elemento.

Hago aquí un pequeño aparte en el que introduzco una plantilla creada por mí donde se muestran los métodos ‘*while*’, ‘*for*’ y ‘*if*’.

```
boolean afirmacion = true;
if (afirmacion){
    // Introducir actividad
}
for (int i=0;i<100;i++){
    // Introducir actividad
}
int i = 0;
while (i<100){
    i++;
}
```

‘*i*++’ indica añadir uno a la variable “*i*”.

Para comentar se utilizar la doble barra. Si se quiere comentar un tramo largo se puede optar por escribir “/*” al comienzo del comentario y cerrar con “*/”.

Algunos comandos, necesitan un manejo específico de excepciones. Como ejemplo, voy a mostrar la preparación de un reproductor de audio.

```
mediaPlayer.prepare();
```

Para resolver la excepción, se puede utilizar la corrección rápida, se selecciona rodear de '*try*' y '*catch*' y deja un código funcional como el siguiente.

```
try {
    mediaPlayer.prepare();
} catch (IOException e) {
    e.printStackTrace();
}
```

3.8 Almacenamiento

Para almacenar información en la aplicación se tienen tres opciones:

1. Almacenamiento interno o archivos: consiste en almacenar los archivos como archivos en la memoria del dispositivo. Necesita permisos de usuario para poder escribir. Es recomendable para grandes archivos multimedia.
2. Preferencias compartidas: son variables en las que se elige el valor final entre opciones. Están diseñadas para guardar las preferencias del usuario. Están formadas por una clave (*key*) que es un texto único, el nombre de la variable; y un valor que es uno de los tipos disponibles.
3. Bases de datos SQLite: buenas para organizar grandes cantidades de información relacionada y estructurada para el fácil acceso.

SQL (Structured Query Language) [ref. 17] es un lenguaje específico para administrar y recuperar información de bases de datos relacionales. Algunos programas de manejo de bases de datos, como Microsoft Access, utilizan SQL como raíz para trabajar. SQLite es una plataforma derivada de SQL que está establecida como sistema de bases de datos de Android.

Figura 22. Logo de SQLite

El objetivo de este trabajo no es hacer una explicación pormenorizada del sistema, pero se van a intentar definir los puntos importantes.

Hay 5 tipos de objetos: nulo, integral, real, texto y blob (información guardada directamente como la introducimos, se utiliza para imágenes, binarios…). Según las consideraciones de nuestro sistema, se genera una tabla indicando el nombre de las columnas, los tipos de objetos y sus características (valores no repetibles, no nulo, valores por defecto…).

Toda esta declaración hay que realizar en el lenguaje base de SQLite, lo cual resulta incómodo. Por tanto, la práctica habitual es crear una nueva clase en otro documento de Java en el que se crea una clase (normalmente llamada *DatabaseHelper*) donde se define a partir del *SQLiteOpenHelper* toda la forma de nuestra tabla, la creación de la misma y la introducción y borrado de datos.

3.9 Permisos de usuario

El acceso a las actividades del sistema está restringido por permisos. Hay dos tipos de permisos:

- Permisos normales: incluyen aquellos en los que la aplicación accede a datos o recursos del exterior a su entorno, pero que suponen un riesgo mínimo de privacidad. Por ejemplo, el huso horario.
- Permisos arriesgados: son aquellos que incluyen o pueden incluir o permitir el acceso a información privada del usuario. Estas actividades están vetadas a no ser que el usuario haya dado consentimiento directo y explícito en la aplicación. A continuación, se recoge una lista de los grupos de permisos arriesgados; cada grupo tiene uno o varios permisos que se pueden buscar en caso de necesidad en la página de desarrolladores de Android:
    1. Calendario.
    2. Registro de llamadas.
    3. Contactos.
    4. Cámara.
    5. Localización.
    6. Micrófono

7. Llamadas.
8. Sensores
9. SMS.
10. Almacenamiento.

Los permisos se piden desde la programación en Java. La descripción que voy a realizar a continuación es un extracto del reproductor con gráfica que se mostrará posteriormente.

Lo normal es construir un booleano que compruebe si tenemos los recursos que deseamos.

```
private boolean revisarPermisoDipositivo() {
    int almacenaje_externo = ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
    int record_audio = ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO);
    return almacenaje_externo == PackageManager.PERMISSION_GRANTED &&
            record_audio == PackageManager.PERMISSION_GRANTED;
}
```

En caso positivo, se continúa el funcionamiento del sistema. En caso negativo, se eleva una petición de los permisos.

```
private void pedirpermiso() {
    ActivityCompat.requestPermissions(this, new String[]{
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.RECORD_AUDIO
    }, REQUEST_PERMISSION_CODE);
}
```

Tras esto, suele ser conveniente hace un "*Override*" (recordamos, control+O para invocarlo) del método de "en resultado a la petición de permiso" para mostrar por pantalla el resultado. El código empleado es:

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_PERMISSION_CODE: {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)
                Toast.makeText(this, "Permiso Concedido",
Toast.LENGTH_SHORT).show();
            else
                Toast.makeText(this, "Permisos necsarios para la
aplicación", Toast.LENGTH_SHORT).show();
        }
        break;
```

```
    }
}
```

Sobre todos estos códigos hay que comentar dos cosas. La primera, *REQUEST_PERMISSION_CODE* es una variable que tenemos que crear. Es un entero que define el número de petición de permiso. Puede ser el que escogemos. Yo he puesto:

```
final int REQUEST_PERMISSION_CODE = 1;
```

'*final*' es añadido para que no pueda recibir variaciones.

Lo segundo que hay que comentar, es que nuestra aplicación tiene que saber que esos permisos van a ser requeridos, esto se hace añadiendo una línea de código al manifiesto:

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

Estas líneas deben estar al mismo nivel que la de aplicación y debe ponerse una línea para cada uno de los permisos. Si se tienen dudas en este aspecto, obsérvese su implementación en el manifiesto incluido en el anexo (ojo referencia).

3.10 Miscelánea

En este apartado quiero incluir algunos temas que no he mencionado y que son demasiado específicos para dedicarles un apartado exclusivo.

- Para quitar la barra de actividades hay que ir al archivo: "`/res/values/styles.xml`" y sustituir la cuarta línea:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
```

Por:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
```

- Si se quiere diseñar una posición obligatoria hay que ir al '*Manifest*' y añadirle a la actividad envuelta el atributo:

```
android:screenOrientation=""
```

Si se tienen dudas, mirar en cualquiera de los 'Manifest' del anexo enlace.

- La razón por la que se puede querer bloquear la pantalla es que cuando se gira la pantalla de un dispositivo móvil, los datos que están en forma temporal son borrados. Es decir, en este caso la gráfica desaparecería por un momento. Aquí el problema no es muy grande, pero en otros casos se puede perder información importante. Es obvio que este problema no lo encontramos en las aplicaciones que tenemos instaladas de manera normal en el móvil. Esto se debe a que guardan en el teléfono información para que cuando se gire la pantalla se pueda acudir a ella y evitar este problema. Hay '*Listeners*' que se encargan exactamente de esto.

- Para modificar el icono se puede optar por dos estrategias. La primera es acudir a la carpeta "`/res/mipmap/`" y sustituir los archivos que hay ahí. La otra opción es introducir los archivos del gráfico que queremos como icono en cualquier carpeta interior del proyecto y en el '*Manifest*' designar el camino hasta los mismo modificando los atributos:

```
android:icon="@mipmap/ic_launcher"
android:roundIcon="@mipmap/ic_launcher_round"
```

  Se puede ver que en la misma zona está la opción de cambiar el nombre de la aplicación.

4 Desarrollo de la aplicación

Recordemos nuestro objetivo. Queremos preparar una salida con un primer tratamiento de datos de una señal de audio. Entre las decisiones que se tienen que llevar a cabo es qué formato van a tener nuestras grabaciones y con qué se va a mostrar las gráficas de las amplitudes del sonido.

4.1 Sistemas de audio de Android Studio

Los comandos básicos nativos de gestión de audio en Android Studio son *MediaRecorder* y *MediaPlayer*. El *Recorder* solo proporciona formatos de audio codificados (.3gpp, AAC, MPEG) lo cual provocó dudas al inicio del proyecto por la posible pérdida de información, pero en primeras instancias se decidió investigar cómo era trabajar con ello.

La forma de la grabadora multimedia es sencilla, se estipulan el formato y localización de salida y condiciones de medida y realiza de manera simple y eficaz el proceso.

El *MediaPlayer* era aún más sencillo para trabajar que el comando grabador. Solo con invocar el archivo y preparar el sistema con una estructura adecuada para detección de errores el sistema funcionaba correctamente.

El problema llegó a la hora de intentar realizar la obtención de los valores de amplitud de la onda. Se probó con el uso del *AudioManager* ya que se encontró que entre los comandos que contenía existía ‘getStreamVolume()’ y ‘getStreamDb()’. Por desgracia, estos comandos hacían referencia solamente al volumen de reproducción del sistema y no a la amplitud.

Por otro lado, se encontró el *VolumeShaper*, un sistema que permite el control del volumen en función de, entre otras cosas, la amplitud de la onda de sonido en cada momento. Al empezar a analizarlo, se descubre que es de *API* 26. Una API mayor a la del sistema con el que se hacen todas las pruebas de las aplicaciones, por tanto, no fue descartado. Cabe mencionar que no se comprobó si esta clase nos permitiría

hacer la tarea deseada debido a que cualquier resultado obtenido por este camino hubiese resultado inútil.

Continuando con la investigación, se decidió optar por *AudioTrack*. *AudioTrack* es un recurso de audio de java que permite tanto la reproducción de sonido a partir de la introducción de buffers del audio en PCM (Pulse-Code Modulation), es decir, el vector de amplitudes.

Con esto, se decidió probar la implementación del sistema trabajando con el audio en formato *raw*. Raw es un formato de audio en el cual los datos están representados de manera directa, sin compresión. Se desarrolló un programa para extraer el vector de amplitudes para enviárselos a la clase *AudioTrack* como datos de reproducción. El resultado no fue satisfactorio. El reproductor solo reproducía unos instantes (~0.2 segundos) antes de detenerse. Tras una búsqueda más exhaustiva en internet del uso de este sistema, se descubrió que como reproductor solo se suele usar para sonidos cortos como notificaciones o en videojuegos. La razón de esto es que se pueden preparar distintos objetos de *AudioTrack* cada uno para un sonido y, al utilizarlo, reproducen con latencia muy baja, algo importante en videojuegos.

Sin embargo, esta vez el camino parecía el correcto. En las pruebas realizadas con *AudioTrack* se observó que sí que realizaba correctamente la grabación de audio y se podía obtener de manera relativamente sencilla el vector de amplitudes. Por tanto, se decidió emplear el formato *WAV* (Waveform Audio Format) que, en resumen, es equivalente al raw, pero con un encabezado. Lo importante sobre este formato es que el reproductor de Android (*MediaPlayer*) lo soporta, por lo que, si se es capaz de grabar la señal en este formato, será posible obtener el vector de amplitudes y la reproducción del audio, los dos elementos que necesitamos para la realización de este proyecto.

4.2 Formato de audio WAV

El formato WAV o .wav (*Waveform Audio Format*) es un formato de audio digital sin comprensión de datos [ref.18, 19]. Utiliza audio PCM (*Pulse-Code Modulation*) como el resto de audios digitales.

Está formado por un encabezado de 44 bytes seguido de los valores de amplitudes. Estos 44 bytes contienen la siguiente información:

- **1-4**: Los caracteres “RIFF” que significan *Resource Interchange File Format*. Cada carácter ocupa un byte.
- **5-8**: Tamaño del archivo global. El tamaño limitado de este campo lleva a que la duración máxima de un archivo wav sea limitada. El límite es de 4 GB, lo que equivale para un sonido muestreado a 44100 Hz y codificado a 16 Bits una duración superior a 6 horas. Por tanto, es más que suficiente para nuestro proyecto.
- **9-12**: Los caracteres “WAVE”. Es la designación del tipo de archivo.
- **13-16**: Los caracteres “*fmt* ” (con el espacio incluido, son cuatro caracteres).
- **17-20**: Longitud del formato, en nuestro caso 16.
- **21-22**: Tipo de formato, ya sea PCM o entero.
- **23-24**: Número de canales (mono, estéreo…)
- **25-28**: Frecuencia de muestreo. En nuestro caso 44100.
- **29-32**: Resultado del cálculo $\frac{Sample\ Rate*BitsPerSample*Channels}{8}$
- **33-34**: Resultado del cálculo $\frac{BitsPerSample*Channels}{8}$
- **35-36**: Bits por muestra.
- **37-40**: Los caracteres “data”.
- **41-44**: Número de datos.
- **44+**: Resto de datos.

Antes de proceder al desarrollo de un sistema que realizase esta tarea, se buscó la existencia de alguna biblioteca o programa que realizase esa tarea y se encontró una publicación de GitHub que tenía como objeto compartir un código de java que realizaba esta tarea. Este código estaba bajo la licencia Apache, Version 2.0 [ref. 20], por lo que es de libre uso y disfrute para el usuario que lo desee. El usuario que

aportó este desarrollo se llama Kevin Mark “kmark” en GitHub [ref. 21]. El código final que se mostrará en el subapartado 4.5 es una modificación de la original.

4.3 Herramientas de representación gráfica y GraphView

Además de la obtención y el tratamiento del sonido, se desea hacer una representación gráfica de las amplitudes de las formas de onda. Para hacerlo, se realiza una búsqueda en diferentes fuentes.

Se encuentran algunas bibliotecas de formas de ondas como la de New Venture Software [ref. 22], una empresa búlgara dedicada en soluciones software. Sin embargo, al analizar el código que aportaban desde su usuario de GitHub se descubre que dan problemas en su funcionamiento, con algunas funciones depreciadas.

Ante esta situación se decide ir a por la otra opción que se había encontrado. Una biblioteca llamada GraphView [ref. 23] desarrollada por Jonas Gehring que tiene bastantes usuarios, así como guías y tutoriales tanto propios como de terceros.

Ofrece multitud de opciones de gráficos e incluye una aplicación tutorial en la que se pueden ver las opciones que aporta. A continuación, se recogen algunas imágenes que muestran algunas de estas opciones.

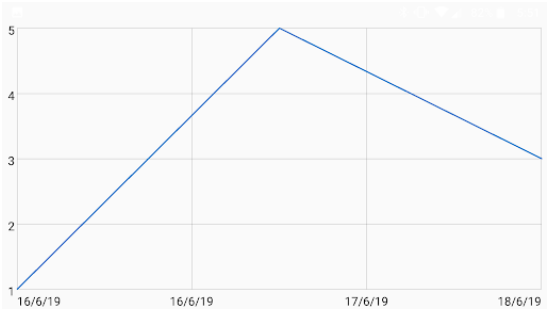


Figura 23. Gráfica con Fechas

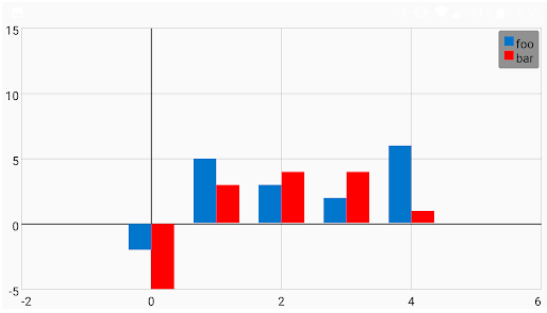


Figura 24. Gráfica de barras

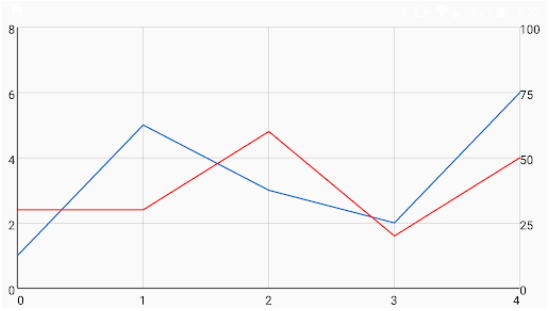


Figura 25. Gráfica con doble escala

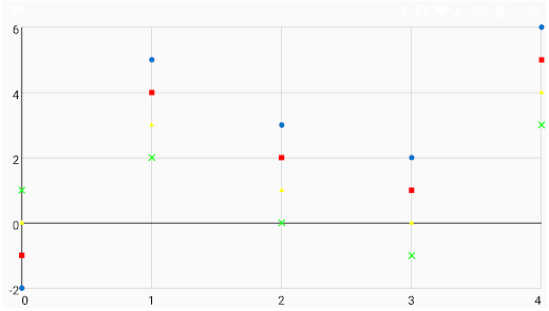


Figura 26. Gráfica de puntos

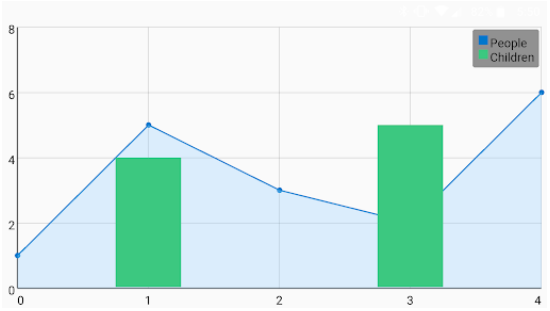


Figura 27. Gráfica mixta con leyenda y estilo

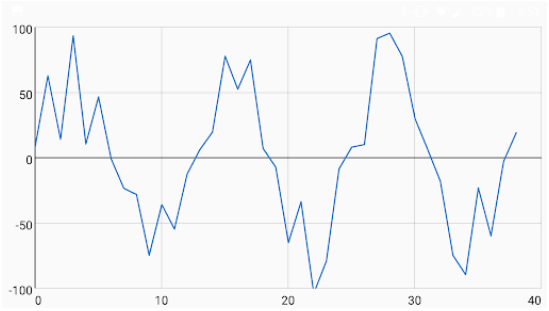


Figura 28. Gráfica modificada a tiempo real

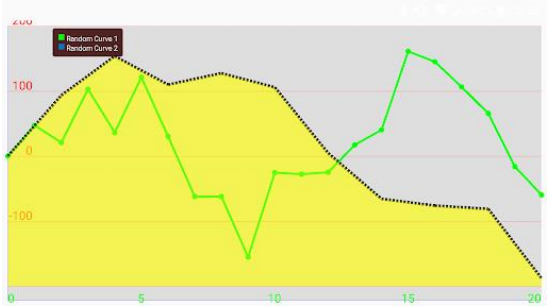


Figura 29. Estilos de gráficas

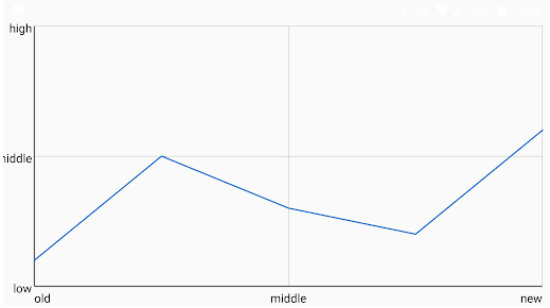


Figura 30. Gráfica con texto en escalas

Por tanto, esta biblioteca nos puede resultar útil para el trabajo que vamos a realizar, ya que entre su amplio catálogo de opciones incluye gráficas temporales que podremos modificar para nuestro uso.

4.4 Aplicación reproductora de sonido guardado

Los sistemas que se han desarrollado para este Trabajo Fin de Grado no incluyen un sistema de almacenamiento de archivos desarrollados. Se utiliza una única ruta o "*path*", lo que significa que, al grabar un nuevo audio, el último archivo es eliminado. Esto nos plantea un problema: si se muestra únicamente una grabadora, resultará

virtualmente imposible mostrar la onda cardiaca en la presentación ya que al hacer una grabación para mostrar el correcto funcionamiento se perderá la muestra de ritmo cardíaco.

Para solventar este problema se ha decidido utilizar dos aplicaciones, la primera reproducirá el audio guardado que contiene los latidos y la segunda será una grabadora que almacenará y reproducirá los datos recién guardados.

El aspecto del programa final al arrancarlo se muestra en la figura siguiente.

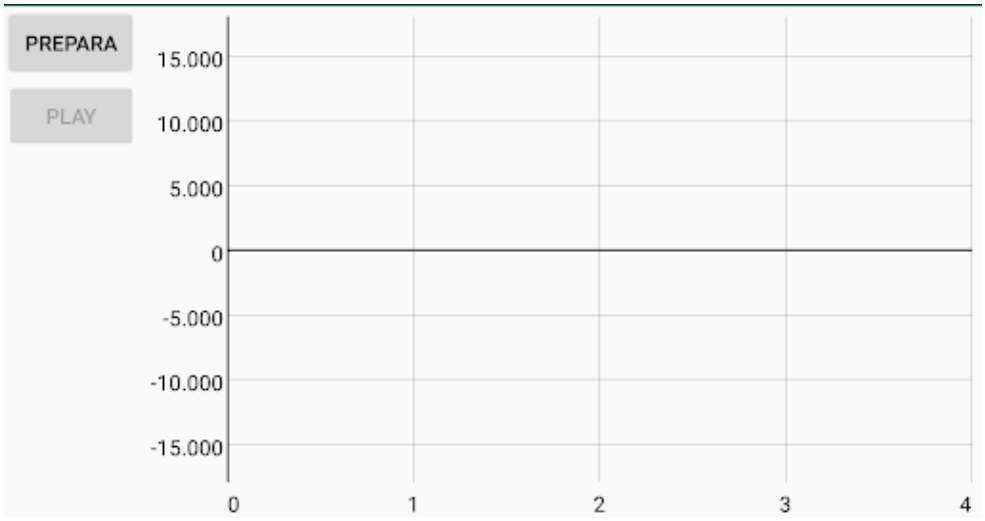


Figura 31. Pantalla inicial de la aplicación

Se utilizan tres objetos, dos botones y una gráfica de la biblioteca "*GraphView*". Los dos botones están dentro de un '*LinearLayout*' vertical que se encuentra en el mismo nivel que la gráfica en un '*LinearLayout*' horizontal.

Se ha configurado el sistema para que se mantenga con orientación bloqueada en modo '*reverseLandscape*' (es decir, al mirarlo los botones del dispositivo quedan a la izquierda, o lo que es lo mismo, la parte de arriba de la vista corresponde a la parte izquierda de la pantalla).

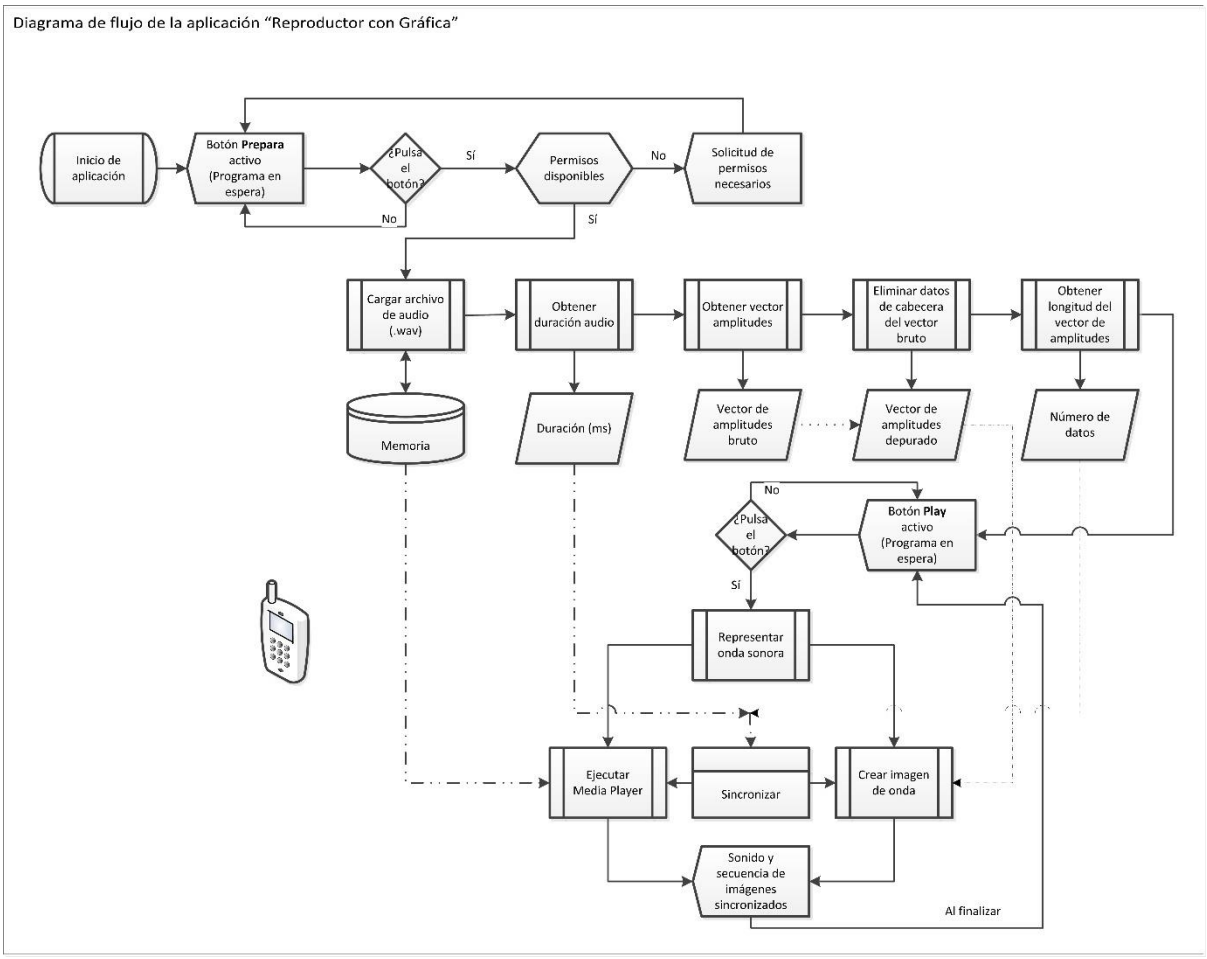


Figura 32. Diagrama de flujo del reproductor con gráfica

Como podemos ver en el diagrama de la aplicación anterior los dos botones trabajan de manera secuencial. En un primer estado tenemos el botón "Prepara" activo y el botón "Play" inactivo (Figura 31). Cuando se da al botón preparar el sistema el sistema comprueba si la aplicación tiene permisos de modificar el sistema de archivos y para grabar sonidos. En caso negativo los pide y en caso positivo procede al programa.

Esta preparación consiste en dos etapas: la primera es la preparación del reproductor de audio ('*MediaPlayer*'); y la segunda consiste en la obtención del archivo de entrada y la traducción de la lista de ceros y unos a las amplitudes. Vamos a extraer a continuación ese fragmento del código para explicarlo.

```
private short[] getAudioSample() throws IOException {
    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
    BufferedInputStream bufferedInputStream = new BufferedInputStream(new
FileInputStream(pathArchivo));
    int read;
    byte[] buff = new byte[1024];
```

```
    while ((read = bufferedInputStream.read(buff)) > 0) {
        byteArrayOutputStream.write(buff, 0, read);
    }
    byteArrayOutputStream.flush();
    byte[] data = byteArrayOutputStream.toByteArray();
    ShortBuffer sb =
ByteBuffer.wrap(data).order(ByteOrder.LITTLE_ENDIAN).asShortBuffer();
    short[] samples = new short[sb.limit()];
    sb.get(samples);
    int a;
    a = samples.length;
    samples = Arrays.copyOfRange(samples, 45, a);
    return samples;
}
```

Lo que se hace es tomar los ceros y unos del fichero, hacer varias transformaciones para conseguir una lista de valores cada uno obtenido de la agrupación de 16 bytes y, por último, retirar los cuarenta y cuatro primeros elementos que configuran el encabezado.

En la preparación se activa también un '*Listener*' encargado de responder cuando la grabación termina. Se invoca aquí porque resulta más correcto generarla tras la generación del reproductor de audio.

```
mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
    @Override
    public void onCompletion(MediaPlayer mp) {
        mediaPlayer.stop();
        try {
            mediaPlayer.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Destrucción del Runnable para evitar que se acumulen y produzcan
        // la ralentización del sistema
        mHandler.removeCallbacks(mTimer);
        // Reactivación del botón de reproducir
        btPlay.setEnabled(true);
    }
});
```

Este fragmento lo que hace es parar el reproductor y volver a prepararlo para la próxima vez que pulsemos el botón y preparar la nueva reproducción. Las líneas de '*mHandler*' y el '*btPlay*' se explicarán en el siguiente apartado.

La preparación es casi inmediata y al acabar el sistema se ve de esta forma:

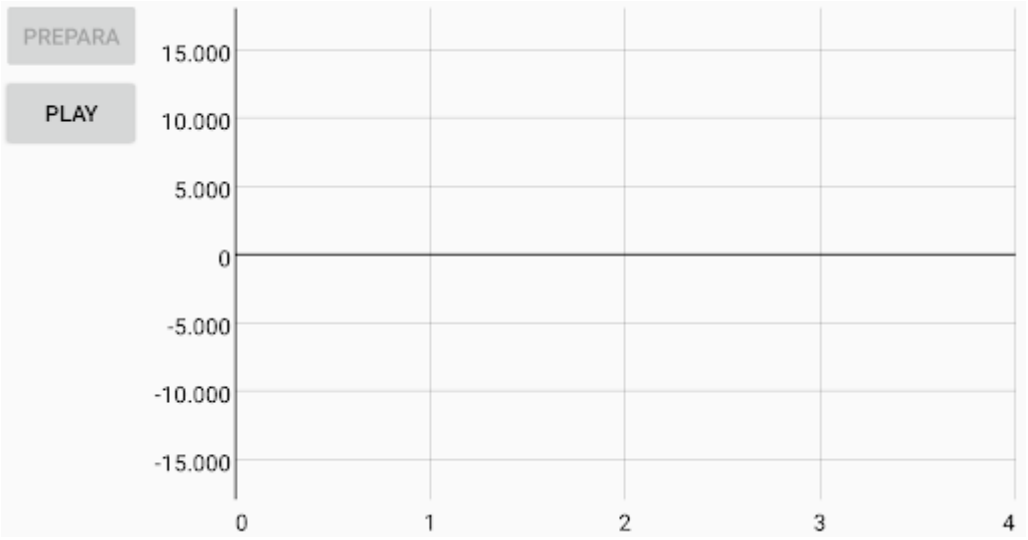


Figura 33. Pantalla con el sistema preparado

Una vez que se activa el botón el sistema inicia la reproducción de audio. Justo tras esto se inicia el bucle de la grabación. Este bucle es un desarrollo de los sistemas que ofrece *GraphView*.

```
mTimer = new Runnable() {
    @Override
    public void run() {
        int tiempo_ms = mediaPlayer.getCurrentPosition();
        desp = (int) (tiempo_ms * 0.441);
        desp = (int) (desp - 1600);
        if (graphLastValue > 40) {
            graphLastValue = 0;
            mSeries.resetData(new DataPoint[]{
                    new DataPoint(graphLastValue, 0)
            });
        }
        for (int i = 1; i < 1600; i++) {
            graphLastValue += 0.1d;
            if (desp + i < 0) {
                mSeries.appendData(new DataPoint(i * 1.0 / 441, 0), false,
4410 * 4);
            } else {
                mSeries.appendData(new DataPoint(i * 1.0 / 441,
muestras[100 * (i + desp)]), false, 4410 * 4);
            }
        }
        mHandler.postDelayed(mTimer, 1);
    }
};
mHandler.postDelayed(mTimer, 1);
```

Básicamente lo que se genera es un '*Runnable*', un hilo que sucede de manera continua. Este hilo es controlado mediante un '*Handler*' (controlador de hilos y otros programas que funcionan a lo largo del tiempo en nuestro dispositivo) que va haciendo que cada vuelta el sistema espera dos milisegundos para volver a entrar.

A la hora de realizar el programa nos encontramos ante un problema: nuestro sistema trabaja en milisegundos y nuestro vector de amplitudes nos da 44100 valores por segundo, por tanto, la traducción no es directa. Además, dibujar 44100 puntos para cada segundo puede resultar demasiado, es decir, si dibujamos unos cuatro segundos de reproducción en una pantalla de móvil hasta qué punto los valores pueden verse reflejados o se pierden en la resolución. Esto aún sin entrar en el tema de latencia por la introducción de estos valores, que en las pruebas previas se notaba que podía ser una limitación. La razón de que la latencia pueda ser una limitación se debe a que el programa de gráficas sigue siendo al fin y al cabo, un sistema diseñado para mostrar gráficas, no un sistema que intente exprimir al máximo las capacidades del móvil para mostrar la mayor cantidad de puntos en el menor tiempo posible.

Tras varias pruebas aportando diferentes sistemas se decide optar por una frecuencia en las ondas mostradas de 441 Hz. Realmente, esta frecuencia puede resultar demasiado baja para mostrar audio de sonidos normales, pero estamos trabajando con latidos cardiacos, que tienen frecuencias más reducidas.

Decidida ya la frecuencia de salida, lo que se hace es obtener el tiempo de la reproducción, transformarlo a nuestra frecuencia de 441 Hz y utilizarlo para dibujar la gráfica.

Cabe mencionar, que en nuestro gráfico no hacemos uso de los cuatro segundos completos, sino que se opta por utilizar un total de 1600 puntos dando un total de 3.628 segundos para que la nueva onda se presenta algo alejada del límite de la pantalla y resulte más cómoda la visualización. Este valor es totalmente arbitrario y se eligió tras probar unas cuantas distribuciones.

Como se ha dicho, el '*Runnable*', es controlado por un '*Handler*' que permite que el programa continúe una vez cada dos milisegundos. Por tanto, cuando acaba la reproducción, el sistema seguirá invocando a la gráfica y gastará más recursos. Además, si se vuelve a pulsar el botón se creará un nuevo '*Runnable*' y se irá saturando el sistema hasta el punto de que tras invocar tres o cuatro '*Runnable'*, el sistema va muy lento y con problemas. La solución a esto es una de las líneas del '*Listener*' de final de reproducción.

```
mHandler.removeCallbacks(mTimer);
```

En esta línea, se desecha esta '*Runnable*' haciendo que deje de ser invocada y que cuando se vuelva a generar no se acumule y genere problemas de carga computacional.

El botón es anulado en toda la reproducción para evitar problemas. Otras opciones serían un condicional que hace el botón inútil cuando el reproductor está reproduciendo, pero, personalmente, me parecía más claro para las personas que estaban viendo el programa hacerlo así.

A continuación, se muestran algunas imágenes tomadas en el funcionamiento de la aplicación.

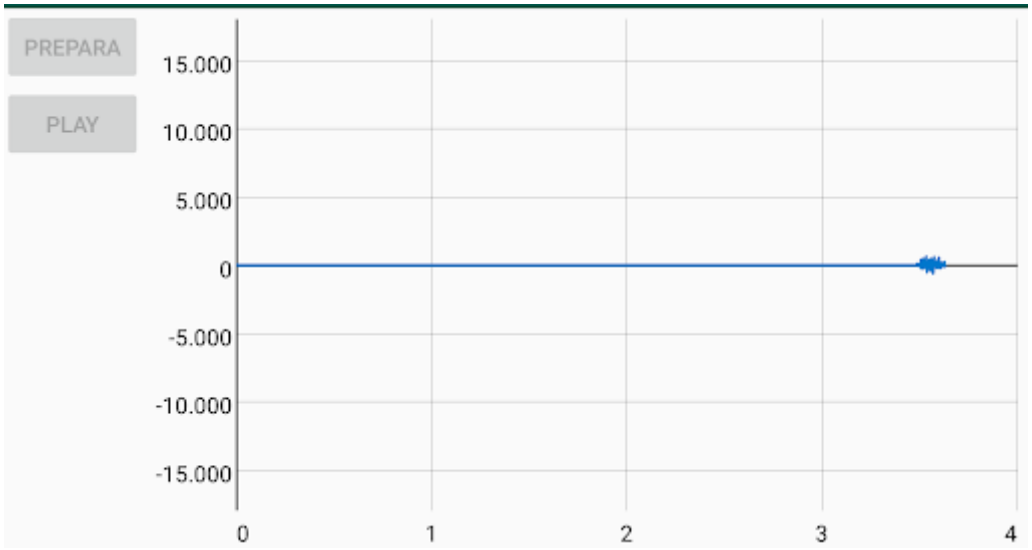


Figura 34. Pantalla al inicio de la reproducción

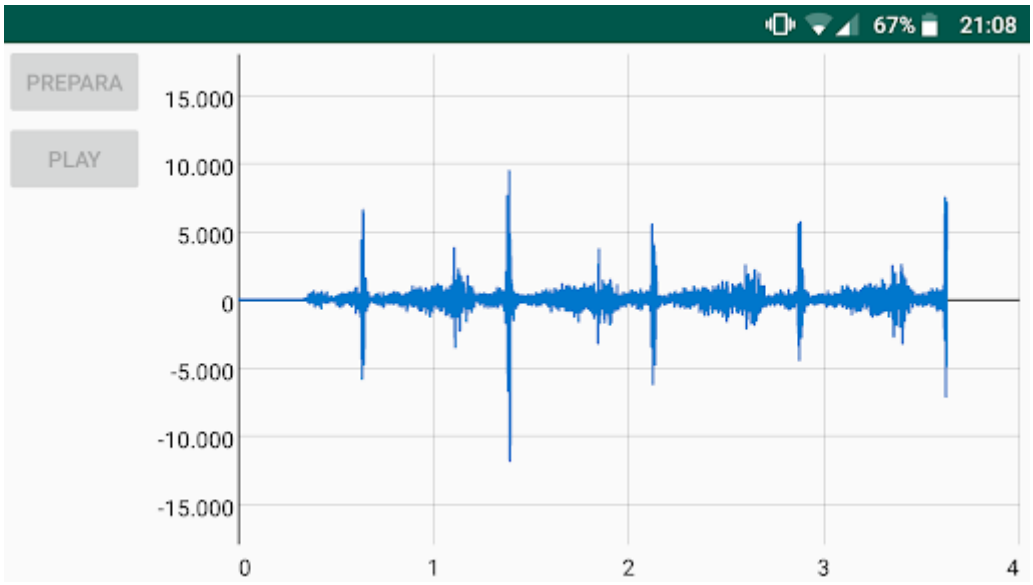


Figura 35. Pantalla durante la reproducción

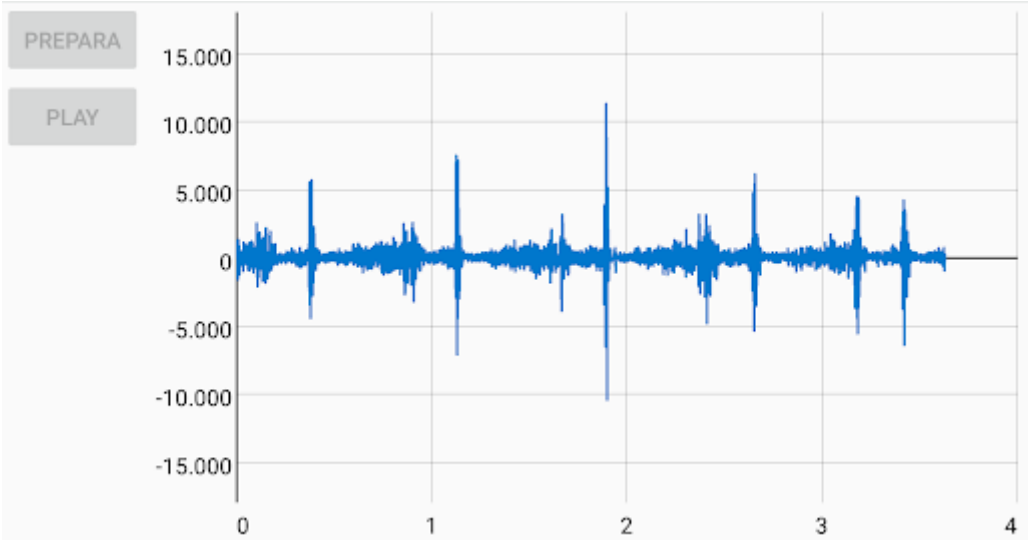


Figura 36. Continuación de la reproducción

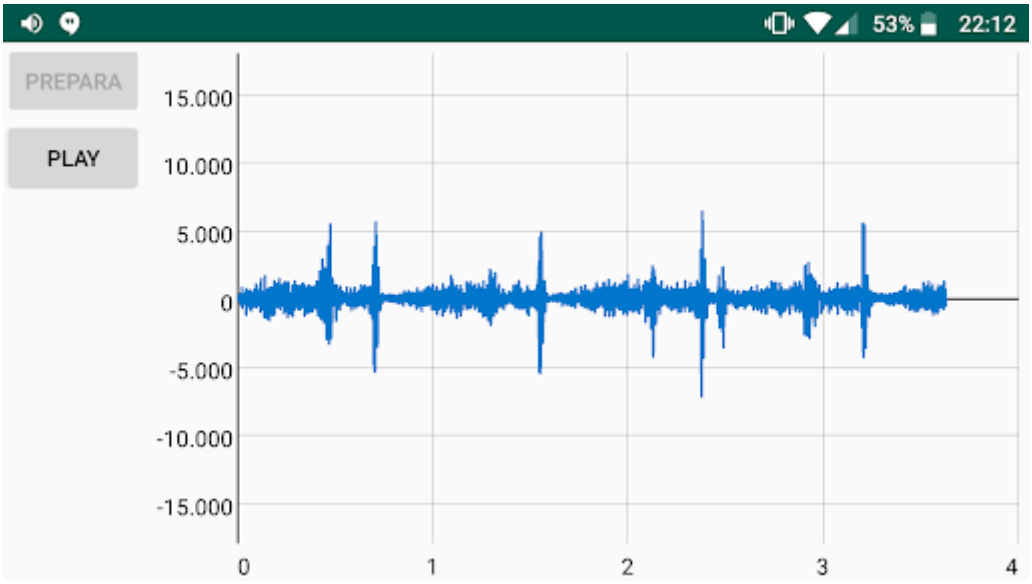


Figura 37. Pantalla al finalizar la reproducción

4.5 Aplicación grabadora en formato .wav

Este programa es una variación del anterior añadiéndole una grabadora en formato WAV. El sistema tiene una apariencia casi idéntica al anterior con la diferencia de que aquí hay dos botones extras que se encargan de gestionar la grabación.

No obstante, la construcción de la interfaz de usuario es idéntica.

Lo único que cabe mencionar como destacado del sistema es que la petición de permisos ha variado su forma, que no su contenido, ya que se ha optado por dejar

la que utilizó Kevin Mark, diferentes creadores generan el mismo servicio de manera estilísticamente distinta.

Sobre los botones cabe mencionar que cuando se activa la grabación los botones “Prepara” y “Play” quedan inutilizados hasta que se detiene, momento en el que “Prepara” se activa y “Play” permanece desactivado ya que la grabación que se iba a reproducir ha sido modificada. Una vez preparado, el botón “Prepara” se desactiva, “Play” se activa y los de grabación permanecen activados. Mientras se reproduce, los botones de grabación no producen ningún efecto y cuando acaba la reproducción se puede elegir entre hacer una nueva reproducción, una nueva grabación o dejar el sistema esperando órdenes.

A continuación, añado algunas imágenes mostrando su funcionamiento.

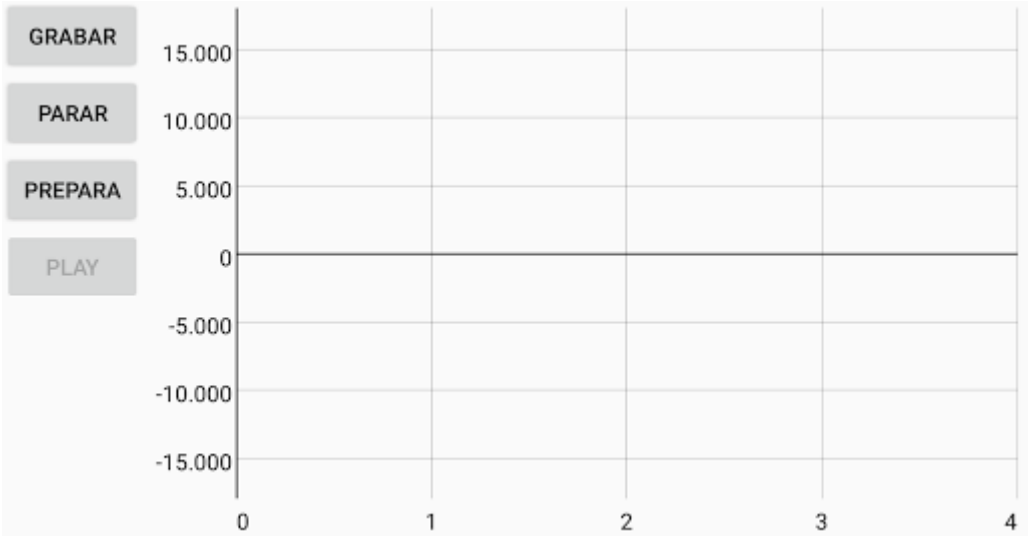


Figura 38. Aplicación en estado inicial

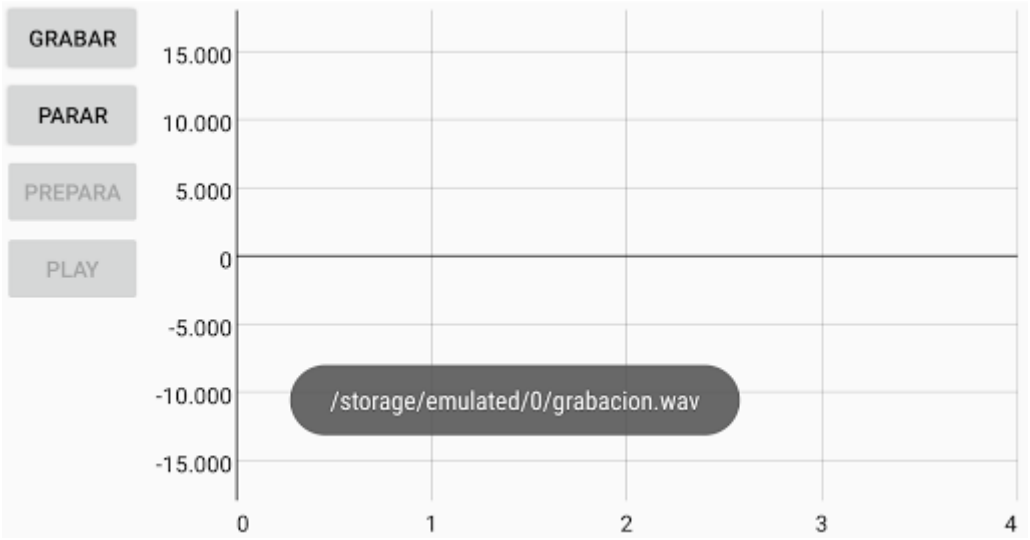


Figura 39. Aplicación iniciando grabación

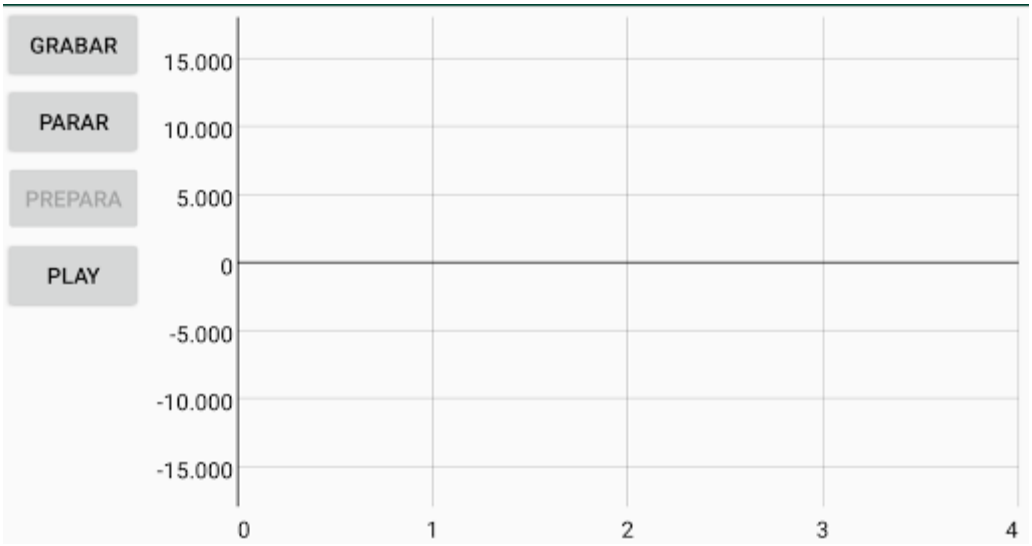


Figura 40. Aplicación preparada para reproducción

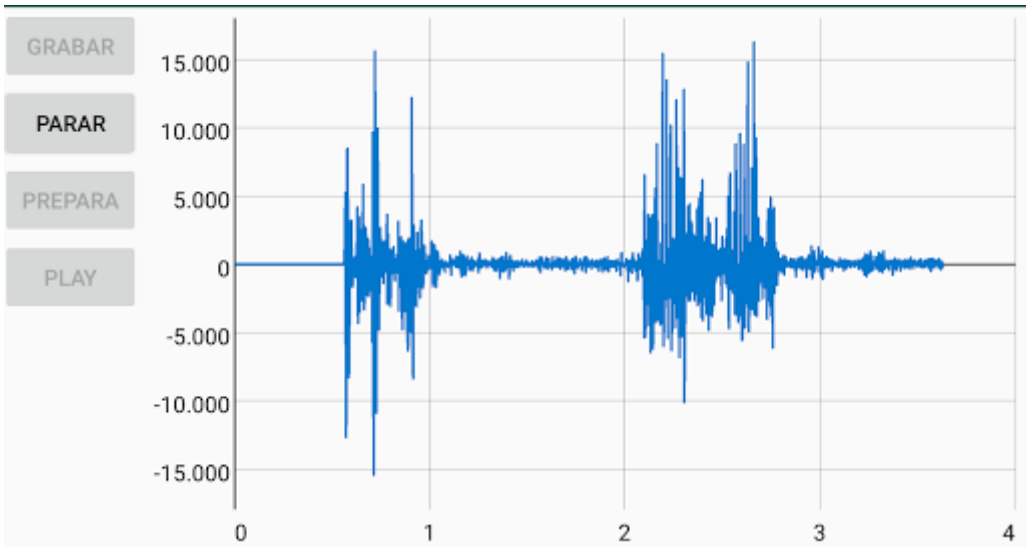


Figura 41. Aplicación en funcionamiento (1)

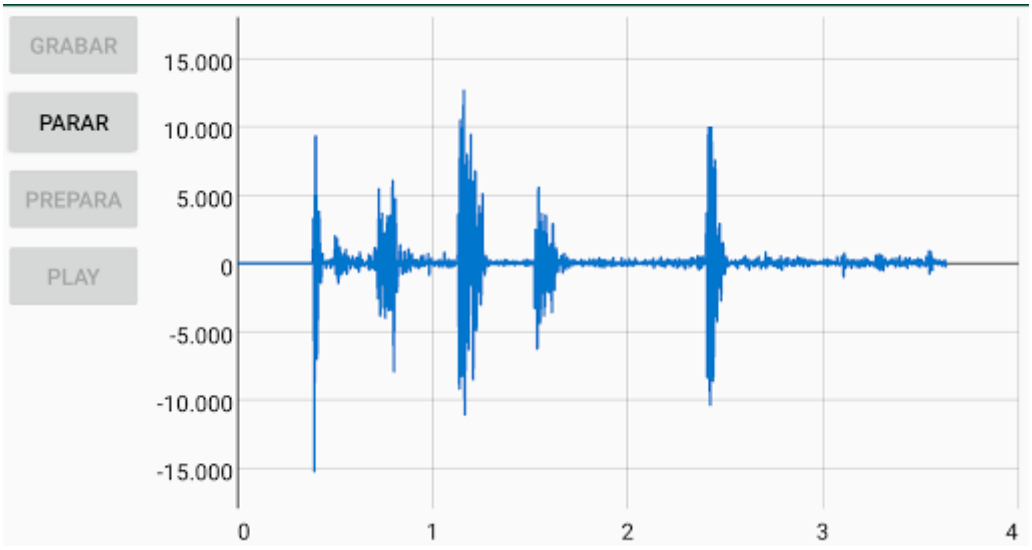


Figura 42. Aplicación en funcionamiento (2)

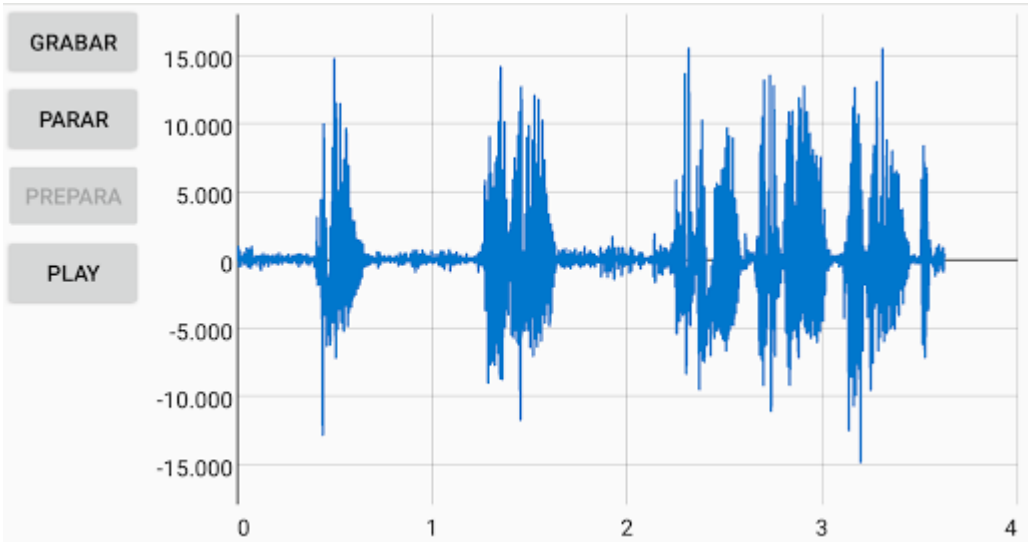


Figura 43. Aplicación terminada la reproducción

5 Otros

Este apartado lo quiero aprovechar para hacer una explicación de diferentes conceptos o prácticas que no he mencionado (o no he ahondado tanto como me gustaría) que un nuevo desarrollador de software Android necesita conocer.

Sobre Android Studio se han encontrado pocos libros que expliquen de manera concisa y eficaz el modo de trabajo en Android Studio, aunque me parece que “Android 100%” [ref.24] es útil para leer y aclarar determinados conceptos.

Para mi formación utilicé un curso recomendado por Android Developers llamado “Android Basics” [ref. 25]. Este curso está alojado en Udacity y dividido en cinco módulos y explica el funcionamiento de la plataforma enfocado a un estudiante sin formación previa de programación. Por todo esto, el tutorial se vuelve largo y pesado en algunas ocasiones. Los diferentes apartados que se han ido desarrollando en este trabajo tienen como objetivo dar una base suficiente que equivale a ese tutorial para personas que saben algo de programación. Aun así, puede resultar interesante acceder al curso porque en él se incluyen entrevistas a desarrolladores profesionales de Google.

Una vez acabada esta formación básica, es útil conocer algunas herramientas para continuar.

La primera es la documentación de Android Studio [ref. 16], en la que se recogen todos los programas. Es semejante a la de otros sistemas, pero quiero mencionar una herramienta útil que tiene:

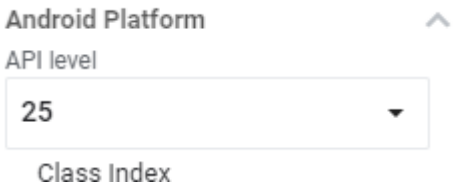


Figura 44. Detalle selección API

En la barra lateral izquierda se puede seleccionar al API con la que se trabaja y el explorador deja en oscuro los comandos y llamadas que no sean soportados. Puede parecer una tontería, pero puede significar un preciado tiempo de estudio en una herramienta que te acabas dando cuenta que no puedes utilizar en tu dispositivo.

La segunda herramienta que hay que conocer es *GitHub* [ref. 26]. *Github* es un repositorio de código en el que se pueden almacenar con

Figura 45. Logo GitHub

opciones gratuitas las diferentes versiones de los programas que generes. Además de este uso, está generalmente extendido para compartir programas de todo tipo, desde aplicaciones Android como las que trabajamos hasta APIs de Deep Learning. No voy a hacer una explicación detallada de su funcionamiento, meramente la forma fundamental de operar con él.

Básicamente, en GitHub se almacenan las aplicaciones enteras iguales a las que se guardan en el ordenador cuando se desarrollan. Por tanto, explorando las carpetas, se puede acceder a sus .xml y sus .java para ver cómo funcionan. Además, existe la opción de clonar el sistema y descargárselo, de tal manera que (tras descomprimirlo) puede ser abierto en nuestro Android Studio y ahí comprobado y modificado libremente (dando siempre crédito a los creadores en caso de que se utilice código ajeno).

La tercera herramienta es *Stack Overflow* [ref. 27]. *Stack Overflow* es una página de preguntas y respuestas utilizada por desarrolladores profesionales para compartir y resolver cuestiones sobre programas. Esta última herramienta sale a menudo cuando se hacen búsquedas en Google sobre asuntos de programación y electrónicos, y es útil para conseguir una primera orientación de los comandos involucrados. Con esta información, resulta más sencillo explorar la documentación de Android y buscar la solución a tus necesidades.

Además de estas páginas, hay que dar valor a la cantidad de canales de YouTube encargados de distribuir información sobre programación. Entre ellos quiero destacar uno que me parece bastante completo.

CodingWithMitch [ref. 28]. Mitch Tabian, propietario del canal, es un programador y desarrollador de Android que se dedica a la impartición de cursos online en su canal de YouTube y en su página web. Incluye guías sobre cómo usar *GraphView*, explicaciones muy buenas del trabajo con bases de datos *SQLite* y diferentes vídeos en los que trata el almacenamiento en la nube entre otras cosas. Tiene varios niveles y programas de más y menos complejidad incluyendo series enteras en las que desarrolla grandes aplicaciones como, por ejemplo, un clon de Instagram.

Para terminar este apartado, incluyo una pequeña lista de atajos de teclados que me han parecido útiles. Se puede encontrar la lista completa en la página de

desarrollador de Android, pero esta recopilación quizá le resulte útil al futuro desarrollador de Android Studio:

1) Control + Alt + Y — Sincronizar, para cuando el proyecto no se ha sincronizado automáticamente o se acaba de añadir una biblioteca.
2) Alt + Enter — Corrección rápida del proyecto. Explicado en el apartado 3.7.
3) Control + Alt + I — Aplicar Sangrías Automáticas. Útil cuando se reestructura la aplicación y las sangrías se han desorganizado.
4) Control + O — Override Methods. Explicado en el apartado 3.7.
5) Control + Alt + O — Optimizar importaciones. Elimina las importaciones no utilizadas en el proyecto. Útil cuando se retira código que incluye clases que se han encontrado innecesarias. Se puede hacer a mano, pero esto es más rápido

6 Conclusiones y futuros desarrollos

En este trabajo de fin de grado, he desarrollado un sistema capaz de grabar sonidos cardíacos que entran por la entrada del micrófono y reproducirlos sonora y visualmente mediante una gráfica. En total, se han generado dos programas distintos para poder mostrar de manera íntegra los resultados que se obtienen. El primero reproduce un audio previamente grabado y el segundo ofrece la posibilidad de realizar grabaciones al momento. En definitiva, el objetivo de

Además, se ha realizado una guía resumida, pero, se desea, clara y profunda que permitirá el desarrollo de más aplicaciones en entorno Android dentro de este proyecto y de otros proyectos del departamento TEISA.

La aplicación aún está en una de sus primeras fases y tiene diferentes posibilidades de mejora. Entre los futuros desarrollos que se pueden buscar están la generación de un sistema de memoria robusto; la generación de herramientas que permitan el envío de grabaciones a otros dispositivos; el desarrollo de un analizador que busque afecciones cardíacas; algún sistema que aumente el volumen de la reproducción para facilitar la audición a personal médico que tenga este sentido deteriorado (recomiendo que la persona que se encargue de esto eche un vistazo a la herramienta de Android ‘VolumeShaper’ aunque requiere API superior a 26); el desarrollo de una interfaz de usuario más atractiva al usuario usando herramientas como ‘RecyclerView’ o ‘ListView’; o la creación de un sistema de conexión específico entre el fonendoscopio y el móvil. Como se comentó en la introducción, el fonendoscopio se conectaba como un micrófono Bluetooth; existen formas de generar una vinculación especial de dispositivo y aplicación, pero requiere no solo trabajar en Android Studio, sino también modificar el sistema Bluetooth de la FPGA.

Por tanto, se espera que este trabajo sirva para un futuro desarrollo exitoso y un producto final atractivo que consiga convertirse en un producto competitivo y demandado.

7 Bibliografía

1. *https://developer.android.com/studio*
2. Google I/O 2013. Presentación de Android Studio. https://www.youtube.com/watch?v=lmv1dTnhLH4&t=446s
3. https://www.jetbrains.com/idea/
4. https://appinventor.mit.edu/
5. http://www.unesco.org/new/fileadmin/MULTIMEDIA/HQ/CI/CI/pdf/In_Focus/mit_app-inventor.pdf
6. https://appinventor.pevest.com/2014/10/23/some-history-behind-app-inventor/
7. https://www.appsheet.com
8. http://www.cs.cornell.edu/home/praveen/praveen.html
9. https://www.linkedin.com/in/praveenseshadri
10. https://twitter.com/pravse
11. https://en.wikipedia.org/wiki/AppSheet
12. *https://developer.android.com/studio/install?hl=ES*
13. https://eu.udacity.com/
14. https://blog.stylingandroid.com/
15. https://material.io/
16. https://developer.android.com/reference
17. https://www.sqlite.org/
18. http://soundfile.sapp.org/doc/WaveFormat/
19. http://www.topherlee.com/software/pcm-tut-wavformat.html
20. https://www.apache.org/licenses/LICENSE-2.0
21. https://gist.github.com/kmark
22. New Venture Software. https://www.newventuresoftware.com/
23. GraphView. https://github.com/jjoe64/GraphView

24. “Android 100%” de Ramón Invarato Menéndez con la colaboración de Ricardo Moya García y Jesús Alberto Casero Gutiérrez. Jarroba.com
25. Curso Udacity. Android Basics, dividido en 5 partes. Por Katherine Kuan, Kunal Chawla, Lyla Fuijiwara y Jessica Lin.
26. https://github.com/
27. https://stackoverflow.com/
28. https://codingwithmitch.com/

Anexos

En estos anexos se incluyen los códigos fuentes de las aplicaciones desarrolladas. Los archivos con dichos códigos están también incluidos en el CD de este trabajo.

1 Código Aplicación Reproductor con Gráfica

1.1 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button
            android:id="@+id/btPreparar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Prepara"
            />

        <Button
            android:id="@+id/btPlay"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Play"/>

    </LinearLayout>



    <com.jjoe64.graphview.GraphView
        android:id="@+id/grafica"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />


</LinearLayout>
```

1.2 MainActivity.java

```
package android.example.reproductorcongrafica;

import android.Manifest;
import android.content.pm.PackageManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;

import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.ShortBuffer;
import java.util.Arrays;

public class MainActivity extends AppCompatActivity {

    // Declaramos las variables que se van a utilizar
    Button btPreparar;
    Button btPlay;
    GraphView grafica;
    String pathArchivo = "";
    MediaPlayer mediaPlayer;
    public short[] muestras;
    LineGraphSeries<DataPoint> mSeries;
    private final Handler mHandler = new Handler();
    private Runnable mTimer;
    private double graphLastValue = 1;
    private int desp = 0;
    final int REQUEST_PERMISSION_CODE = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Ejecuciones al iniciar el programa
        // Reconocemos las variables del entorno y las aplicamos a
variables de Java
        btPreparar = (Button) findViewById(R.id.btPreparar);
        btPlay = (Button) findViewById(R.id.btPlay);
        grafica = (GraphView) findViewById(R.id.grafica);

        // Configuramos la vista de nuestra gráfica
```

```
        grafica.getViewport().setYAxisBoundsManual(true);
        grafica.getViewport().setMinY(-18000);
        grafica.getViewport().setMaxY(18000);
        grafica.getViewport().setXAxisBoundsManual(true);
        grafica.getViewport().setMinX(0);
        grafica.getViewport().setMaxX(4);

        // Bloqueamos el botón de reproducir hasta que se haya preparado el
sistema
        btPlay.setEnabled(false);

        // Creamos la serie de puntos y la añadimos a la gráfica
        mSeries = new LineGraphSeries<>();
        grafica.addSeries(mSeries);

        // Cuando se pulsa el boton de preparación
        btPreparar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Comprobamos los permisos del dispositivo
                if (revisarPermisoDipositivo()) {
                    // Generación de la dirección del archivo
                    pathArchivo =
Environment.getExternalStorageDirectory().getAbsolutePath()
                            + "/audio.wav";

                    // Obtención del vector de amplitudes
                    try {
                        muestras = getAudioSample();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                    // Preparación del reproductor multimedia
                    mediaPlayer = new MediaPlayer();
                    try {
                        mediaPlayer.setDataSource(pathArchivo);
                        mediaPlayer.prepare();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    // Actuación del sistema cuando finaliza la
reproducción
                    mediaPlayer.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
                        @Override
                        public void onCompletion(MediaPlayer mp) {
                            mediaPlayer.stop();
                            try {
                                mediaPlayer.prepare();
                            } catch (IOException e) {
                                e.printStackTrace();
                            }
                            // Destrucción del Runnable para evitar que se
acumulen y produzcan
                            // la ralentización del sistema
                            mHandler.removeCallbacks(mTimer);
                            // Reactivación del botón de reproducir
                            btPlay.setEnabled(true);
                        }
```

```
                    });
                    // Activación de reproducir y desactivación de preparar
                    btPlay.setEnabled(true);
                    btPreparar.setEnabled(false);
                } else {
                    // Pedimos los permisos del dispositivo
                    pedirpermiso();
                }
            }
        });
        // Cuando se pulsa el botón de reproducir
        btPlay.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Bloqueamos el botón de tocar
                btPlay.setEnabled(false);
                // Comenzamos la reproducción de sonido
                mediaPlayer.start();

                // Generamos los puntos de la gráfica
                mTimer = new Runnable() {
                    @Override
                    public void run() {
                        int tiempo_ms = mediaPlayer.getCurrentPosition();
                        desp = (int) (tiempo_ms * 0.441);
                        desp = (int) (desp - 1600);
                        if (graphLastValue > 40) {
                            graphLastValue = 0;
                            mSeries.resetData(new DataPoint[]{
                                    new DataPoint(graphLastValue, 0)
                            });
                        }
                        for (int i = 1; i < 1600; i++) {
                            graphLastValue += 0.1d;
                            if (desp + i < 0) {
                                mSeries.appendData(new DataPoint(i * 1.0 /
441, 0), false, 4410 * 4);
                            } else {
                                mSeries.appendData(new DataPoint(i * 1.0 /
441, muestras[100 * (i + desp)]), false, 4410 * 4);
                            }
                        }
                        mHandler.postDelayed(mTimer, 1);
                    }
                };
                mHandler.postDelayed(mTimer, 1);
            }
        });
    }

    // Sistema que revisa los permisos del dispositivo
    private boolean revisarPermisoDipositivo() {
        int almacenaje_externo = ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE);
        int record_audio = ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO);
        return almacenaje_externo == PackageManager.PERMISSION_GRANTED &&
                record_audio == PackageManager.PERMISSION_GRANTED;
    }
```

```
    // Sistema que pide los permisos del dispositivo
    private void pedirpermiso() {
        ActivityCompat.requestPermissions(this, new String[]{
                Manifest.permission.WRITE_EXTERNAL_STORAGE,
                Manifest.permission.RECORD_AUDIO
        }, REQUEST_PERMISSION_CODE);
    }

    // Override que gestiona los resultados de la petición de permisos
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        switch (requestCode) {
            case REQUEST_PERMISSION_CODE: {
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)
                    Toast.makeText(this, "Permiso Concedido",
Toast.LENGTH_SHORT).show();
                else
                    Toast.makeText(this, "Permisos necsarios para la
aplicación", Toast.LENGTH_SHORT).show();
            }
            break;
        }
    }

    // Sistema de obtención del vector de amplitudes
    private short[] getAudioSample() throws IOException {
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
        BufferedInputStream bufferedInputStream = new
BufferedInputStream(new FileInputStream(pathArchivo));
        int read;
        byte[] buff = new byte[1024];
        while ((read = bufferedInputStream.read(buff)) > 0) {
            byteArrayOutputStream.write(buff, 0, read);
        }
        byteArrayOutputStream.flush();
        byte[] data = byteArrayOutputStream.toByteArray();
        ShortBuffer sb =
ByteBuffer.wrap(data).order(ByteOrder.LITTLE_ENDIAN).asShortBuffer();
        short[] samples = new short[sb.limit()];
        sb.get(samples);
        int a;
        a = samples.length;
        samples = Arrays.copyOfRange(samples, 45, a);
        return samples;
    }

}
```

1.3 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.example.reproductorcongrafica">
```

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:screenOrientation="reverseLandscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

2 Grabadora WAV

2.1 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    >


    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button
            android:id="@+id/btGrabar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Grabar"/>

        <Button
            android:id="@+id/btParar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Parar" />

        <Button
            android:id="@+id/btPreparar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Prepara"
            />

        <Button
            android:id="@+id/btPlay"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Play"/>

    </LinearLayout>

    <com.jjoe64.graphview.GraphView
        android:id="@+id/grafica"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />


</LinearLayout>
```

2.2 MainActivity.java

```
package android.example.grabadorawav;

import android.media.MediaPlayer;
import android.os.Environment;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.SystemClock;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;

import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.RandomAccessFile;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.ShortBuffer;
import java.util.Arrays;
import java.util.Locale;

public class MainActivity extends AppCompatActivity {

    private static final int PERMISSION_RECORD_AUDIO = 0;
    private static final int PERMISSION_WRITE_EXTERNAL_STORAGE = 1;

    private RecordWaveTask recordTask = null;

    Button btGrabar;
    Button btParar;
    Button btPreparar;
    Button btPlay;
    GraphView grafica;
    String pathArchivo = "";
    MediaPlayer mediaPlayer;
    public short[] muestras;
```

```
    LineGraphSeries<DataPoint> mSeries;
    private final Handler mHandler = new Handler();
    private Runnable mTimer;
    private double graphLastValue = 1;
    private int desp = 0;
    final int REQUEST_PERMISSION_CODE = 1;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        pathArchivo =
Environment.getExternalStorageDirectory().getAbsolutePath()
                + "/grabacion.wav";

        btGrabar = (Button) findViewById(R.id.btGrabar);
        btParar = (Button) findViewById(R.id.btParar);
        btPreparar = (Button) findViewById(R.id.btPreparar);
        btPlay = (Button) findViewById(R.id.btPlay);
        grafica = (GraphView) findViewById(R.id.grafica);

        grafica.getViewport().setYAxisBoundsManual(true);
        grafica.getViewport().setMinY(-18000);
        grafica.getViewport().setMaxY(18000);
        grafica.getViewport().setXAxisBoundsManual(true);
        grafica.getViewport().setMinX(0);
        grafica.getViewport().setMaxX(4);

        btPlay.setEnabled(false);

        mSeries = new LineGraphSeries<>();
        grafica.addSeries(mSeries);

        btGrabar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (ContextCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.RECORD_AUDIO)
                        != PackageManager.PERMISSION_GRANTED) {
                    // Request permission
                    ActivityCompat.requestPermissions(MainActivity.this,
                            new String[]{Manifest.permission.RECORD_AUDIO},
                            PERMISSION_RECORD_AUDIO);
                    return;
                }
                if (ContextCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
                        != PackageManager.PERMISSION_GRANTED) {
                    // Request permission
                    ActivityCompat.requestPermissions(MainActivity.this,
                            new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                            PERMISSION_WRITE_EXTERNAL_STORAGE);
                    return;
                }
                // Permission already available
                btPreparar.setEnabled(false);
                btPlay.setEnabled(false);
```

```
                launchTask();
            }
        });

        //noinspection ConstantConditions
        btParar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!recordTask.isCancelled() && recordTask.getStatus() ==
AsyncTask.Status.RUNNING) {
                    recordTask.cancel(false);
                    btPreparar.setEnabled(true);
                } else {
                    Toast.makeText(MainActivity.this, "Task not running.",
Toast.LENGTH_SHORT).show();
                }
            }
        });

        btPreparar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (ContextCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.RECORD_AUDIO)
                        != PackageManager.PERMISSION_GRANTED) {
                    // Request permission
                    ActivityCompat.requestPermissions(MainActivity.this,
                            new String[]{Manifest.permission.RECORD_AUDIO},
                            PERMISSION_RECORD_AUDIO);
                    return;
                }
                if (ContextCompat.checkSelfPermission(MainActivity.this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
                        != PackageManager.PERMISSION_GRANTED) {
                    // Request permission
                    ActivityCompat.requestPermissions(MainActivity.this,
                            new
String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                            PERMISSION_WRITE_EXTERNAL_STORAGE);
                    return;
                }
                preparar();
            }
        });

        btPlay.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                btPlay.setEnabled(false);
                btGrabar.setEnabled(false);
                reproducir();

            }
        });

        // Restore the previous task or create a new one if necessary
        recordTask = (RecordWaveTask)
getLastCustomNonConfigurationInstance();
        if (recordTask == null) {
            recordTask = new RecordWaveTask(this);
```

```
        } else {
            recordTask.setContext(this);
        }
    }

    private void preparar() {
        // Obtención del vector de amplitudes
        try {
            muestras = getAudioSample();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Preparación del reproductor multimedia
        mediaPlayer = new MediaPlayer();
        try {
            mediaPlayer.setDataSource(pathArchivo);
            mediaPlayer.prepare();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Actuación del sistema cuando finaliza la reproducción
        mediaPlayer.setOnCompletionListener(new
MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                mediaPlayer.stop();
                try {
                    mediaPlayer.prepare();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                // Destrucción del Runnable para evitar que se acumulen y
produzcan
                // la ralentización del sistema
                mHandler.removeCallbacks(mTimer);
                // Reactivación del botón de reproducir
                btPlay.setEnabled(true);
                btGrabar.setEnabled(true);
            }
        });
        // Activación de reproducir y desactivación de preparar
        btPlay.setEnabled(true);
        btPreparar.setEnabled(false);
    };

    private short[] getAudioSample() throws IOException {
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
        BufferedInputStream bufferedInputStream = new
BufferedInputStream(new FileInputStream(pathArchivo));
        int read;
        byte[] buff = new byte[1024];
        while ((read = bufferedInputStream.read(buff)) > 0) {
            byteArrayOutputStream.write(buff, 0, read);
        }
        byteArrayOutputStream.flush();
        byte[] data = byteArrayOutputStream.toByteArray();
        ShortBuffer sb =
ByteBuffer.wrap(data).order(ByteOrder.LITTLE_ENDIAN).asShortBuffer();
```

```
        short[] samples = new short[sb.limit()];
        sb.get(samples);
        int a;
        a = samples.length;
        samples = Arrays.copyOfRange(samples, 45, a);
        return samples;
    }

    private void reproducir() {
        // Comenzamos la reproducción de sonido
        mediaPlayer.start();

        // Generamos los puntos de la gráfica
        mTimer = new Runnable() {
            @Override
            public void run() {
                int tiempo_ms = mediaPlayer.getCurrentPosition();
                desp = (int) (tiempo_ms * 0.441);
                desp = (int) (desp - 1600);
                if (graphLastValue > 40) {
                    graphLastValue = 0;
                    mSeries.resetData(new DataPoint[]{
                            new DataPoint(graphLastValue, 0)
                    });
                }
                for (int i = 1; i < 1600; i++) {
                    graphLastValue += 0.1d;
                    if (desp + i < 0) {
                        mSeries.appendData(new DataPoint(i * 1.0 / 441, 0),
false, 4410 * 4);
                    } else {
                        mSeries.appendData(new DataPoint(i * 1.0 / 441,
muestras[100 * (i + desp)]), false, 4410 * 4);
                    }
                }
                mHandler.postDelayed(mTimer, 1);
            }
        };
        mHandler.postDelayed(mTimer, 1);
    };

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        switch (requestCode) {
            case PERMISSION_RECORD_AUDIO:
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    // Permission granted
                    launchTask();
                } else {
                    // Permission denied
                    Toast.makeText(this, "\uD83D\uDE41",
Toast.LENGTH_SHORT).show();
                }
                break;
        }
    }

    private void launchTask() {
```

```
        switch (recordTask.getStatus()) {
            case RUNNING:
                Toast.makeText(this, "Task already running...",
Toast.LENGTH_SHORT).show();
                return;
            case FINISHED:
                recordTask = new RecordWaveTask(this);
                break;
            case PENDING:
                if (recordTask.isCancelled()) {
                    recordTask = new RecordWaveTask(this);
                }
        }

        File wavFile = new File(pathArchivo);
        Toast.makeText(this, wavFile.getAbsolutePath(),
Toast.LENGTH_LONG).show();
        recordTask.execute(wavFile);
    }

    @Override
    public Object onRetainCustomNonConfigurationInstance() {
        recordTask.setContext(null);
        return recordTask;
    }

    private static class RecordWaveTask extends AsyncTask<File, Void,
Object[]> {

        // Configure me!
        private static final int AUDIO_SOURCE =
MediaRecorder.AudioSource.MIC;
        private static final int SAMPLE_RATE = 44100; // Hz
        private static final int ENCODING = AudioFormat.ENCODING_PCM_16BIT;
        private static final int CHANNEL_MASK =
AudioFormat.CHANNEL_IN_MONO;
        //

        private static final int BUFFER_SIZE = 2 *
AudioRecord.getMinBufferSize(SAMPLE_RATE, CHANNEL_MASK, ENCODING);

        private Context ctx;

        private RecordWaveTask(Context ctx) {
            setContext(ctx);
        }

        private void setContext(Context ctx) {
            this.ctx = ctx;
        }

        /**
         * Opens up the given file, writes the header, and keeps filling it
with raw PCM bytes from
         * AudioRecord until it reaches 4GB or is stopped by the user. It
then goes back and updates
         * the WAV header to include the proper final chunk sizes.
         *
         * @param files Index 0 should be the file to write to
         * @return Either an Exception (error) or two longs, the filesize,
```

```
elapsed time in ms (success)
         */
        @Override
        protected Object[] doInBackground(File... files) {
            AudioRecord audioRecord = null;
            FileOutputStream wavOut = null;
            long startTime = 0;
            long endTime = 0;

            try {
                // Open our two resources
                audioRecord = new AudioRecord(AUDIO_SOURCE, SAMPLE_RATE,
CHANNEL_MASK, ENCODING, BUFFER_SIZE);
                wavOut = new FileOutputStream(files[0]);

                // Write out the wav file header
                writeWavHeader(wavOut, CHANNEL_MASK, SAMPLE_RATE,
ENCODING);

                // Avoiding loop allocations
                byte[] buffer = new byte[BUFFER_SIZE];
                boolean run = true;
                int read;
                long total = 0;

                // Let's go
                startTime = SystemClock.elapsedRealtime();
                audioRecord.startRecording();
                while (run && !isCancelled()) {
                    read = audioRecord.read(buffer, 0, buffer.length);

                    // WAVs cannot be > 4 GB due to the use of 32 bit
unsigned integers.
                    if (total + read > 4294967295L) {
                        // Write as many bytes as we can before hitting the
max size
                        for (int i = 0; i < read && total <= 4294967295L;
i++, total++) {
                            wavOut.write(buffer[i]);
                        }
                        run = false;
                    } else {
                        // Write out the entire read buffer
                        wavOut.write(buffer, 0, read);
                        total += read;
                    }
                }
            } catch (IOException ex) {
                return new Object[]{ex};
            } finally {
                if (audioRecord != null) {
                    try {
                        if (audioRecord.getRecordingState() ==
AudioRecord.RECORDSTATE_RECORDING) {
                            audioRecord.stop();
                            endTime = SystemClock.elapsedRealtime();
                        }
                    } catch (IllegalStateException ex) {
                        //
                    }
```

```
                    if (audioRecord.getState() ==
AudioRecord.STATE_INITIALIZED) {
                        audioRecord.release();
                    }
                }
                if (wavOut != null) {
                    try {
                        wavOut.close();
                    } catch (IOException ex) {
                        //
                    }
                }
            }

            try {
                // This is not put in the try/catch/finally above since it
needs to run
                // after we close the FileOutputStream
                updateWavHeader(files[0]);
            } catch (IOException ex) {
                return new Object[] { ex };
            }

            return new Object[] { files[0].length(), endTime - startTime };
        }

        /**
         * Writes the proper 44-byte RIFF/WAVE header to/for the given
stream
         * Two size fields are left empty/null since we do not yet know the
final stream size
         *
         * @param out         The stream to write the header to
         * @param channelMask An AudioFormat.CHANNEL_* mask
         * @param sampleRate  The sample rate in hertz
         * @param encoding    An AudioFormat.ENCODING_PCM_* value
         * @throws IOException
         */
        private static void writeWavHeader(OutputStream out, int
channelMask, int sampleRate, int encoding) throws IOException {
            short channels;
            switch (channelMask) {
                case AudioFormat.CHANNEL_IN_MONO:
                    channels = 1;
                    break;
                case AudioFormat.CHANNEL_IN_STEREO:
                    channels = 2;
                    break;
                default:
                    throw new IllegalArgumentException("Unacceptable
channel mask");
            }

            short bitDepth;
            switch (encoding) {
                case AudioFormat.ENCODING_PCM_8BIT:
                    bitDepth = 8;
                    break;
                case AudioFormat.ENCODING_PCM_16BIT:
                    bitDepth = 16;
```

```
                    break;
                case AudioFormat.ENCODING_PCM_FLOAT:
                    bitDepth = 32;
                    break;
                default:
                    throw new IllegalArgumentException("Unacceptable
encoding");
            }

            writeWavHeader(out, channels, sampleRate, bitDepth);
        }

        /**
         * Writes the proper 44-byte RIFF/WAVE header to/for the given
stream
         * Two size fields are left empty/null since we do not yet know the
final stream size
         *
         * @param out        The stream to write the header to
         * @param channels   The number of channels
         * @param sampleRate The sample rate in hertz
         * @param bitDepth   The bit depth
         * @throws IOException
         */
        private static void writeWavHeader(OutputStream out, short
channels, int sampleRate, short bitDepth) throws IOException {
            // Convert the multi-byte integers to raw bytes in little
endian format as required by the spec
            byte[] littleBytes = ByteBuffer
                    .allocate(14)
                    .order(ByteOrder.LITTLE_ENDIAN)
                    .putShort(channels)
                    .putInt(sampleRate)
                    .putInt(sampleRate * channels * (bitDepth / 8))
                    .putShort((short) (channels * (bitDepth / 8)))
                    .putShort(bitDepth)
                    .array();

            // Not necessarily the best, but it's very easy to visualize
this way
            out.write(new byte[]{
                    // RIFF header
                    'R', 'I', 'F', 'F', // ChunkID
                    0, 0, 0, 0, // ChunkSize (must be updated later)
                    'W', 'A', 'V', 'E', // Format
                    // fmt subchunk
                    'f', 'm', 't', ' ', // Subchunk1ID
                    16, 0, 0, 0, // Subchunk1Size
                    1, 0, // AudioFormat
                    littleBytes[0], littleBytes[1], // NumChannels
                    littleBytes[2], littleBytes[3], littleBytes[4],
littleBytes[5], // SampleRate
                    littleBytes[6], littleBytes[7], littleBytes[8],
littleBytes[9], // ByteRate
                    littleBytes[10], littleBytes[11], // BlockAlign
                    littleBytes[12], littleBytes[13], // BitsPerSample
                    // data subchunk
                    'd', 'a', 't', 'a', // Subchunk2ID
                    0, 0, 0, 0, // Subchunk2Size (must be updated later)
            });
```

```
        }

        /**
         * Updates the given wav file's header to include the final chunk
sizes
         *
         * @param wav The wav file to update
         * @throws IOException
         */
        private static void updateWavHeader(File wav) throws IOException {
            byte[] sizes = ByteBuffer
                    .allocate(8)
                    .order(ByteOrder.LITTLE_ENDIAN)
                    // There are probably a bunch of different/better ways
to calculate
                    // these two given your circumstances. Cast should be
safe since if the WAV is
                    // > 4 GB we've already made a terrible mistake.
                    .putInt((int) (wav.length() - 8)) // ChunkSize
                    .putInt((int) (wav.length() - 44)) // Subchunk2Size
                    .array();

            RandomAccessFile accessWave = null;
            //noinspection CaughtExceptionImmediatelyRethrown
            try {
                accessWave = new RandomAccessFile(wav, "rw");
                // ChunkSize
                accessWave.seek(4);
                accessWave.write(sizes, 0, 4);

                // Subchunk2Size
                accessWave.seek(40);
                accessWave.write(sizes, 4, 4);
            } catch (IOException ex) {
                // Rethrow but we still close accessWave in our finally
                throw ex;
            } finally {
                if (accessWave != null) {
                    try {
                        accessWave.close();
                    } catch (IOException ex) {
                        //
                    }
                }
            }
        }

        @Override
        protected void onCancelled(Object[] results) {
            // Handling cancellations and successful runs in the same way
            onPostExecute(results);
        }

        @Override
        protected void onPostExecute(Object[] results) {
            Throwable throwable = null;
            if (results[0] instanceof Throwable) {
                // Error
                throwable = (Throwable) results[0];
                Log.e(RecordWaveTask.class.getSimpleName(),
```

```
throwable.getMessage(), throwable);
            }

            // If we're attached to an activity
            if (ctx != null) {
                if (throwable == null) {
                    // Display final recording stats
                    double size = (long) results[0] / 1000000.00;
                    long time = (long) results[1] / 1000;
                    Toast.makeText(ctx, String.format(Locale.getDefault(),
"%.2f MB / %d seconds",
                            size, time), Toast.LENGTH_LONG).show();
                } else {
                    // Error
                    Toast.makeText(ctx, throwable.getLocalizedMessage(),
Toast.LENGTH_LONG).show();
                }
            }
        }
    }
}
```

2.3 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.example.grabadorawav" >

    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity"
            android:screenOrientation="reverseLandscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```