UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

HIIT Workouts: personaliza tu entrenamiento desde tu Smartphone

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Durá Alonso

Tutor: David de Andrés Martínez

2015/2016

Resumen

En este proyecto desarrollo una aplicación para realizar ejercicios de tipo HIIT, una nueva forma de hacer ejercicio que se basa en alternar intervalos cortos de alta intensidad con intervalos más largos de una intensidad menor.

Hay varias aplicaciones en el mercado que ofrecen ayuda para realizar este tipo de ejercicios, pero ninguna ofrece el grado de personalización que busco ofrecer con mi aplicación, de forma sencilla y clara.

Mi aplicación está orientada a teléfonos Android, e incluirá las últimas tecnologías y guías de diseño ofrecidas por Google y Android en la programación de aplicaciones móvil.

Palabras clave: Android, Smartphone, deporte, HIIT, entrenamiento.

Abstract

In this proyect I'll be developing an application for HIIT workouts, which is a new way of excercising based in alternating short, high intensity intervals, with longer, softer intervals.

There are several applications already in the market that offer help with Hiit training, but none offer the customization degree that I'm looking to offer with my application, in a simple and clear way.

My application is targeted towards android phones and it will include the latest technologies and design guidelines that Google and Android offer in phone applications development.

Keywords: Android, smartphone, sports, HIIT, workout.

Tabla de contenidos

1. Introducción 7
1.1. Contexto 7
1.2. Motivación 8
1.3. Objetivo 8
1.4. Trabajo Realizado 9
1.5. Estructura de la Memoria 9
2. Estado del mercado 10
2.7. Tabla comparativa 12
3. Especificación 12
3.1. Capa de Presentación 12
3.2. Capa de Datos 13
3.3. Capa de Casos de Uso 15
4. Diseño 16
4.1. Persistencia de datos 16
4.2. Interfaz 16
5. Implementación 21
5.1. Actividad Principal 21
5.2. Objeto HiitItem 25
5.3. Actividad de Ajustes 27
5.3. Actividad de Ejercicio 29
5.3.1. Toolbar button – Iniciar ejercicio 35
5.4. Base de Datos 38
5.5. Actividad de Historia 40
6. Resultados – Tests de usabilidad 44
7. Conclusiones 44
7.1. Conclusiones Técnicas 44
7.2 Conclusiones Personales 45
7.1. Objetivos No Conseguidos 45
7.2. Trabajo FUTURO 45
8. Glosario de Términos 46
9. Bibliografía 48
9.2. Documentación Online 48

Índice de Figuras

Figura 1 Estructura Hiit 8
Figura 2 Tabla Comparativa 12
Figura 3 Codigo SQL para la creacion de la BD con las tablas workout y history 14
Figura 4 Codigo SQL para la eliminacion de una entrada 14
Figura 5 Diagrama de Casos de Uso 16
Figura 6 Diseño de la Actividad principal y de ejercicio 17
Figura 7 Diseño de actividad de Ajustes 18
Figura 8 Prototipo de estructura de datos en XML 19
Figura 9 Estructura conceptual y en BD de Hiits 20
Figura 10 Codigo SQL para cargar un ejercicio 20
Figura 11 Codigo de Intents con informacion extra 20
Figura 12 codigo SQL para buscar entrenamientos 20
Figura 13 Codigo para creacion de botones 22
Figura 14 Tratamiento boton fab 23
Figura 15 Entradas para el NavigationDrawer 24
Figura 16 Tratamiento de items en NavigationDrawer 24
Figura 17 Elemento HiitItem 25
Figura 18 inputType de EditText 26
Figura 19 Constructores clase HiitItem 26
Figura 20 Setters y Getters de HiitItem 26
Figura 21 pref_headers 27
Figura 22 XML ajustes generales 28
Figura 23 Interfaz Ajustes Generales 28
Figura 24 Obtener los sonidos de notificación 29
Figura 25 Interfaz de Ajustes de Notificación 29
Figura 26 Estrucutra del Layout 30
Figura 27 Inicialización de las Variables 30
Figura 28 Constructor MyAdapter 31
Figura 29 Creación del ViewHolder 31
Figura 30 Actualizar el ViewHolder 32
Figura 31 Carga de intervalos por primera vez 33
Figura 32 Creacion de MyAdapter 34
Figura 33 Actualizacion de valores en la base de datos 34
Figura 34 Modificacion del fab listener 35
Figura 35 Tratamiento de Preferncias del Usuario 36
Figura 36 Guardar informacion en la BD 36
Figura 37 Realización del ejercicio 37
Figura 38 método getAllNames 38
Figura 39 método getItems 39
Figura 40 método addWorkoutLive para guardar ejercicio con fecha 39
Figura 41 Método para devolver la historia de ejercicios realizados 40
Figura 42 Método para obtener nombres y fechas 40
Figura 43 Actividad historia 41
Figura 44 Actividad principal de historia 42
Figura 45 Arbol de componentes de Historia 42
Figura 46 CustomFragmentAdapter getItem 42
Figura 47 CustomFragmentPagerAdapter getPageTitle 43

Figura 48 Estructura de fragment_history 43
Figura 49 Creacion del Fragment 44

1. Introducción

1.1. Contexto

Con el auge de los Smartphone, cada vez son más personas las que incorporan las herramientas que éstos ofrecen a sus vidas diarias. De esta forma, atrás han quedado los días en los que solo grandes empresarios tenían una PDA (los "abuelos" de los Smartphone) para recordarles reuniones o llamadas. Todos utilizamos nuestros smartphones tanto para cosas serias como para nimiedades del día a día. Desde aplicaciones tan simples como una lista de la compra, pasando por aplicaciones más complejas como convertir tu teléfono en un mando a distancia para tu pc, es innegable que utilizamos nuestro Smartphone para todo. Uno de los sectores en los que está teniendo un gran éxito su uso es el del fitness.

Para comprobar el éxito de las aplicaciones fitness debemos retroceder hasta el inicio del mercado de "apps". En Julio de 2008, Apple lanzó la "App Store" con tan solo 500 aplicaciones, que fueron descargadas 10 millones de veces en solo la primera semana [1]. Estos números no hicieron más que aumentar con el tiempo, habiendo 10.000 aplicaciones disponibles tan solo 4 meses más tarde, pasando por 100.000 un año más tarde, hasta llegar a los 2.000.000 apps actuales [2]. En Octubre del 2008, Android lanzó su Android Market, con tan solo 50 aplicaciones disponibles. Siguiendo un aumento semejante, aunque menor al principio, que el de su competidora Apple, actualmente podemos encontrar 2.200.000 apps en el Android Market (Ahora llamado Play Store) [2].

A pesar de que ya existían comunidades en internet considerablemente grandes como MyFitnessPal [3], con 75 millones de usuarios, una de las primeras apps de "fitness" en aparecer fue "MapMyFitness", compañía subsidiaria de MyFitnessPal y que daría lugar a otras compañías como MapMyRide, MapMyRun o MapMyWalk [4]. Esta aplicación está integrada ahora en más de 400 gadgets de seguimiento deportivo, y almacena más de 160 millones de rutas. Desde esta aplicación ha llovido mucho, y ahora podemos encontrar miles de aplicaciones, muchas de ellas respaldadas por grandes empresas como Google, Nike o Adidas, lo que hace que el mundo de las apps de fitness sea una industria muy productiva, llegando incluso a mover catorce mil millones de dólares el año 2015 [5]. Como ejemplo, MyFitnessPal fue adquirida por UnderArmour, famosa compañía de ropa deportiva, en 2013 por 150 millones de dólares.

Por 2009 las apps de fitness evolucionan y ya no son solo aplicaciones, sino que se combinan con gadgets externos para ofrecer una experiencia más cercana a la realidad. En 2009, la compañía francesa Withings anunció una báscula electrónica con Wi-Fi que sincronizaba tu peso con tu Smartphone, y este calculaba peso, BMI, etc.

A partir de 2011 se produce una explosión de dispositivos como pulseras, camisetas inteligentes, raquetas inteligentes e incluso calcetines inteligentes. Todo esto se sincroniza con aplicaciones (tanto propias como genéricas) en los smartphones de los usuarios, con lo que se obtienen datos mucho más reales y precisos [4] [6].

1.2. Motivación

A pesar de toda esta cantidad de aplicaciones que cubren necesidades muy variadas, el mundo del fitness avanza muy rápido, y siempre hay algún descubrimiento nuevo que aún no está cubierto por estas apps existentes. Este es el caso que nos ocupa; Desde hace relativamente poco, está cobrando fuerza un tipo de entrenamiento basado en intervalos. Este entrenamiento se conoce como Hiit [7], High Intensity Interval Training.

El Hiit es un es un entrenamiento que consiste en alternar ejercicio muy intenso y corto con uno más suave y largo. En este caso, el ejercicio intenso será esprintar a máxima velocidad, y el ejercicio suave andar. Por ejemplo, esprintar durante 30 segundos, andar durante 60 segundos, y repetir unas 8 veces. Se ha demostrado que este ejercicio es mucho más eficaz para la pérdida de grasa que un ejercicio tradicional. Además de esto, al finalizar un HIITs se entra en un estado llamado EPOC (Excess-post excercise Oxigen Consumption) que acelera tu metabolismo y quema más calorías durante incluso 24 horas después de realizar el ejercicio, a diferencia de un ejercicio tradicional donde prácticamente no se quema nada al finalizar el ejercicio. [8]

Un ejercicio Hiit está formado por varios intervalos. Cada intervalo posee un número de repeticiones de ese intervalo, y dos secciones, una sección sprint y una sección descanso. Cada sección posee a su vez un número de veces a repetir esa sección dentro del intervalo, y un tiempo de duración de esa sección. Esto se puede ver de forma más intuitiva en la Figura 1

Figura 1 Estructura Hiit

1.3. Objetivo

Aunque sí que es cierto que ya existen una gran cantidad de apps dedicadas exclusivamente al Hiit (Buscar Hiit en el Play store ofrece 113 resultados a la fecha de escritura de este trabajo), y otras más amplias (Como Runtastic) que ya han añadido HIITs a sus funciones básicas, no he encontrado ninguna que ofrezca todo lo que busco en una aplicación de estas características.

Un ejercicio HIIT es un ejercicio muy intenso, y por lo tanto, pienso que lo ideal sería encontrar una aplicación que ofreciera un grado de personalización máxima. La gran mayoría de las aplicaciones ofrecen configurar la duración del periodo de estrés o sprint, la duración del periodo de descanso, y el número de repeticiones. Considero que sería muy beneficioso para el usuario tener más control sobre estos tiempos y poder

decidir cuánto dura específicamente cada intervalo. Quizás sería interesante comenzar con unos intervalos de sprint mayores y descanso menor, y según se van realizando iteraciones, disminuir el tiempo del sprint y aumentar el tiempo del descanso.

Este será el pilar básico de mi aplicación, ofrecer al usuario un control total y absoluto sobre cómo desea realizar su ejercicio.

El funcionamiento será sencillo, el usuario podrá escoger de una lista de HIITs, que él mismo podrá modificar, el ejercicio que quiera hacer, y una vez lo inicie, mediante auriculares, la aplicación le indicará al usuario cuando debe esprintar o descansar, además de otra información que el usuario podrá elegir recibir o no, como número de intervalos restantes, número de intervalos realizadas, tiempo restante, tiempo transcurrido, etc.

A diferencia de un ejercicio de running tradicional, en el HIIT el tiempo que se tarda en realizar un ejercicio está pre-definido, y la velocidad no es un factor decisivo, sino el esfuerzo; por ejemplo, es más eficaz realizar la sección de esfuerzo más lenta sobre una rampa, que más rápida sobre una superficie plana. Es por esto que no planeo incluir funcionalidad de envío de estados en Facebook u otras redes sociales, puesto que no hay una métrica más allá de la duración pre-definida de un ejercicio que pudiera ser interesante compartir.

Cuando el usuario realice el ejercicio se le dará información a través de los auriculares o vibración del móvil para que el usuario sepa cuando tiene que cambiar de ritmo.

Las demás funciones que mi aplicación incorporará son aquellas que nos encontramos en la gran mayoría de las aplicaciones de deporte, con algunos cambios dada la naturaleza de los HIITs. Planeo incluir un historial de ejercicios, por el que el usuario pueda ver un histórico con los ejercicios que ha realizado. Se podrá configurar el tipo de aviso que el usuario quiera recibir sobre los intervalos (un pitido, silbido, voz, etc.), así como la posibilidad de añadir una cuenta atrás hasta el siguiente cambio de intervalo.

1.4. Trabajo Realizado

Para la creación de esta aplicación móvil realicé una serie de reuniones con el tutor, en las que determinamos qué tipo de interfaz le ofreceríamos al usuario final, y que tecnologías de las que ofrece Android eran las más adecuadas para esto.

Una vez tuvimos clara la estructura visual que queríamos ofrecerle al usuario mediante bocetos, y teniendo en cuenta que uno de los objetivos de esta aplicación era ofrecerle al usuario una experiencia sencilla llegamos a la conclusión de que lo óptimo sería programar esta aplicación teniendo en mente un objetivo de Android SDK 24 para así poder utilizar las últimas aportaciones de Google al diseño en Android, como Floating Action Buttons, RecyclerViews o Fragments.

En cuanto a la persistencia de datos, optamos por elegir un almacenamiento basado en base de datos SQLite3, por ser la opción más rápida y menos pesada.

1.5. Estructura de la Memoria

En primer lugar, realizaré un estudio del mercado, para ver qué características ofrecen las aplicaciones ya existentes y así encontrar que funcionalidades debería ofrecer mi aplicación para destacar sobre las demás.

Sabiendo esto, hablaré sobre estas funcionalidades que mi aplicación deberá implementar, y las decisiones que he tomado para implementarlas.

Más adelante, entraré en detalle tanto en el diseño de la aplicación como en la implementación. En cuanto al diseño hablaré tanto del diseño visual de la aplicación, como del diseño de la base de datos que he elegido. En la implementación de la aplicación, que será la sección más amplia, añadiré secciones de código interesantes o con las que haya tenido problemas de implementación, así como las soluciones propuestas.

Por último, como conclusión hablaré de los objetivos conseguidos, los objetivos que no haya conseguido implementar y el posible trabajo futuro a realizar sobre la aplicación.

2. Estado del mercado

Como he comentado en la introducción, actualmente hay una gran cantidad de aplicaciones que ofrecen ejercicios HIIT, aunque muchas de ellas solo difieren en la presentación de la interfaz. De todas formas, a la gran mayoría de estas aplicaciones les falta ese grado de personalización (O si lo ofrecen, lo hacen de forma muy confusa para el usuario) que busco con mi aplicación.

Tabata Timer for hiit [9]
"El metodo Tabata consiste en realizar 8 series de 20 segundos cada una, con el mayor número de repeticiones en este tiempo, con 10 segundos de descanso entre ellas. Aplicación fácil de usar - Temporizador Tabata"

Tabata Timer For HIIT es una buena aplicación, bastante completa, para ejercicios HIIT. Ofrece configurar tiempo de preparación (Un calentamiento previo), tiempo de esfuerzo, tiempo de descanso y número de ciclos (Repeticiones). Aparte de esto, permite guardar los días que realizas HIITs y enseña tu rendimiento de una forma bastante intuitiva. También posee una funcionalidad que permite al usuario mantener un control sobre su peso, aunque de forma no automatizada. Le falta la funcionalidad de poder configurar el número de repeticiones y de tiempos dentro de cada intervalo.

Hiit interval training timer [10]
"Our app is a simple and efficient timer that will be your best companion during each and every one of your INTERVAL TRAINING workouts!"

Hiit interval Training timer es una de las aplicaciones HIIT más sencillas que podemos encontrar en el Market. Únicamente posee la funcionalidad que permite al usuario introducir el tiempo de preparación, esfuerzo y descanso, y número de ciclos. No ofrece ningún tipo de sincronización con Facebook, ni seguimiento de peso, ni ninguna otra función que despiste de la funcionalidad principal de los HIITs. Es una aplicación ligera, rápida y eficiente. Provee un menú de configuración muy sencillo que permite configurar el tipo de aviso y la cuenta atrás. Tampoco ofrece configurar las repeticiones ni los tiempos dentro de cada iteración.

HIIT intervalo entrenamiento [11]
"Caynax HIIT - entrenamiento de alta intensidad intervalo"

HIIT intervalo entrenamiento ofrece varias funcionalidades extra. Aparte del claro temporizador y personalización de tiempos de preparación, esfuerzo, descanso y número de ciclos, la aplicación ofrece también una gran cantidad de ejercicios ya pre-grabados para todos los niveles. También permite programar recordatorios para los siguientes entrenamientos, y así tener un seguimiento más completo del ejercicio realizado. En cuanto a los ajustes de la aplicación, permite, como las anteriores, definir el tipo de aviso y tiempos, así como ajustes del teléfono más completos como mantener la pantalla encendida o mostrar notificaciones. No ofrece la funcionalidad de modificar las repeticiones y los tiempos dentro de cada iteración.

Runtastic [12]
"¡Empieza con la app gratis Runtastic GPS Running y Fitness y seremos tu entrenador personal para actividades de running, ciclismo o caminatas vía GPS!"

Runtastic es una de las aplicaciones más completas para corredores que hay en el mercado. Actualmente para acceder a la sección HIIT es necesario ser miembro PRO, lo que requiere pago, pero que también ofrece una gran cantidad de funcionalidades extra (No solo orientadas al HIIT) como estadísticas muy detalladas, rutas pre-definidas, medición de ritmo cardiaco, conexión smartwatch, etc. De todas formas, a pesar de lo interesante de estas características extra, todo esto queda fuera de los objetivos de mi aplicación. Runtastic de pago no ofrece la posibilidad de configurar las repeticiones ni los tiempos dentro de cada iteración.

A HIIT Interval Timer [13]
"Easy to use workout timer for interval training. Create your own sets and workouts!"

A HIIT Interval Timer es la aplicación que mayor posibilidad de personalización ofrece, a pesar de que es algo complejo utilizarla. Las funcionalidades que ofrece son bastante escasas, pero ésta sí que permite configurar cada repetición independientemente dentro de cada intervalo, aunque es cierto que la forma de hacerlo es algo tosca. La aplicación tiene un diseño que no está actualizado a las nuevas directrices de google para la programación en Android, es decir, no sigue Material Design, y navegar por sus abundantes menús para configurar los intervalos HIIT es una experiencia bastante contra-intuitiva.

10 HIIT Workout Calisthenics [14]
"These short, intense workouts provide improved athletic capacity and condition, improved glucose metabolism, and improved fat burning."

10 HIIT Workout Calisthenics es una aplicación interesante, puesto que extiende el entrenamiento HIIT más allá del ámbito del corredor también ofrece entrenamientos para bici, o incluso levantamiento de pesas. Así pues, a pesar de ofrecer una gran cantidad de opciones, se sacrifica la profundidad de cada una de ellas, y no ofrece una gran cantidad de personalización dentro de cada ejercicio.

2.7. Tabla comparativa

Para la tabla comparativa he escogido solo las medidas que planeo implementar con mi aplicación. Así pues, características como conexión con redes sociales o aplicación de Hiits más allá de correr no aparecen aquí.

	Configuración de intervalos independientes	História	Menús sencillos	Interfaz Material Design	Ejercicios Definidos	Pago
Tabata Timer for Hiit	No	Si	No	No	Si	No
Hiit Interval Training Timer	No	No	Si	No	Si	Anuncios
HIIT Intervalo Entrenamiento	No	Si	Si	Si	Si	Más Opciones
Runtastic	No	Si	Si	Si	Si	Si
A Hiit Interval Timer	Si	No	No	No	Si	Más Opciones
10 HIIT Workout Calisthenics	No	No	No	No	Si	No

Figura 2 Tabla Comparativa

Teniendo esto en cuenta, a pesar de que hay aplicaciones que se quedan cerca de tener todas las características, solo una posee la configuración total de intervalos, y se sacrifica intuitividad y sencillez para conseguirlo. Es por esto, que los pilares de mi aplicación serán ofrecer la configuración de intervalos independientes sin sacrificar un diseño que siga las pautas del Material Design de Google, y que por lo tanto sea cómodo y fácil de utilizar para el usuario medio.

3. Especificación

3.1. Capa de Presentación

Por la naturaleza de la aplicación, y puesto que la he querido enfocar de una forma muy directa para únicamente realizar un tipo de ejercicio muy específico, he optado por elegir un diseño de interfaz sencillo y directo, de forma que el usuario no tenga que pasar por una miríada de menús o configuraciones simplemente para realizar un sólo ejercicio de 20 minutos. Es por esto que la aplicación se iniciará directamente mostrando todos los ejercicios disponibles, varios que vendrán pre-definidos y los que el usuario haya querido crear. Los ejercicios se mostrarán como botones en una lista, siendo cada botón un ejercicio diferente, cuyo texto será el nombre que el usuario haya elegido para él.

Siguiendo las guías de diseño de Google para programación en Android más actuales, la aplicación poseerá un floating action button o *fab* [15], que estará presente en la mayoría de las pantallas o actividades. El fab es un botón circular normalmente ubicado en la esquina inferior derecha, que permite al usuario realizar acciones

simples, generalmente de adición. En caso de que el usuario quisiera crear un nuevo Hiit, simplemente tendrá que hacer click en el *fab* desde la actividad principal y la aplicación lo llevará a una nueva actividad de edición que, manteniendo el mismo patrón de diseño que la actividad de correr, le permitirá al usuario añadir o eliminar intervalos. Quiero reutilizar la misma actividad tanto para cuando el usuario selecciona un ejercicio para hacerlo, como para cuando el usuario quiera crear un nuevo ejercicio para que así la experiencia para el usuario sea más unificada y transparente, puesto que para él la aplicación sólo tendrá dos actividades, una en la que selecciona el Hiit a realizar, y otra en la que lo realiza/edita/crea. De esta forma, incluso si el usuario quisiera realizar un Hiit, pero en el último momento quisiera cambiar algún intervalo específico, podría hacerlo directamente desde esa misma actividad.

Desde la ventana principal, de nuevo siguiendo las guías más actuales de google, el usuario podrá acceder deslizando hacia la derecha, en el lado izquierdo de la pantalla, al menú de navegación (Navigation Bar O *NavBar*). Este menú permitirá al usuario entrar a la actividad de configuración, obtener información acerca del entrenamiento Hiit, ver un histórico de su actividad, y por último ver la información sobre la aplicación, como versión, email para enviar sugerencias, etc. En el caso de que en el futuro opte por monetizar la aplicación realizando una versión con anuncios y una versión "pro" sin anuncios, aquí aparecerá un link para comprarla.

La actividad de ajustes, a pesar de ser bastante escasa dada la naturaleza simple y directa de la aplicación y los ejercicios Hiit, seguirá con la filosofía de que el usuario realice deporte de la forma más personalizada posible y le permitirá cierto grado extra de configuración.

Desde esta actividad de ajustes, el usuario podrá configurar el tipo de aviso que quiere recibir cuando el intervalo cambie (Aviso por voz, pitido, silbido, etc.), el número de avisos (segundos) que quiere recibir antes de que el intervalo acabe (Avisar 3 segundos antes, 2 segundos, etc.) o recibir información extra (Por mensaje de voz) en cada cambio de intervalo como la repetición actual en la que se encuentra, tiempo transcurrido, tiempo restante o número de intervalos restantes.

En la actividad de historia, el usuario podrá ver tanto la fecha en la que creó cierto ejercicio, como la fecha en la que lo realizó.

3.2. Capa de Datos

Para almacenar todos los intervalos y Hiits he optado por utilizar el motor de base de datos SQLite3, que viene por defecto en Android y utiliza pocos recursos.

Ya que la aplicación es totalmente offline, toda la información será guardada en el móvil de forma local, por lo que no es necesario conectar la base de datos con ningún servidor externo.

Al iniciar la aplicación se creará una base de datos con una tabla "workout" que tendrá las siguientes columnas: name, date, spinnerGen, spinnerSprint, timeSprint, spinnerRest, timeRest. También se creará una tabla "history" con tan solo dos columnas, "name" y "date".

```
CREATE TABLE workout (name TEXT, date TEXT, spinnerGeneral INTEGER,
spinnerSprint INTEGER, timeSprint INTEGER, spinnerRest INTEGER, timeRest
INTEGER)
"CREATE TABLE history (name TEXT, date TEXT)"
```

Figura 3 Codigo SQL para la creacion de la BD con las tablas workout y history

En la tabla workout, "name" es el nombre del ejercicio que el usuario ha elegido. Inicialmente ideé la aplicación de forma que si el usuario no elige ningún nombre (matando la aplicación antes de salvar, por ejemplo) la aplicación genera uno automáticamente siguiendo esta estructura "Workout_[Año]_[Mes]_[DiaDelMes], aunque más tarde llegué a la conclusión de que pedirle el nombre del ejercicio al usuario justo antes de iniciar la actividad de modificación o creación de la aplicación era más eficaz, y por lo tanto no era necesario generar un nombre automáticamente. Sobre esto hablaré en profundidad más adelante.

Este campo "name" se utilizará para agrupar los diferentes intervalos dentro de cada entrenamiento Hiit, por lo tanto tenemos que asegurarnos que no se repita el mismo nombre para varios ejercicios diferentes.

Siguiendo con las columnas de la base de datos, el atributo date (fecha) se genera automáticamente con la fecha del teléfono con una estructura predefinida [Año] _ [Mes] _ [DíaDelMes] al guardar o editar un entrenamiento. De esta forma, el usuario podrá ver un historial de los últimos ejercicios creados y modificados en el apartado "História" en el menú lateral de la NavBar.

Las columnas spinnerGen, spinnerSprint, timeSprint, spinnerRest y timeRest están asociadas a la información que el usuario puede configurar dentro de cada iteración de cada Hiit. Número de repeticiones de un intervalo, numero de repeticiones del periodo Sprint, duración del sprint, repeticiones del periodo de descanso, y duración del descanso, respectivamente.

Para que el usuario pueda comprobar su histórico de ejercicios realizados se utilizará la tabla "history". Esta tabla añade una entrada nueva cada vez que el usuario realiza un ejercicio guardando el nombre y la fecha actual (En este caso utilizando la hora también). De esta forma en el apartado Historia el usuario puede ver qué día realizó cada ejercicio.

Desde esta pantalla de Historia el usuario puede eliminar las entradas que crea oportuno, simplemente seleccionando la entrada que quiera y apretando en el botón eliminar de la toolbar, en la esquina superior derecha. Esto lanza una petición SQL a la base de datos para que elimine la entrada con la fecha y el nombre seleccionado mediante

```
DELETE FROM history *
WHERE name=[nombre_selecionado]
```

Figura 4 Codigo SQL para la eliminacion de una entrada

Utilizar únicamente el nombre como filtro de eliminación es suficientemente preciso, ya que el nombre no se repetirá en ejercicios diferentes.

3.3. Capa de Casos de Uso

Con todo esto, los casos de uso que se pretenden cubrir con esta aplicación son los siguientes.

Si el usuario desea simplemente iniciar la aplicación y realizar uno de los ejercicios pre-definidos, simplemente tendrá que iniciar la aplicación, seleccionar el ejercicio deseado y apretar en el botón de iniciar ejercicio de la barra de herramientas (Esquina superior derecha), lo que automáticamente iniciará la cuenta atrás

Para crear un nuevo entrenamiento desde cero, haciendo click en el *fab* de la pantalla principal, el usuario entrará en la actividad de creación (Visualmente igual que la actividad de ver el ejercicio), desde aquí, seleccionar otra vez el nuevo *fab* hará aparecer una ventana emergente que permitirá al usuario modificar todos los campos relacionados al Hiit (seleccionar el número de repeticiones tanto del intervalo general, como del sprint y el descanso, así como el tiempo para el sprint y para el descanso). Al guardar los cambios, estos se verán reflejados en la actividad base. Si el usuario selecciona una iteración ya creada volverá a aparecer esta misma ventana emergente (Con los datos que había en la iteración) y desde aquí el usuario podrá modificar lo que considere oportuno. Esto se podrá hacer tanto desde la creación de un ejercicio nuevo como seleccionando un ejercicio ya creado.

Desde la actividad de creación el usuario puede guardar el ejercicio, descartar los cambios o realizar la actividad. Para descartar los cambios se pide confirmación al usuario para asegurar que no ha apretado el botón de descartar por error. Si confirma que quiere descartar, se ignoran todos los cambios y se vuelve a la pantalla principal.

Desde la actividad principal, como ya he comentado antes, haciendo swipe izquierdo para abrir el *NavBar* el usuario podrá acceder también a la pantalla de Ajustes, así como recibir ayuda sobre Hiits y sobre la aplicación.

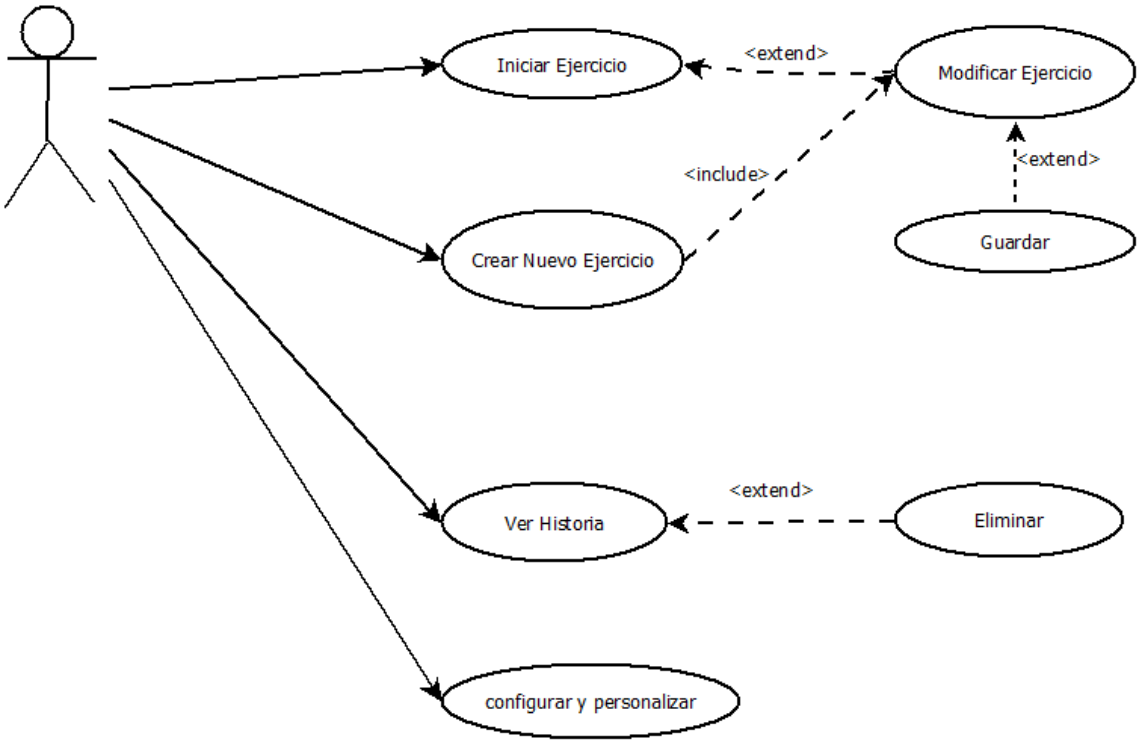

Figura 5 Diagrama de Casos de Uso

4. Diseño

4.2. Interfaz

Como ya he mencionado anteriormente, mi principal objetivo era realizar una aplicación lo más sencilla y directa posible para el usuario final, tanto de peso como de uso. Mi intención era que cualquier persona que esté acostumbrada a utilizar un teléfono Android pueda iniciar esta aplicación y de forma automática, sin necesidad de un tutorial, pueda saber cómo utilizarla. Para esto era imprescindible seguir las guias de diseño de Google en cuanto a Material Design [16].

Aquí detallaré la interfaz visual que he diseñado, que será diferente a la aplicación final, debido a cambios que he considerado oportuno realizar durante la ejecución del trabajo, tanto por motivos de programación, como a consecuencia al realizar pruebas de uso.

La actividad principal y la actividad de ejercicio se pueden apreciar en la Figura 6.

En la pantalla principal el usuario puede encontrar lo que se podría esperar de una aplicación Google cualquiera. Un floating Action Button (o *fab*) [15] con el icono de añadir, y un Navigation Drawer [17] accesible mediante swipe derecho o selecionando el “hamburger menú”.

En la pantalla principal aparecerá una lista con botones, que representarán los ejercicios que el usuario ya haya creado. Tanto si el usuario selecciona un ejercicio ya creado, como si selecciona crear uno nuevo, la actividad que se lanza es la misma.

Una vez dentro de la actividad que contiene los diferentes intervalos de un ejercicio, se mantiene el *fab*. Los intervalos son cada uno un CardView [18], que implementan de forma nativa los conceptos de elevación y sombras de Google.

Para utilizar además las animaciones recomendadas por Google, será necesario utilizar un elemento Coordinator Layout [19] que contenga a la barra de herramientas, el floating action button y a un RecyclerView [20], que es el contenedor donde las CardViews (intervalos) se guardarán. A esto entraremos en mucho más detalle más adelante en la sección de implementación (apartado 5.3 Actividad de Ejercicio), pero en la Figura 26 se puede apreciar esta estructura. La funcionalidad de este Coordinator Layout es, como su propio nombre indica, coordinar los elementos que contenga dentro de sí, para que las animaciones entre ellos sean coherentes [21]. Con esto conseguimos efectos como que al hacer scroll hacia abajo en la actividad que muestra los intervalos, tanto el fab como la barra de herramientas superior se escondan de manera automática y coordinada, para ocultar lo menos posible de la lista de intervalos/cardviews. Si dejamos de hacer scroll el floating action button y la barra de herramientas se mantienen ocultos, pero si hacemos scroll en dirección contraria vuelven a aparecer.

Figura 6 Diseño de la Actividad principal y de ejercicio

Como se puede ver en la Figura 7 Diseño de actividad de Ajustes, en el menú lateral al que se puede acceder desde la pantalla principal encontramos las entradas de Ajustes, Ayuda y Acerca de.

Ayuda y Acerca de simplemente harán aparecer un menú emergente que explicará qué es un ejercicio Hiit y como realizarlo, e información acerca de la aplicación y donde mandar sugerencias y quejas, respectivamente.

Desde la actividad de ajustes se ofrecerá una serie de configuraciones al usuario como el tipo de aviso a recibir, los segundos con antelación con los que quiere que se le avise antes del cambio de intervalo, la información extra que quiere escuchar, etc.

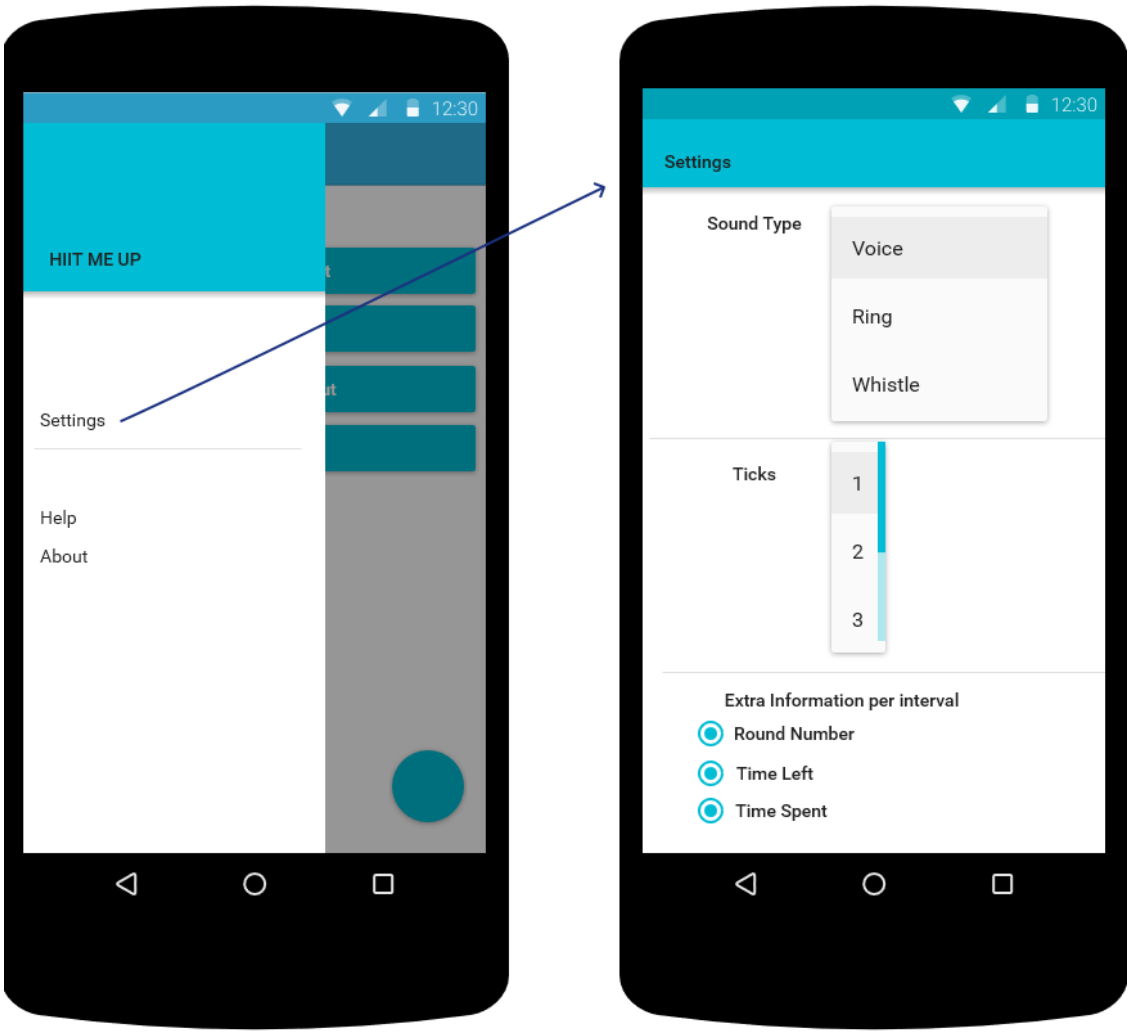

Figura 7 Diseño de actividad de Ajustes

4.1. Persistencia de datos

Para guardar los datos había dos opciones viables. O bien utilizar bases de datos, o bien utilizar almacenamiento externo/interno en ficheros XML.

La opción de utilizar ficheros XML era interesante por la intuitiva forma en la que se almacenaba y accedía a los datos. Utilizando una estructura como la que podemos ver en la Figura 88 se podría ver a primera vista los valores guardados y la estructura que siguen.

```
<Hiit name=”nombre”>
	<fecha=”fecha_de_hoy”/>
	<spinnerGen=”1”>
		<spinnerSprint=”1”/>
		<timeSprint=”30”/>
		<spinnerRest=”1”/>
		<timeRest=”50”/>
	</spinnerGen>
	<spinnerGen=”5”>
		<spinnerSprint=”1”/>
		<timeSprint=”25”/>
		<spinnerRest=”1”/>
		<timeRest=”55”/>
	</spinnerGen>
</Hiit>
```

Figura 8 Prototipo de estructura de datos en XML

Un Hiit con nombre engloba una fecha y uno o varios spinner general (que representa el número de repeticiones), que a su vez engloba a dos spinners y tiempos. Aun así, esta aproximación presentaba varios problemas. Tamaño de archivos generados (el número de líneas aumenta en 9 por cada Hiit nuevo que el usuario genere) y lidiar con permisos de escritura y lectura para guardar este archivo XML generado en almacenamiento interno o externo. Esto último requeriría pedirle al usuario permisos de escritura y lectura en almacenamiento al instalar y ejecutar la aplicación. Por estos dos motivos, con el objetivo de simplificar la experiencia al máximo para el usuario, opté por la aproximación basada en bases de datos, que genera algún problema más pero sólo de cara a la programación, y ninguno de cara al usuario.

La dificultad que conlleva trabajar con bases de datos está únicamente relacionada con el agrupamiento de los Hiits. Lo lógico sería definir la columna “Name” que está relacionada con el atributo que identifica a cada ejercicio (y que por lo tanto es único), como la columna de identificación de la base de datos. Con esto la base de datos se encargaría de forma nativa de que no haya varias entradas con un mismo nombre repetido. Por desgracia, en la base de datos, al poder tener cada entrenamiento un número indefinido de intervalos, cada fila no es un ejercicio, sino una iteración. Ahora el nombre de cada entrenamiento, a pesar de tener que ser en teoría un nombre único no repetido, no podemos hacerlo con una restricción de base de datos porque se deberá repetir para diferentes intervalos dentro de un mismo ejercicio, pero no para diferentes ejercicios. Esto se ve de forma más intuitiva en la Figura 99.

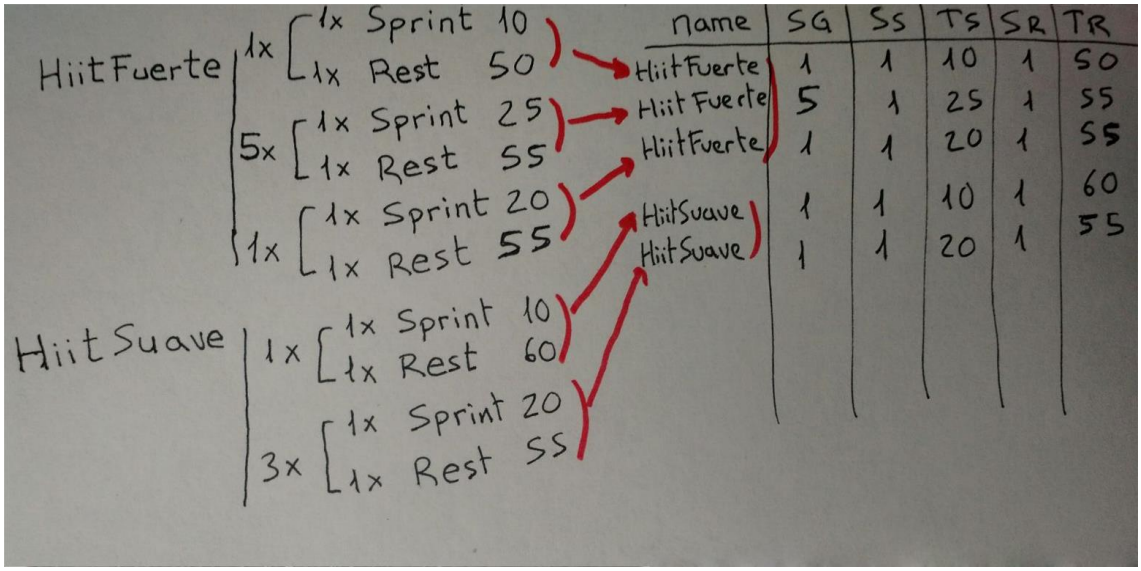

Figura 9 Estructura conceptual y en BD de Hiits

HiitFuerte y HiitSuave son dos ejercicios diferentes que deben tener dos entradas diferentes en la pantalla principal. HiitFuerte tiene tres intervalos, uno que se repite una vez, uno que se repite cinco veces, y un último que se repite una vez, por lo que al entrar a, por ejemplo, HiitFuerte, tendremos que cargar todas las filas de la base de datos que tengan de nombre HiitFuerte mediante una simple consulta SQL.

```
SELECT spinnerGeneral, spinnerSprint, timeSprint, spinnerRest,
timeRest FROM workout WHERE name = HiitFuerte
```
Figura 10 Codigo SQL para cargar un ejercicio

El nombre del ejercicio lo obtenemos añadiéndolo como extra al intent que lanza la actividad desde la pantalla principal, que lo obtendremos o bien del propio texto del botón si ya está creado el ejercicio, o del menú emergente que le pide el nombre al usuario al crear un ejercicio nuevo. En la Figura 11 podemos ver el código por el que se coge el nombre del botón y se añade al Intent que lanzará la actividad.

```
Intent intent = new Intent(MainActivity.this, HiitContainer.class);
workoutName = button.getText().toString();
intent.putExtra("workoutName",workoutName);
startActivity(intent);
```
Figura 11 Codigo de Intents con informacion extra

Por esto la verificación que se realizará para no repetir nombres de ejercicio deberá ser a través de código, y no por base de datos. Cuando el usuario introduce el nombre para el nuevo entrenamiento, buscamos en la base de datos con una consulta sencilla para verificar que no existe.

```
SELECT FROM workout WHERE name = <nombre entrenamiento>
```
Figura 12 codigo SQL para buscar entrenamientos

Como realizamos esto solo cuando el usuario vaya a crear un ejercicio nuevo, y le pedimos el nombre al usuario, y lanzamos esta consulta antes de lanzar la nueva actividad, no hay riesgo de que la consulta nos devuelva la actividad que acabamos de crear o que estamos editando.

Si esta consulta nos devuelve algo, significa que el nombre ya existe en la base de datos, y por lo tanto la aplicación le pedirá al usuario un nombre nuevo.

5. Implementación

5.1. Actividad Principal

Para la actividad principal, he decidido seguir una estructura de Navigation Drawer Activity con Floating Action Button (*fab*). Esto facilita que el usuario sepa ya desde el principio las funcionalidades básicas que la aplicación ofrece de forma intuitiva, al mantener una estructura similar a todas las aplicaciones nativas Android.

En esta primera ventana serán visibles los ejercicios que el usuario haya definido (Más unos básicos ya creados que vienen pre-cargados) en forma de botones. Estos botones solo tienen el nombre que el usuario haya decidido darles.

En la Figura 13 se muestra el código relacionado con esto, que explico a continuación.

Para esto he optado por realizar una simple consulta a la base de datos (Más información sobre como manejo las conexiones a la base de datos en el apartado 5.4 – Base de Datos) en la que se le pide a la base de datos los nombres distintos (Sin repetir) que hay, y por cada nombre, la aplicación genera un botón.

Para tener en cuenta la posición de cada botón, y no generar un botón encima de otro ya existente, utilizo una variable global “counter” que asocio como id a cada botón y que incremento en 1 cada vez que añado uno.

Utilizando una variable LayoutParams y aplicándola al botón, defino los parámetros de posición de ese botón de forma que el botón con la id “counter” se genere debajo del botón con la id “counter-1” (El último que ha sido generado). Si es el primero, no encuentra el anterior y se genera al inicio de la pantalla.

Al ser creados estos botones de forma dinámica, no están pre-definidos en el código y por lo tanto es necesario modificar su funcionalidad tratando sus onClick listeners (métodos que son llamados cuando el usuario apreta en el botón) cada vez que son creados, por esto, antes de añadir el botón a la vista hay que modificarlo. Simplemente añadiendo como String extra al Intent que lanzará la nueva actividad el nombre del botón pulsado (Que es el nombre del ejercicio) y añadiendo como valor booleano extra que el ejercicio que se lanza para hacer o modificar no es un ejercicio nuevo, sino que ya está creado.

```
//SELECT (unique) names from workout. Every name is a button
workoutNames = sqlite.getAllNames();
int i=0;
if (!workoutNames.isEmpty()){ //There are buttons
    do{
        counter+=1; //global counter for buttons IDs and positioning

        final Button btn = new Button(MainActivity.this);
        btn.setText(workoutNames.get(i));
        RelativeLayout rl = (RelativeLayout)
findViewById(R.id.relativeLayout);
        btn.setId(counter);
        RelativeLayout.LayoutParams params =
                new RelativeLayout.LayoutParams(
                        CoordinatorLayout.LayoutParams.MATCH_PARENT,
                        CoordinatorLayout.LayoutParams.WRAP_CONTENT);
        params.addRule(RelativeLayout.BELOW, counter-1);
        btn.setLayoutParams(params1);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { //Chooses the clicked button
                Intent intentFromButton = new Intent(MainActivity.this,
HiitContainer.class);
                workoutName = btn.getText().toString();

                intentFromButton.putExtra("workoutName", workoutName);
                intentFromButton.putExtra("isNew", false); //This means we're
using a previously created button
                startActivity(intentFromButton);
            }
        });
        rl.addView(btn);

        i+=1;
    } while ( i<workoutNames.size() );
}
```

Figura 13 Codigo para creacion de botones

Con esto cargamos a la pantalla principal los ejercicios que el usuario ya haya creado en su base de datos. Para crear un nuevo ejercicio es necesario modificar el comportamiento del *fab*. Así pues, modificando el onClick listener del *fab*, tenemos que lanzar primero una ventana emergente (En Android AlertDialog), que le pida al usuario el nombre del nuevo ejercicio, y luego lanzar la actividad de edición de Ejercicio, a lo que entraré en profundidad en el apartado 5.3 Actividad de Ejercicio.

Mi intención inicial era que simplemente apretando el *fab* se llevara al usuario a la actividad de edición de ejercicio, y ahí el usuario podría introducir el nombre de la actividad que deseara, pero al final he optado por pedirle el nombre antes de iniciar la actividad, de esta forma el proceso interno de creación es mucho más directo y unidireccional. Se guarda el botón con el nombre y se manda mediante una String Extra en el intent. De esta forma si el usuario mata la aplicación antes de guardar explícitamente, el nombre ya está seleccionado y se puede guardar en la base de datos. Así también se deja la interfaz de edición de ejercicio más limpia, al ahorrarnos un EditText en el que el usuario tuviera que introducir el nombre de la actividad.

La edición del onClick listener del *fab* simplemente consiste en preparar el intent que lanzará la actividad de edición del ejercicio, añadiéndole la información extra necesaria.

Así pues, antes de nada generamos una ventana emergente AlertDialog con una línea de texto editable (EditText) que permita al usuario introducir un nombre de ejercicio.

Si el usuario apreta en Guardar en esta ventana, el comportamiento que ha de seguir la aplicación es prácticamente el mismo que los pasos que he indicado anteriormente para añadir los botones directamente desde la base de datos. Añadir como información extra al intent el nombre del ejercicio, añadir un valor booleano que nos indica que es un ejercicio nuevo, definir los parámetros que se utilizarán para cargar el botón en la posición adecuada en la interfaz, y modificar el onClick listener del botón para que al selecionarlo más tarde, el usuario acceda a este ejercicio en particular.

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fabMain);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(final View view) {
        counter +=1; //Just to add new button BELOW existing one
        final Intent intent = new Intent(MainActivity.this, HiitContainer.class);
        final AlertDialog.Builder nameOfWorkoutBuilder = new
AlertDialog.Builder(MainActivity.this);
        final EditText saveName = new EditText(MainActivity.this);
        nameOfWorkoutBuilder.setTitle("Name of the Workout");
        nameOfWorkoutBuilder.setView(saveName);
        nameOfWorkoutBuilder.setPositiveButton("Save", new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                workoutName = saveName.getText().toString();
                nameExists = sqlite.doesItExist(workoutName);
                if ( nameExists ) { //The name already exists in the database
                    nameOfWorkoutBuilder.setTitle("Please use a different Name");
                } else {
                    intent.putExtra("workoutName",workoutName);
                    intent.putExtra("isNew",true); //it's a new workout

                    RelativeLayout rl = (RelativeLayout)
findViewById(R.id.relativeLayout);
                    final Button bt = new Button(MainActivity.this);

                    bt.setId(counter); bt.setText(workoutName);
                    RelativeLayout.LayoutParams params1 =
                            new RelativeLayout.LayoutParams(
                                    CoordinatorLayout.LayoutParams.MATCH_PARENT);
                    params1.addRule(RelativeLayout.BELOW, counter-1);
                    bt.setLayoutParams(params1);

                    bt.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View view) { //chooses the correct Button
                            Intent intentFromButton = new Intent(MainActivity.this,
HiitContainer.class);
                            workoutName = bt.getText().toString();
                            intentFromButton.putExtra("workoutName", workoutName);
                            intentFromButton.putExtra("isNew", false);
                            startActivity(intentFromButton);
                        }
                    });
                    rl.addView(bt); //Add button before launching activity
                    startActivity(intent);
                }
            }
        });
        nameOfWorkoutBuilder.setNegativeButton("Cancel", new
DialogInterface.OnClickListener(){
            public void onClick(DialogInterface dialog, int id){
                dialog.dismiss();
            }
        });
        nameOfWorkoutBuilder.show(); } });
```

Figura 14 Tratamiento boton fab

En cuanto al Navigation Drawer, al definir esta actividad principal como una NavigationDrawer Activity, se generan de forma automática los menús correspondientes del navigation drawer, por lo que solo hay que modificar las entradas por las que deseemos en el fichero res/menu/activity_main_drawer.xml.

Se modifica este XML de forma que se reflejen solo las entradas requeridas, y se cargan mediante android:icon los iconos correspondientes, en este caso ajustes, historia, ayuda, y acerca de.

```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/nav_settings"
        android:title="Settings"
        android:icon="@android:drawable/ic_menu_manage"
        />

    <item
        android:id="@+id/nav_history"
        android:title="History"
        android:icon="@android:drawable/ic_menu_my_calendar"
        />

    <item
        android:id="@+id/nav_help"
        android:title="Help"
        android:icon="@android:drawable/ic_menu_directions"
        />

    <item
        android:id="@+id/nav_about"
        android:title="About"
        android:icon="@android:drawable/ic_menu_help"
        />
</group>
```

Figura 15 Entradas para el NavigationDrawer

Tratar los clicks en estos items también se realiza de forma muy sencilla, simplemente hay que sobrescribir el método onNavigationItemSelected.

Cuando el usuario seleciona una entrada, se genera un atributo itemId del elemento seleccionado. Con ifs se verifica qué itemId ha seleccionado el usuario y se lanza la actividad mediante intent, o el AlertDialog correspondiente.

En la Figura 16 se puede observar como se trata cuando el usuario seleciona el menú de ajustes.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_settings){
        Intent intent = new Intent(MainActivity.this, SettingsActivity.class);
        startActivity(intent);
        return true;
    }
[…]
}
```

Figura 16 Tratamiento de items en NavigationDrawer

5.2. Objeto HiitItem

Para representar cada intervalo dentro de cada ejercicio, he decidido crear una clase y un layout desde cero, puesto que así tengo más control sobre los métodos que me permitirán obtener y establecer variables.

Para simplificar la programación, he optado por definir dos tipos de layout, uno que solo mostrará datos, y otro sobre el que se podrá editar. Lo he definido así puesto que la funcionalidad que busco es que el usuario vea los intervalos en una lista, y si quiere modificar alguna, al selecionarla aparecerá una ventana emergente sobre la que ya podrá introducir sus datos. No he querido que el usuario pueda introducir los datos directamente sobre la lista de intervalos porque al requerir el teclado Android, cada vez que el usuario quiera cambiar algo, Android sacaría de forma automática el teclado en pantalla, lo que puede mover y desplazar la interfaz, y en el caso de que el usuario quiera modificar muchos valores, esto puede ser bastante frustrante. Utilizando ventanas emergentes, la interfaz da sensación de quedar mucho más estática para el usuario, a pesar de requerir generar una ventana nueva, puesto que el "fondo" (la lista con los intervalos) se mantiene totalmente estática.

La única diferencia entre HiitItem y HiitItem_Edit es que en HiitItem_Edit, las repeticiones y tiempos son EditText que el usuario puede modificar, en lugar de TextViews. En la Figura 17 se explica qué elemento representa cada ítem dentro de un intervalo. Esta figura también es la representación visual que tendrá cada intervalo en la actividad de ejercicio.

Figura 17 Elemento HiitItem

Los elementos HiitItem están alineados unos con otros de forma que siempre mantengan esa estructura, tanto si el móvil está en posición vertical u horizontal. Están también montados sobre un widget CardView, de nuevo para seguir las instrucciones de google. Los elementos CardView introducen de forma nativa los bordes redondeados, y la sombra por elevación [18].

Para la interfaz HiitItem_Edit, inicialmente pensé que los elementos que indican el número de repeticiones deberían ser spinners (De ahí su nombre en estas clases), pero al implementarlos me di cuenta de que en caso de que el usuario quiera realizar una iteración un alto número de veces, ya que el spinner es un menú desplegable, debería hacer scroll sobre el spinner cada vez, y puesto que lo habitual en Hiits es realizar unas 10 – 20 repeticiones en total, esta solución no me pareció viable, y opté por elegir EditTexts en los que el usuario pueda introducir el número de repeticiones

directamente como texto. Definiendo en el layout estos EditTexts de forma que solo acpten números.

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/hiEditSpinnerGeneral"
    android:selectAllOnFocus="true"
    android:inputType="number"
    />
```

Figura 18 inputType de EditText

Hasta aquí es la definición visual del objeto HiitItem. También he definido una clase en java que representará a este objeto en la que hay definidos varios métodos muy útiles.

Cada objeto HiitItem tendrá los elementos que reflejan las repeticiones generales (spinnerGeneral), repeticiones sprint y rest (spinnerSprint y spinnerRest respectivamente) y los tiempos sprint y rest (timeSprint y timeRest). Así pues, esta clase tendrá dos constructores, uno con todos estos atributos y uno vacío, como se puede apreciar en la Figura 19.

```
public class HiitItemClass {
    String spinnerGeneral;
    String spinnerSprint;
    String timeSprint;
    String spinnerRest;
    String timeRest;

    public HiitItemClass(String spinnerGeneral, String spinnerSprint,
                         String timeSprint, String spinnerRest,
                         String timeRest) {
        this.spinnerGeneral = spinnerGeneral;
        this.spinnerSprint = spinnerSprint;
        this.timeSprint = timeSprint;
        this.spinnerRest = spinnerRest;
        this.timeRest = timeRest;
    }

    public HiitItemClass() {
    }
}
```

Figura 19 Constructores clase HiitItem

Además de todos los métodos getters and setters para obtener y establecer estos valores

```
[. . . ]

public String getSpinnerGeneral() {
    return spinnerGeneral;
}

public void setSpinnerGeneral(String spinnerGeneral) {
    this.spinnerGeneral = spinnerGeneral;
}

[. . . ]
```

Figura 20 Setters y Getters de HiitItem

5.3. Actividad de Ajustes

Para la actividad de ajustes he seguido el esqueleto que Android Studio ofrece cuando se crea una actividad de tipo Settings nueva.

Lo primero que se crea es una Actividad SettingsActivity. Esta actividad carga varios archivos XML, que es lo que más interesa modificar. La clase SettingsActivity define todos los métodos necesarios para guardar los ajustes seleccionados de forma automática, por lo que no hay que modificar demasiado aquí. Los ajustes seleccionados se guardan utilizando el nombre de la entrada (key) en el fichero xml que define la interfaz.

Los ficheros XML que utilizaré para mi aplicación son pref_headers.xml, pref_general.xml, y pref_notification.xml.

Pref_headers sirve de menú padre de pref_general y pref_notification. Con él definimos las dos entradas para los ajustes generales y los ajustes de notificaciones, así como sus iconos y fragments que implementarán.

```
<preference-headers
xmlns:android="http://schemas.android.com/apk/res/android">

    <header
        android:fragment="[…].SettingsActivity$GeneralPreferenceFragment"
        android:icon="@drawable/ic_info_black_24dp"
        android:title="@string/pref_header_general" />

    <header
        android:fragment="[…].SettingsActivity$NotificationPreferenceFragment"
        android:icon="@drawable/ic_notifications_black_24dp"
        android:title="@string/pref_header_notifications" />

</preference-headers>
```

Figura 21 pref_headers

En el menú Ajustes General el usuario podrá seleccionar de un menú desplegable la antelación con la que quiere que la aplicación le avise antes de que se cambie el intervalo, y el tipo de notificaciones de voz que quiere recibir, en caso de que quiera recibir alguna.

El menú desplegable se implementa de forma sencilla, utilizando un ListPreference, y cogiendo los valores (1 – 5) que se mostrarán.

Para el tipo de notificaciones a recibir, primero definimos un botón de tipo Switch (SwitchPreference) que permitirá al usuario si quiere o no recibir notificaciones de voz. Además de esto, he definido un grupo de conjunto de preferencias (PreferenceCategory) que englobará varios checkboxes. Lo he hecho así ya que definiendo una dependencia en este grupo con el Switch, se pueden desactivar todos los checkboxes de golpe si el usuario no desea recibir notificaciones.

En la Figura 22 se puede ver la estructura de la interfaz en XML, así como la dependencia de la que hablo en el párrafo anterior, y la Figura 233 es una captura de pantalla de la actividad de Ajustes, comprobando la funcionalidad del botón que desactiva los checkboxes.

```xml
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <ListPreference
        android:entries="@array/pref_ticks"
        android:entryValues="@array/pref_ticks_values"
        android:key="example_tick_list"
        android:title="@string/pref_title_how_many_ticks" />
    <SwitchPreference
        android:defaultValue="false"
        android:key="general_voice"
        android:title="@string/pref_gen_voice" />
    <PreferenceCategory
        android:dependency="general_voice"
        android:title="What extra information do you want to hear?">
        <CheckBoxPreference
            android:key="example_checkbox_list_iteration_number"
            android:title="Current Iteration"
        <CheckBoxPreference
            android:key="example_checkbox_list_iterations_remaining"
            android:title="Remaining Iterations" />
        <CheckBoxPreference
            android:key="example_checkbox_list_time_spent"
            android:title="Time Spent" />
        <CheckBoxPreference
            android:key="example_checkbox_list_time_remaining"
            android:title="Time Remaining" />
        <CheckBoxPreference
            android:key="example_checkbox_list_sprint_rest"
            android:title="Sprint or Rest" />
    </PreferenceCategory>
</PreferenceScreen>
```

Figura 22 XML ajustes generales

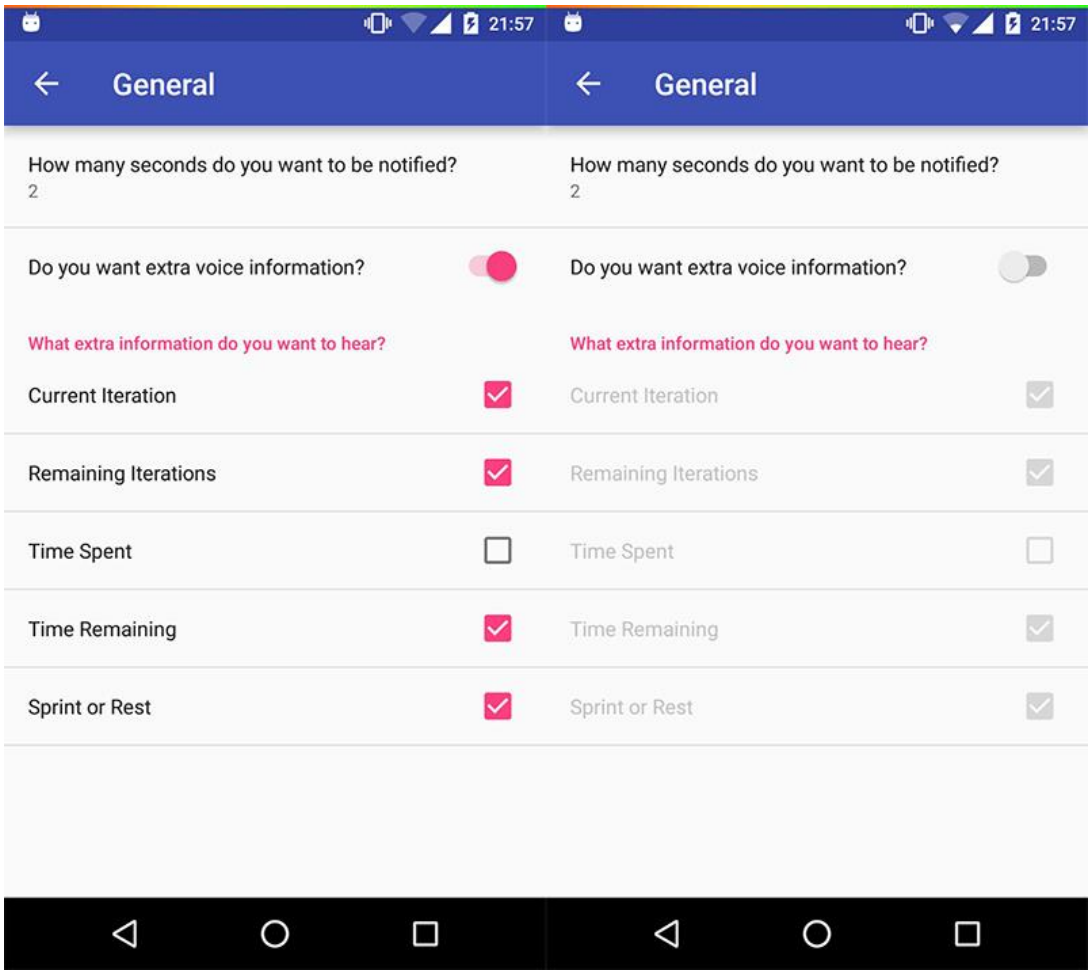

Figura 23 Interfaz Ajustes Generales

En las preferencias de notificación, el usuario podrá elegir si quiere o no recibir notificaciones sonoras, elegir qué sonido quiere para notificarle el cambio a sprint y a descanso, y si quiere o no recibir notificaciones en forma de vibración. En esta pantalla también utilizo la herramienta de dependencias, de forma que si el usuario no quiere recibir notificaciones de audio, los menús desplegables del audio no serán seleccionables.

Los menús desplegables muestran todos los sonidos de tipo notificación a los que el móvil tiene acceso. Incluso si el usuario ha instalado un programa que le permitiera seleccionar sonidos más allá de los que están predefinidos en Android, este menú dará la opción de seleccionar ese programa para seleccionar los sonidos extra.

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
«...»
<SwitchPreference
    android:defaultValue="true"
    android:key="notifications_new_message"
    android:title="@string/pref_title_new_message_notifications" />

<RingtonePreference
    android:defaultValue="content://settings/system/notification_sound"
    android:dependency="notifications_new_message"
    android:key="notifications_new_message_ringtone_sprint"
    android:ringtoneType="notification"
    android:title="@string/pref_title_ringtone_sprint" />

«...»
</PreferenceScreen>
```

Figura 24 Obtener los sonidos de notificación

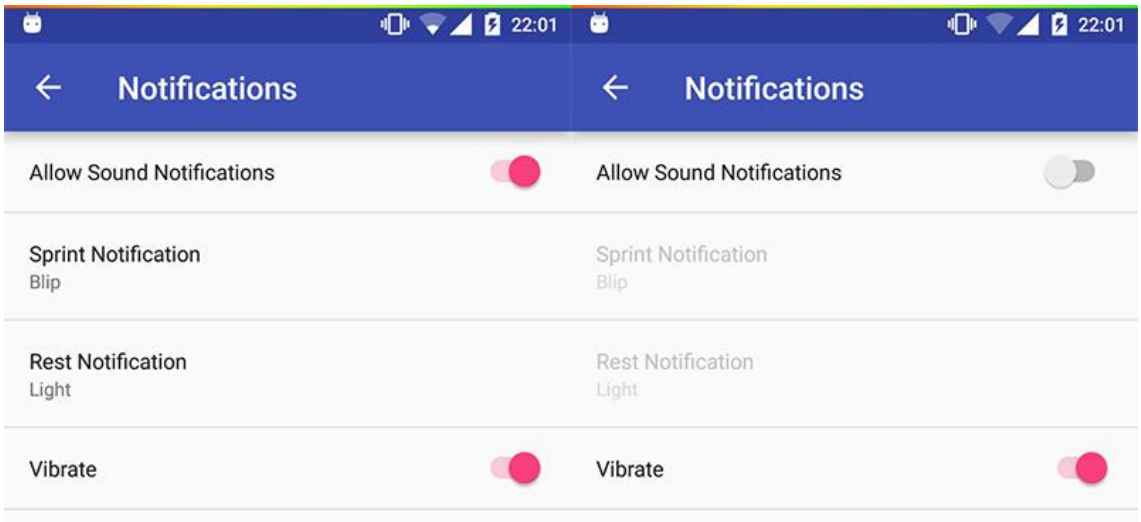

Figura 25 Interfaz de Ajustes de Notificación

5.3. Actividad de Ejercicio

Para esta actividad es necesario entrar en detalle tanto en la lógica, como en la interfaz elegida.

Para la capa de presentación, como se puede ver en la Figura 26, opté por usar de base un recyclerView (RecyclerHiitContainer) que es una versión más avanzada y flexible de un ListView [20]. El RecyclerView, además de ser un contenedor que permite mostrar grandes conjuntos de datos, es muy eficiente al desplazar todos estos datos al hacer scroll, y además permite varias opciones de coordinación de animaciones con otros

elementos. En el caso que nos ocupa, estos otros elementos son la toolbar y un Floating Action Button (fab).

Estos tres elementos se encuentran todos envueltos en un coordinatorLayout, que es el que se encarga de coordinar todas estas animaciones.

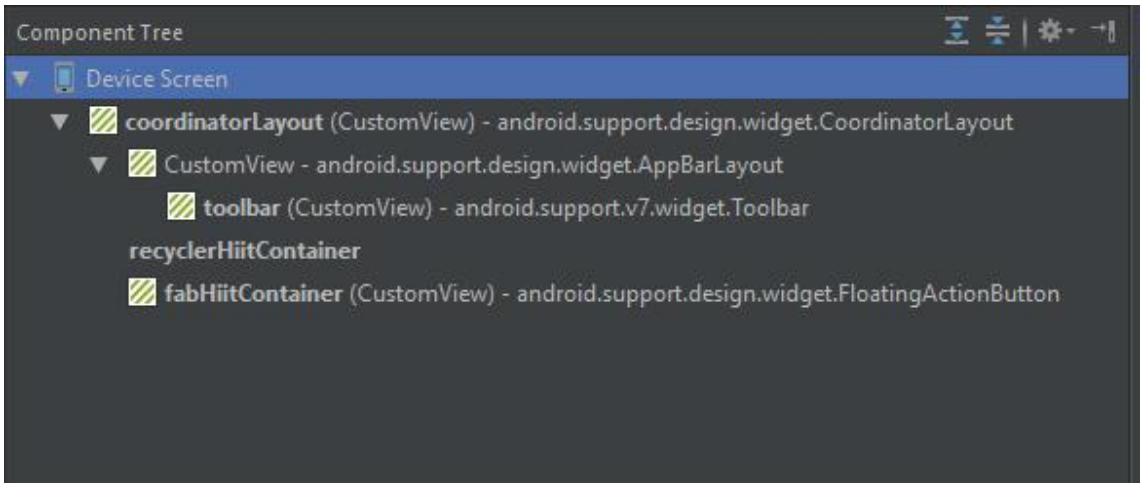

Figura 26 Estrucutra del Layout

Para configurar estas animaciones, cada elemento tiene un atributo *app:layout_scrollFlags* que permite configurar las animaciones por scroll. En este caso, he optado por "scroll | enterAlways", con esto lo que consigo es que al hacer scroll hacia abajo, la toolbar desaparezca. Para hacerla aparecer, enterAlways define que al hacer Scroll en dirección contraria aparecerá. Podría configurarse de forma que solo apareciera cuando se haga scroll hacia arriba y se llegue al primer elemento, o que apareciera al dejar de hacer scroll hacia abajo. He optado por esta porque me parece más natural, en el caso de que haya una gran cantidad de ítems, no tener que volver hasta arriba del todo para guardar/descartar/correr.

En cuanto a la capa lógica lo primero que debemos hacer es obtener el nombre del ejercicio que el usuario ha introducido en la pantalla principal de la aplicación. Tanto si ha sido creando un ejercicio nuevo como si ha sido seleccionando uno ya existente, en los dos casos el intent que ha lanzado la actividad actual tendrá una String extra con el nombre del ejercicio, y un valor booleano que nos dirá si venimos de un ejercicio nuevo o de uno ya existente.

La clase principal de la actividad de ejercicio es HiitContainer, que contendrá todos los elementos Hiit, es decir, los intervalos de cada ejercicio.

Para tratar con los elementos mencionados en la capa de presentación, es necesario inicializar las siguientes variables, CoordinatorLayout, RecyclerView, MyAdapter y LayoutManager.

```
CoordinatorLayout coordinator;
RecyclerView mRecycler;
MyAdapter mAdapter;
RecyclerView.LayoutManager mLayoutManager;
```

Figura 27 Inicialización de las Variables

MyAdapter es una clase que he creado y que hereda de RecyclerView.Adapter, pero que he modificado para añadir funcionalidades y aplicarla a mis estructuras de datos. Tiene

un ArrayList de elementos HiitItem, ya que esta clase será la encargada de "cargarlos", un listener OnItemClickListener, y se define también la clase ViewHolder [22], que define una vista e información sobre su lugar en el RecyclerView.

```
private ArrayList<HiitItemClass> data;
private OnItemClickListener listener;

public MyAdapter(ArrayList<HiitItemClass> data, MyAdapter.OnItemClickListener
listener){
    this.data = data;
    this.listener = listener;
}
public static class ViewHolder extends RecyclerView.ViewHolder {
    public View v;

    public TextView spinnerGeneral;
    public TextView spinnerSprint;
    public TextView timeSprint;
    public TextView spinnerRest;
    public TextView timeRest;


    public ViewHolder(View view) {
        super(view);
        v = view;

        spinnerGeneral = (TextView) view.findViewById(R.id.hiSpinnerGeneral);
        spinnerSprint = (TextView) view.findViewById(R.id.hiSprintReps);
        timeSprint = (TextView) view.findViewById(R.id.hiTimeSprint);
        spinnerRest = (TextView) view.findViewById(R.id.hiRestReps);
        timeRest = (TextView) view.findViewById(R.id.hiTimeRest);
    }
}
```

Figura 28 Constructor MyAdapter

Como ya he dicho, MyAdapter se encarga de obtener los objetos HiitItem y cargarlos en el RecyclerView (ViewHolder). Esto se consigue con varios métodos. Para crear el ViewHolder necesario para que el RecyclerView represente los elementos es necesario modificar el método onCreateViewHolder, que básicamente carga el layout hiititem en una Vista, y luego asocia esa vista a un Holder.

```
@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    // Create the View from an XML layout resource
    View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.hiititem, parent, false); //The non-editable one

    // Create a ViewHolder that will hold references to the View
    MyAdapter.ViewHolder holder = new ViewHolder(view);
    return holder;
}
```

Figura 29 Creación del ViewHolder

Para actualizar la información que muestra el ViewHolder, es necesario modificar el método onBindViewHolder, que recibe el holder creado, y la posición, que hace referencia al elemento HiitItem del ArrayList definido anteriormente.

Este método simplemente selecciona un elemento HiitItem del ArrayList data y, como está definido en "5.2 Objeto HiitItem", utiliza los getters de la clase para obtener la información requerida. Luego asocia esa información al Holder creado (Que como hemos visto en la Figura 29, tiene la estructura layout de hiititem).

```
@Override
public void onBindViewHolder(ViewHolder holder, final int position) {

    holder.spinnerGeneral.setText(data.get(position).getSpinnerGeneral());

    holder.spinnerSprint.setText(data.get(position).getSpinnerSprint());
    holder.timeSprint.setText(data.get(position).getTimeSprint());

    holder.spinnerRest.setText(data.get(position).getSpinnerRest());
    holder.timeRest.setText(data.get(position).getTimeRest());

    // Asscociate the click listener to the View
    holder.v.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            listener.onItemClicked(position);
        }
    });
}
```

Figura 30 Actualizar el ViewHolder

Volviendo a la clase principal de la Actividad de Ejercicio HiitContainer, se encuentran mediante findViewById los elementos Coordinator y RecyclerView definidos en el layout y se inicializan.

En cuanto a la conexión con la base de datos e insertar/extraer datos, hay dos formas de proceder. Mediante métodos creados dentro de la clase específica que he creado para modificar funcionalidades de la base de datos (MySqliteOpenHelper, que extiende SQLiteOpenHelper, a lo que entraré en profundidad en el apartado 5.4 Base de Datos), o bien abriendo una conexión de escritura directamente a la base de datos (SQLiteDatabase), y utilizando variables ContentValues. Un ContentValue es un conjunto de pares nombre-valor, que luego pueden ser insertados directamente en la base de datos.

Como he comentado en el apartado anterior, el intent que lanza esta actividad, aparte de tener el nombre del ejercicio, tiene un valor booleano que indica si la actividad es nueva o no. En caso de ser un ejercicio nuevo, el programa genera automáticamente una iteración del ejercicio mediante el método generateData() con valores básicos, y lo añade a la interfaz, y a la base de datos, junto a la fecha. Una vez insertados los datos en la base de datos, se guarda la ID de la fila que acaba de insertar, ya que hará falta para eliminar esa fila si más tarde el usuario desa desacartar los cambios.

Si el usuario ha elegido un ejercicio ya creado, simplemente se cargan desde la base de datos los intervalos que tengan el nombre seleccionado. getItems(workoutName) es un método que he creado dentro de la clase MySqliteOpenHelper sobre el que hablaré más adelante en la sección 5.4 Base de Datos.

```
sqlite = new MySqliteOpenHelper(getApplicationContext());
final SQLiteDatabase db = sqlite.getWritableDatabase();
final ContentValues values = new ContentValues();

workoutName = getIntent().getStringExtra("workoutName");
isNew = getIntent().getBooleanExtra("isNew",true);

if(isNew){ //If it's new always generate ONE Hiit Interval and save
    data = generateData();
    Calendar c = Calendar.getInstance();
    datewo =
c.get(Calendar.YEAR)+"_"+c.get(Calendar.MONTH)+"_"+c.get(Calendar.DAY_OF_MONTH
);
    //Add all the attributes to the database
    values.put( "name", workoutName);
    values.put( "date",datewo);
    values.put( "spinnerGeneral", data.get(0).getSpinnerGeneral() );
    values.put( "spinnerSprint", data.get(0).getSpinnerSprint() );
    values.put( "timeSprint", data.get(0).getTimeSprint() );
    values.put( "spinnerRest", data.get(0).getSpinnerRest() );
    values.put( "timeRest", data.get(0).getTimeRest() );

    rowId.add(db.insert("workout",null,values));
    values.clear();

} else{ //we're editing an existing one, no need to generate new data
    data = sqlite.getItems(workoutName);
}
```

Figura 31 Carga de intervalos por primera vez

Ahora es necesario modificar el comportamiento del adapter que he definido anteriormente. Para esto hay que modificar el listener onItemClicked del adapter.

El comportamiento que busco es que al apretar un ítem (es decir, un HiitItem/intervalo) aparezca una ventana emergente que permita al usuario modificarlo (cargando el layout hiititem_edit) y guardar los cambios.

En la Figura 32 se crea el objeto MyAdapter mAdapter, y se sobreescribe el método para el listener onItemClicked de forma que cuando el usuario selecione un intervalo, se genere una ventana emergente de tipo AlertDialog con el layout hiititem_edit cargado.

Para que la ventana emergente tenga los valores del ítem que el usuario ha elegido, es necesario obtener estos dato desde el ArrayList de HiitItems, buscar la referencia contenedores de la ventana emergente (Que tienen la estructura layout hiititem_edit), y cargar los datos ahí.

En la Figura 32 podemos ver como ejemplo como cargar el dato obtenido de el ArrayList de HiitItems (data.get(position)) en el número de repeticiones generales (SpinnerGeneral) del layout hiitItem_edit, que luego cargaremos en la ventana emergente AlertDialog.

```
MyAdapter mAdapter = new MyAdapter(data, new MyAdapter.OnItemClickListener() {
    @Override
    public void onItemClicked(final int position) {
        //Fill the information with the existing one
        AlertDialog.Builder builder = new AlertDialog.Builder(HiitContainer.this);
        LayoutInflater inflaterDialog = HiitContainer.this.getLayoutInflater();
        final View dialogView_edit = inflaterDialog.inflate(R.layout.hiititem_edit,null);

        //Get all the values from the clicked item into the hiititem_edit popup
        EditText temp;

        temp = (EditText) dialogView_edit.findViewById(R.id.hiEditSpinnerGeneral);
        final String spinnerGeneral = data.get(position).getSpinnerGeneral();
        temp.setText(spinnerGeneral);

            [ … ]
   builder.setView(dialogView_edit);
   builder.show();
   }
});
```

Figura 32 Creacion de MyAdapter

Modificar el valor de un ítem implica modificar esa entrada de la base de datos. Para esto, es necesario guardar todos los valores del HiitItem seleccionado (workoutName, datewo, spinnerGeneral, spinnerSprint, timeSprint, spinnerRest y timeRest) antes de modificarlo, y eliminar de la base de datos la fila que los contenga, para más tarde añadir a la base de datos los nuevos valores.

```
String[] args = new String[]{
        workoutName,datewo,spinnerGeneral, spinnerSprint,
timeSprint,spinnerRest,timeRest
};
db.delete("workout",
        "name=? AND date=? AND spinnerGeneral=? AND spinnerSprint =? AND
timeSprint=? AND spinnerRest=? AND timeRest=?",
        args);


Calendar c = Calendar.getInstance();
datewo =
c.get(Calendar.YEAR)+"_"+c.get(Calendar.MONTH)+"_"+c.get(Calendar.DAY_OF_MONTH
);

values.put("name", workoutName);
values.put("date", datewo);
values.put( "spinnerGeneral", spinnerGen );
values.put( "spinnerSprint", spinnerS );
values.put( "timeSprint", timeS );
values.put( "spinnerRest", spinnerR );
values.put( "timeRest", timeR );
rowId.add(db.insert("workout",null, values));
values.clear();
dialog.dismiss();
```

Figura 33 Actualizacion de valores en la base de datos

La ventana de edición de ejercicio también tiene un floating action button (fab), por lo que es necesario alterar su comportamiento.

De nuevo, será necesario modificar simplemente el listener onClick. Al apretar en él, deberá aparecer una ventana emergente de tipo AlertDialog que carge el layout hiitItem_edit en la que el usuario pueda introducir los datos que quiera.

Como se puede observar en la Figura 34, de nuevo se crea un AlertDialog al que cargamos la vista hiititem_edit. Al apretar el botón de guardar, se obtienen los valores

que el usuario ha introducido (Así como la fecha), y se insertan en la base de datos. Para que estos cambios se reflejen en la interfaz, los nuevos valores deben ser añadidos en “data”, el ArrayList que contiene todos los elementos HiitItem, utilizando el constructor de la clase HiitItemClass, que recibe como cadena de texto los valores asociados a la ventana emergente.

```
final FloatingActionButton fabHC = (FloatingActionButton)
findViewById(R.id.fabHiitContainer);
fabHC.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(HiitContainer.this);
        final LayoutInflater inflaterDialog = HiitContainer.this.getLayoutInflater();
        final View dialogViewFab = inflaterDialog.inflate(R.layout.hiititem_edit,null);
        builder.setView(dialogViewFab);
        builder.setPositiveButton("Save", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {

                EditText sg = (EditText)
dialogViewFab.findViewById(R.id.hiEditSpinnerGeneral);
                [ … ]

                Calendar c = Calendar.getInstance();
                datewo =
c.get(Calendar.YEAR)+"_"+c.get(Calendar.MONTH)+"_"+c.get(Calendar.DAY_OF_MONTH);
                values.clear();
                values.put("name", workoutName);
                values.put("date", datewo);
                values.put( "spinnerGeneral", sg.getText().toString() );
                [ … ]
                rowId.add(db.insert("workout",null, values));

                data.add(new HiitItemClass(
                        sg.getText().toString(),ss.getText().toString(),
                        ts.getText().toString(),sr.getText().toString(),
                        tr.getText().toString()));
                dialog.dismiss();
            }
        });
        builder.show();
    }
});
```

Figura 34 Modificacion del fab listener

En cuanto a la barra de herramientas, los tres iconos (Guardar, descartar e iniciar ejercicio) se tratan de la misma forma que tratamos los elementos del NavBar en la ventana principal, simplemente modificando el listener onOptionsItemSelected y verificando qué botón estamos apretando cada vez.

5.3.1. Toolbar button - Iniciar ejercicio

Para tratar la interacción con el usuario cuando se comienza el ejercicio, lo primero es obtener los ajustes. Esto se obtiene utilizando un elemento SharedPreferences que almacena la configuración que el usuario ha introducido en las actividades de ajustes.

Se define si el usuario quiere notificaciones sonoras, si quiere vibración y el número de ticks, utilizando los nombres de las entradas de estos ajustes que han sido definidas anteriormente en la interfaz xml de los ajustes generales y de notificaciones.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);

final boolean notification =
prefs.getBoolean("notifications_new_message",true);
final boolean vibrate =
prefs.getBoolean("notifications_new_message_vibrate",true);
final int ticks = prefs.getInt("example_tick_list",0);
final Vibrator v = (Vibrator)
getApplicationContext().getSystemService(Context.VIBRATOR_SERVICE);
```

Figura 35 Tratamiento de Preferncias del Usuario

Para poder acceder al histórico de ejercicios, aquí me guardo en la base de datos (en una tabla "history" diferente a la que hemos utilizado hasta ahora, más información sobre esta tabla en el apartado 5.4 Base de Datos) la fecha y hora, y el nombre del ejercicio.

addWorkoutLive es un método que está definido en la clase MySqliteOpenHelper que he definido (explicado en profundidad en el apartado 5.4 Base de Datos) y que añade a la base de datos el nombre y la fecha del entrenamiento.

```
Calendar c = Calendar.getInstance();
datewo = c.get(Calendar.YEAR)+"_"+c.get(Calendar.MONTH)+"_"+
        c.get(Calendar.DAY_OF_MONTH)+"_"+c.get(Calendar.HOUR_OF_DAY);

sqlite = new MySqliteOpenHelper(getApplicationContext());
sqlite.addWorkoutLive(workoutName, datewo);
```

Figura 36 Guardar informacion en la BD

Para tratar los avisos al usuario simplemente hay que acceder en orden a los valores que hay en cada intervalo guardado. Una vez tengamos los tiempos, en cada iteración se repite el intervalo entero el número de veces que indique SpinnerGeneral, y se repite cada tiempo de sprint/descanso el número de veces que indique spinnerSprint y SpinnerRest respectivamente.

En la Figura 37 podemos comprobar como el spinnerGeneral (sg) controla el número de repeticiones de cada iteración (Cada iteración consta de un spinnerSprint (ss) y un SpinnerRest (sr)), y de sus tiempos. Se realiza primero la sección sprint, que se repite con un método "for" el número de veces que sea necesario. Lo mismo con la sección de descanso.

Para controlar el temporizador, se utiliza CountDownTimer, que es una clase que Android tiene incluida. Requiere el total en milisegundos, y los milisegundos que forman un "tick". Se le pasa el tiempo de sprint/descanso a milisegundos, y se define cada "tick" como un segundo (1000 ms). En cada segundo que pasa, si el número de segundos que faltan hasta que se acabe la sección es igual o menor al número de ticks que el usuario ha definido, se reproduce un sonido. Esto sirve como cuenta atrás para que el usuario sepa cuando una iteración en particular está a punto de acabar.

Para utilizar los sonidos que el usuario ha elegido, primero se debe obtener el valor en las SharedPreferences asociado al sonido de sprint (cuyo título identificativo es "notifications_new_message_ringtone_sprint") y al sonido de descanso ("notifications_new_message_ringtone_rest"). Con esto se carga el sonido específico en una variable Uri (Uniform Resource Identifier, identificador uniforme de recursos).

Se utiliza RingtoneManager.getRingtone, que devuelve el tono asociado a este uri. Si las notificaciones están desactivadas, se verifica que la vibración esté activada y simplemente se hace vibrar el teléfono durante un segundo.

```
String ringtoneSprint =
prefs.getString("notifications_new_message_ringtone_sprint","DEFAULT_SOUND");
final Uri soundnotifSprint = Uri.parse(ringtoneSprint);

String ringtoneRest =
prefs.getString("notifications_new_message_ringtone_rest","DEFAULT_SOUND");
final Uri soundnotifRest = Uri.parse(ringtoneRest);

for(int sg=1; sg<=spinnerGeneral; sg++){ //repeats general iteration
    for(int ss=1; ss<=spinnerSprint; ss++){ //repeats Sprint iteration
        long tsInMs = timeSprint*1000;

        new CountDownTimer(tsInMs, 1000) {
            public void onTick(long millisUntilFinished) {
                String muf = String.valueOf(millisUntilFinished / 1000);
                if(millisUntilFinished/1000 <= ticks){
                    Ringtone r =
RingtoneManager.getRingtone(getApplicationContext(), soundNotifSprint);
                    r.play();
                }
            }
            public void onFinish() {
                if(notification){
                    Ringtone r =
RingtoneManager.getRingtone(getApplicationContext(), soundNotifSprint);
                    r.play();
                    if(vibrate) v.vibrate(1000);
                } else {
                    if(vibrate) v.vibrate(1000);
                }
            }
        }.start();
    }
    for(int sr=1; sr<spinnerRest; sr++){ //repeats rest iteration
        long trInMs = timeRest*1000;

        new CountDownTimer(trInMs, 1000) {
            public void onTick(long millisUntilFinished) {
                String muf = String.valueOf(millisUntilFinished / 1000);
                if(millisUntilFinished/1000 <= ticks){
                    Ringtone r =
RingtoneManager.getRingtone(getApplicationContext(), soundnotifRest);
                    r.play();
                }
            }
            public void onFinish() {
                if (notification) {
                    Ringtone r =
RingtoneManager.getRingtone(getApplicationContext(), soundnotifRest);
                    r.play();
                    if(vibrate) v.vibrate(1000);
                } else {
                    if(vibrate) v.vibrate(1000);
                }
            }
        }.start();
    }
}
```

Figura 37 Realización del ejercicio

5.4. Base de Datos

Para trabajar con un sistema de bases de datos en Android es conveniente modificar la clase que Android ofrece por defecto, SQLiteOpenHelper.

Para esto, he creado una clase MySqliteOpenHelper, que hereda de SQLiteOpenHelper y he añadido varios métodos que facilitarán el acceso a la base de datos.

Antes que nada, para facilitar la programación, he definido varios Strings que usaré más adelante como por ejemplo TABLE_WORKOUT, que contiene el nombre de la tabla de la base de datos, o USEFUL_COLUMNS, que contiene los nombres de las columnas que se suelen utilizar.

En la actividad principal se utilizaba un método llamado getAllNames, que devolvía una ArrayList con todos los nombres únicos, no repetidos que aparecían en la base de datos. Cada uno de estos nombres deberá ser representado como un botón.

En el método getAllNames, primero se crea o lee la base de datos utilizando el método getReadableDatabase. Esto devuelve un objeto SQLiteDatabase sobre el que se pueden realizar consultas. Esta consulta se realiza utilizando el método query, de forma que devuelva de forma única (Definido con el primer argumento a true) sólo la primera columna “name”, sobre la tabla TABLE_WORKOUT.

Los datos que devuelva esta consulta se guardan utilizando una variable de tipo Cursor [23], que luego se recorrerá y añadirá al ArrayList names. En caso de no haber ninguna, se devolvería un ArrayList vacío.

```
public ArrayList<String> getAllNames(){ //Only get the names of the workout
(For the buttons)
    ArrayList<String> names = new ArrayList<>();
    SQLiteDatabase db = this.getReadableDatabase();
    String [] COLUMNS={"name"};
    Cursor cursor = db.query(true, TABLE_WORKOUT,
COLUMNS,null,null,null,null,null,null);

    if (cursor.moveToFirst()){
        do{
            names.add( cursor.getString(0) );
        }while (cursor.moveToNext());
    }
    return names;
}
```

Figura 38 método getAllNames

El método getItems(String workoutName) es otro de los métodos que es necesario implementar. Este método devolverá todas las filas de la base de datos que posean como nombre “workoutName”. Ya que los nombres no se pueden repetir, no es necesario comprobar que todas las filas pertenecen al mismo ejercicio más allá de la comprobación de su nombre.

Aquí la query utilizada es algo más compleja. Sigue el siguiente modelo:

Db.query(Nombre de la tabla de la base de datos, columnas a seleccionar, condición a cumplir, elemento a combinar con la condición, Group By, Having, Order By). De nuevo se guarda este resultado en una variable de tipo Cursor, que luego será recorrida.

Ahora se crea un elemento HiitItemClass (utilizando el constructor vacío definido en la clase HiitItem y los setters) que se añadirá a un ArrayList de HiitItemClass.

El método cursor.getString(i) devuelve la i-ésima columna de cada fila, por lo que simplemente debemos seguir el mismo orden que el usado en la petición. Si la lista está vacía, generamos un elemento HiitItemClass con valores genéricos.

```
public ArrayList<HiitItemClass> getItems(String workoutName){
    SQLiteDatabase db = this.getReadableDatabase();
    ArrayList<HiitItemClass> hiitList = new ArrayList<>();
    //name date spinnerGeneral spinnerSprint timeSprint spinnerRest timeRest

    Cursor cursor = db.query(TABLE_WORKOUT,USEFUL_COLUMNS,"name=?",new
String[]{workoutName},null,null,null);

    if (cursor.moveToFirst()){
        do{
            //Build HiitItemObject
            HiitItemClass hiitItem = new HiitItemClass();
            hiitItem.setSpinnerGeneral(cursor.getString(0));
            hiitItem.setSpinnerSprint(cursor.getString(1));
            hiitItem.setTimeSprint(cursor.getString(2));
            hiitItem.setSpinnerRest(cursor.getString(3));
            hiitItem.setTimeRest(cursor.getString(4));

            hiitList.add(hiitIteme);
        }while (cursor.moveToNext());
    } else { //if empty, create generic one
        hiitList.add( new HiitItemClass("1","1","30","1","50") );
    }
    return hiitList;
}
```

Figura 39 método getItems

Como hemos visto en el apartado 5.3.1 Toolbar button – Iniciar Ejercicio, cada vez que el usuario realiza un ejercicio se llama al método addWorkoutLive(). Este método recibe el nombre del ejercicio y la fecha y hora de la realización y lo guarda en una tabla diferente de la base de datos. Hasta ahora solo se había utilizado la tabla workout, pero en este caso se utilizará la tabla history.

```
public void addWorkoutLive(String name, String date){
    final ContentValues values = new ContentValues();
    values.put( "name", name );
    values.put( "date", date );
    SQLiteDatabase db = this.getReadableDatabase();
    db.insert(TABLE_HISTORY, null, values);
}
```

Figura 40 método addWorkoutLive para guardar ejercicio con fecha

Para la actividad historia, definimos dos métodos, getRunningHistory y getNamesAndDates.

getRunningHistory devuelve un ArrayList de pares (nombre, fecha), simplemente realizando una consulta que devuelve las columnas nombre y fecha. Al ser la tabla de historia, aquí no es necesario realizar ninguna comprobación, sino que se le piden todas las filas.

```
public ArrayList<Pair> getRunningHistory() {
    SQLiteDatabase db = this.getReadableDatabase();
    Pair workout = new Pair();
    ArrayList<Pair> workoutList = new ArrayList<>();

    String [] COLUMNS={"name","date"};
    Cursor c = db.query(TABLE_HISTORY, COLUMNS,
            null, null, null, null, null, null);
    if(c.moveToFirst()){
        do{
            String name = c.getString(0);
            String date = c.getString(1);
            workout.setLeftAndRight(name,date);
            workoutList.add(workout);
        }while(c.moveToNext());
    }
    return workoutList;
}
```

Figura 41 Método para devolver la historia de ejercicios realizados

En el caso de getNamesAndDates, nos devuelve también un ArrayList de pares nombre,fecha. Al obtener estos valores de la base de datos de ejercicios, y no de la de historia, es necesario en la consulta agrupar los resultados por nombre, ya que cada nombre puede tener una gran cantidad de filas. Esto se consigue con el quinto argumento del método db.query, que representa GROUP BY.

Es necesario utilizar MAX(date) en las columnas que nos devolverá porque GROUP BY exige que las columnas por las que no estés agrupando sean "unidas" de alguna forma. Aquí al ser todas las fechas las mismas para un mismo nombre, utilizar MAX(date) no afectará en nada, devolviéndonos el valor deseado.

```
public ArrayList<Pair> getNamesAndDates(){ //Only get the names of the workout
(For the buttons)
    Pair nameAndDate = new Pair();
    ArrayList<Pair> list= new ArrayList<>();

    SQLiteDatabase db = this.getReadableDatabase();
    String [] COLUMNS={"name", "MAX(date)"};
    Cursor cursor = db.query(TABLE_HISTORY, COLUMNS, null, null, "name", null,
null, null); //Gets all distinct names

    if (cursor.moveToFirst()){
        do{

nameAndDate.setLeftAndRight(cursor.getString(0),cursor.getString(1));
            list.add(nameAndDate);
        }while (cursor.moveToNext());
    }
    return list;
}
```

Figura 42 Método para obtener nombres y fechas

Los métodos que crean, eliminan y actualizan la base de datos simplemente se modifican para que se apliquen sobre la base de datos que sea conveniente, TABLE_WORKOUT o TABLE_HISTORY o ambas.

5.5. Actividad de Historia

La actividad de Historia permite al usuario ver dos listas. Una en la que verá qué ejercicio ha realizado qué día, y otra en la que verá cuando creó cada ejercicio. Ya que son dos informaciones semejantes y con la misma estructura, he decidido que esta actividad esté implementada en forma de pager, por la cual el usuario verá en la

cabecera los títulos historia de creación e historia de ejercicios, y deslizando la pantalla hacia la izquierda o la derecha podrá cambiar entre estas, lo que se puede apreciar en la Figura 43.

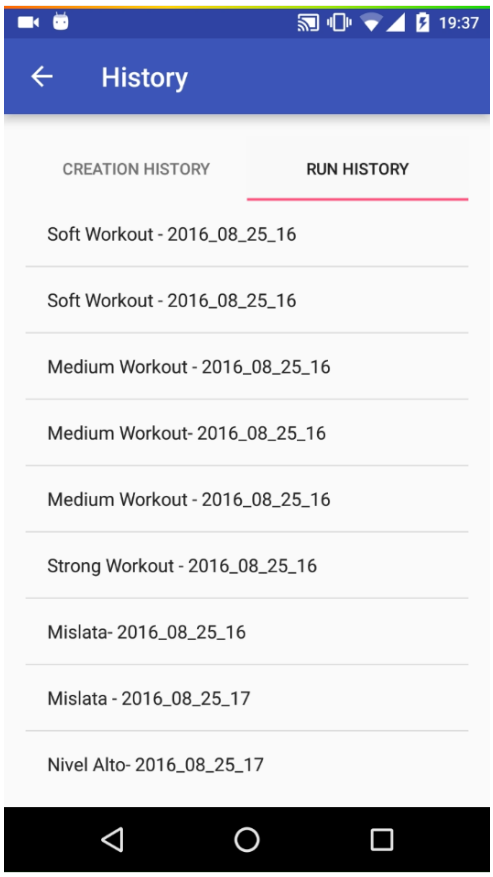

Figura 43 Actividad historia

Para esta actividad es necesario definir una actividad principal, un adapter, dos fragments y dos layouts. El adapter es una clase que hereda de FragmentPagerAdapter, modificada.

La actividad ViewPagerActivity es la actividad principal, que se ejecuta cuando se seleciona en el Navigation Drawer la opción de História. En ella simplemente necesitamos cargar su layout, crear el adaptador y aplicarlo al pager.

TabLayout es lo que mostrará si el usuario está en la historia de creación o en la de correr, por lo que lo forzamos para que sea visible, y le asociamos el pager definido.

La Figura 44 representa la actividad principal ViewPagerActivity

```
public class ViewPagerActivity extends AppCompatActivity {

    ViewPager pager;

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

        pager = (ViewPager) findViewById(R.id.pager);
        // Create a new PageAdapter and associate it to the ViewPager
        CustomFragmentPagerAdapter cfpa = new
CustomFragmentPagerAdapter(getSupportFragmentManager(),this);
        pager.setAdapter(cfpa);

        TabLayout tl = (TabLayout) findViewById(R.id.tabLayout);
        tl.setVisibility(View.VISIBLE);
        tl.setupWithViewPager(pager);
    }
}
```

Figura 44 Actividad principal de historia

En cuanto a la interfaz asociada a esta actividad, activity_view_pager.xml, en la Figura 45 se puede ver su árbol de componentes, consta del pager, que es donde se cargarán los datos, y del tablayout, donde aparecen los nombres de cada página.

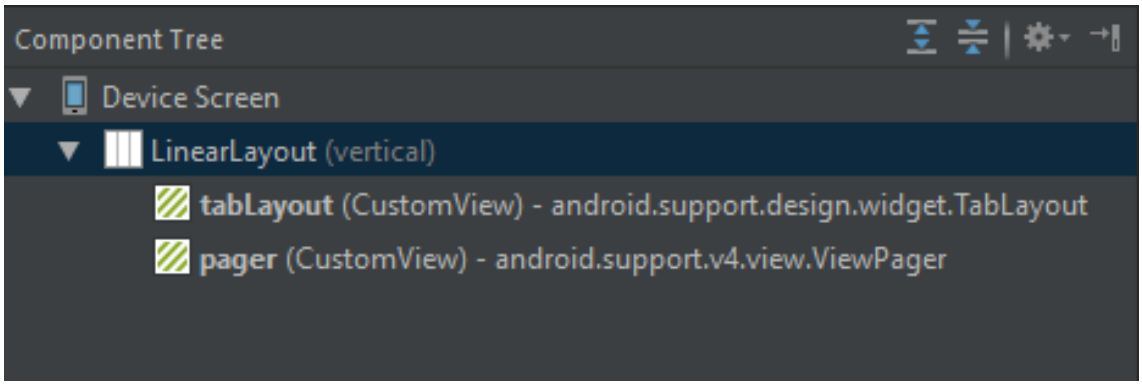

Figura 45 Arbol de componentes de Historia

La función de la clase adapter que he creado, CustomFragmentPagerAdapter, es mantener cada página en el FragmentManager. Los métodos más importantes, y que han de ser modificados son getItem, y getPageTitle.

getItem Figura 46 devolverá, teniendo en cuenta en la página en la que el usuario se encuentre (En este caso sólo dos páginas), qué clase hay que mostrar al usuario.

```
@Override
public Fragment getItem(int position) {
    Fragment result = null;
    switch(position) {
        case 0:
            result = new HistoryCreationFragment();
            break;
        case 1:
            result = new HistoryRunFragment();
            break;
    }
    return result;
}
```

Figura 46 CustomFragmentAdapter getItem

GetPageTitle, Figura 47, devuelve el título de cada una de estas páginas

```
@Override
public CharSequence getPageTitle(int position) {

    switch(position) {
        case 0:
            return "Creation History";
        case 1:
            return "Run History";
    }
    return super.getPageTitle(position);

}
```

Figura 47 CustomFragmentPagerAdapter getPageTitle

En las clases de cada página será necesario inflar la vista con la interfaz xml del fragment (fragment_history), que será la misma interfaz para las dos clases, ya que la información mostrada va a tener la misma estructura.

La estructura de la interfaz se puede observar en la Figura 48, consta de un RelativeLayout donde se cargarán las líneas de la base de datos, montado sobre un ScrollView que permitirá realizar scroll en el caso de que haya una gran cantidad de entradas.

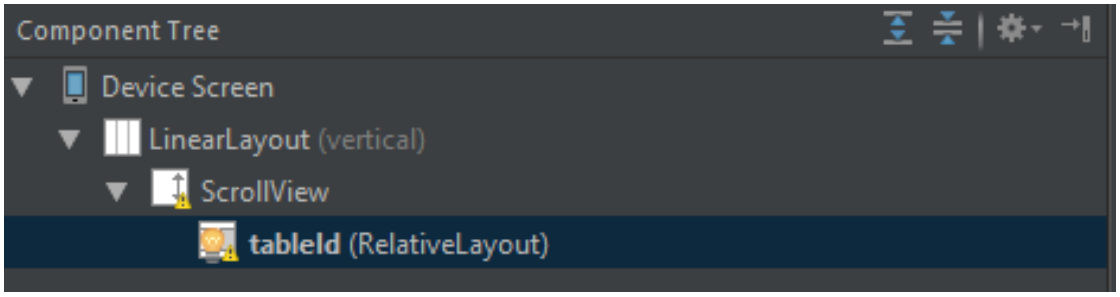

Figura 48 Estructura de fragment_history

Para el caso de la historia de ejercicios realizados, se accede a la base de datos utilizando un método que he definido en la clase MySqliteOpenHelper, getRunningHistory(); (getNamesAndDates en el caso de la página del histórico de creación de ejercicios).

Se recorre el ArrayList con los pares nombre-fecha obtenidos con el método getRunningHistory, y, habiendo obtenido la referencia al RelativeLayout, de cada par nombre-fecha creamos un TextView con el nombre y la fecha, y lo añadimos al RelativeLayout.

En el caso del histórico de creación de ejercicios, la estructura a seguir sería la misma, excepto que en vez de llamar a getRunningHistory(), se llama a getNamesAndDates()

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_history, null);
    ArrayList<Pair> workoutList;

    MySqliteOpenHelper sqlite = new
MySqliteOpenHelper(getActivity().getApplicationContext());
    workoutList = sqlite.getRunningHistory();

    String workoutName, workoutDate;

    if(!workoutList.isEmpty()){
        for(int i=0;i<workoutList.size();i++){
            RelativeLayout tl = (RelativeLayout) view.findViewById(R.id.tableId);
            TextView tv = new TextView(getActivity().getApplicationContext());
            workoutName = workoutList.get(i).getName();
            workoutDate = workoutList.get(i).getDate();
            tv.setText(workoutName+" - "+workoutDate);
            ViewPager.LayoutParams params = new ViewPager.LayoutParams();
            params.width = ViewGroup.LayoutParams.MATCH_PARENT;
            params.height = ViewGroup.LayoutParams.WRAP_CONTENT;
            tl.addView(tv,i,params);
        }
    }
    return view;
}
```

Figura 49 Creacion del Fragment

6. Resultados - Tests de usabilidad

Para realizar unas pruebas iniciales, he distribuido la aplicación entre varios amigos. He intentado cubrir el mayor número de usuarios posibles, por lo que entre los usuarios de prueba hay usuarios que realizan Hiits de forma habitual, usuarios que simplemente hacen running, usuarios acostumbrados a iPhone y usuarios acostumbrados a Android.

Los usuarios más habituados a utilizar Android no han tenido ningún problema con la aplicación, tanto los que están acostumbrados a realizar Hiits como los que no habían hecho nunca, estos últimos utilizaron el menú de ayuda para saber cómo realizar los Hiits, mientras que el primer grupo utilizó la aplicación directamente.

Como era de esperar, en cuanto a los usuarios acostumbrados a iPhone ha habido más variedad. En general, con el pequeño grupo de muestra que tenía, la mayoría ha necesitado ayuda para realizar el ejercicio, o lo ha configurado la primera vez de forma errónea.

A grosso modo, los resultados han sido positivos, exceptuando a los usuarios que no están acostumbrados a un móvil Android, pero al ser una aplicación exclusiva de Android por el momento, ellos tampoco entrarían en mi público objetivo.

7. Conclusiones

7.1. Conclusiones Técnicas

A pesar de tener algo de conocimiento en programación para Android, nunca había realizado una aplicación completa utilizando AndroidStudio, por lo que ha sido muy interesante aprender a utilizarlo, y me ha parecido un programa muy potente con

muchas opciones para autocompletar y generar código genérico que facilita enormemente la tarea de programar.

El uso de SQLite3 también ha sido muy interesante, un motor muy rápido y sencillo de utilizar, con muchas opciones tanto avanzadas para usuarios que tengan experiencia con bases de datos, como para usuarios que no hayan tenido apenas contacto con ellas, como es mi caso. La forma de conectar con las bases de datos me ha parecido muy intuitiva y sencilla, que sirve muy bien como inicio a programar con temas relacionados con bases de datos.

7.2 Conclusiones Personales

En general puedo decir que estoy contento con el trabajo realizado, esta es una aplicación que quería hacer por mi cuenta aunque se me hubiera asignado otro trabajo de fin de grado, y es una aplicación que planeo utilizar de forma habitual, por lo que para mí era importante que tuviera todas las características que me planteé inicialmente.

El seguir las pautas de diseño de Google para Material Design ha resultado tener una curva de aprendizaje mayor de lo que me esperaba, pero una vez avanzas lo suficiente hay muchas tareas en cuanto a programar que se realizan de una forma muy automática, por no hablar de que visual y usablemente, la aplicación es mucho más atractiva, por lo que los beneficios que se obtienen son mucho mayores que ese trabajo extra inicial que implica seguir las pautas de diseño de Google.

La estructura de datos que he escogido, a pesar de que creo que es la más adecuada teniendo en cuenta las opciones disponibles, creo que es mejorable, sobre todo en lo relacionado a la organización de la información. Utilizar una fila para cada iteración y repetir los nombres de los ejercicios es una medida que genera muchos problemas y que hay que solucionar con programación extra, por lo que quizás sería interesante utilizar varias tablas relacionadas entre sí para un diseño más intuitivo, y utilizar las restricciones de bases de datos nativas que ofrece SQLite.

7.1. Objetivos No Conseguidos

El mayor objetivo no conseguido ha sido el de implementar un sistema de Text-To-Speech para que el usuario pudiera recibir a través de los auriculares información extra por voz, referente al número de intervalos realizados, tiempo restante, etc.

Implementar este sistema resultó ser más costoso de lo anticipado y acarreaba utilizar librerías extra, por lo que simplemente no tuve tiempo suficiente.

7.2. Trabajo Futuro

El primer paso claro sería acabar de implementar el sistema Text-To-Speech para conseguir tener la App completa que me planteé hacer. Una vez esto estuviera acabado, se podría implementar un sistema GPS que guardara el recorrido del usuario, teniendo en cuenta inclinación de la pista, y añadir esto a la actividad de Histórico, para que además de saber qué ejercicio se realizó cada dia, el usuario pueda saber también dónde lo realizó.

Como ya he comentado, creo que el almacenamiento de datos es un aspecto mejorable, una mejor organización de la información podría quitarle gran parte del trabajo de

verificar la información obtenida al código, lo que podría resultar en una aplicación más ligera y rápida.

Para la gente que no tenga mucha experiencia con móviles Android o con ejercicios Hiit, me gustaría implementar en el futuro una actividad tutorial, que indique cual es la funcionalidad de cada botón, superpuesto en la propia actividad donde se ubica cada botón, y no solo en forma de ventana emergente en la pantalla principal.

También sería interesante subir la aplicación al Google Market, y así poder utilizar Google Ads para que, de forma no intrusiva, la aplicación mostrara anuncios. Con esto añadiría una opción dentro de la aplicación por la que los usuarios pudieran comprar una licencia para utilizar la versión de la aplicación sin anuncios. Además de eliminar los anuncios, los usuarios tendrían acceso a las funcionalidades que añadiera en el futuro. Teniendo la aplicación básica que se presenta en este proyecto con anuncios de forma gratuita. Para subir la aplicación a Google Market también sería necesario realizar una traducción a, como mínimo, el Español, trabajo que no requeriría casi esfuerzo, pero que abriría la aplicación a un gran número de usuarios potenciales.

8. Glosario de Términos

Hiit – *High Intensity Interval Training* – Entrenamiento de intervalos de alta intensidad. Un Hiit está formado por varios intervalos que se pueden repetir un número determinado de veces. Cada intervalo tiene dos secciones, Sprint y Descanso. Cada una de estas secciones se puede repetir un número determinado de veces. Estas secciones tienen asociada una duración (Generalmente unos 30 segundos para el Sprint y unos 60 segundos para el Descanso)

Android SDK – *Android Software Development Kit* – Conjunto de herramientas de desarrollo utilizadas para crear aplicaciones para la plataforma android. Incluye las librerías requeridas, un debugger, un emulador, APIs, código fuente de ejemplo y tutoriales para el sistema operativo Android.

Fab – *Floating Action Button* – Botón de Acción flotante. Botón circular que, siguiendo las guías de diseño de google, suele estar presente en la esquina inferior derecha de las pantallas principales de una aplicación Android.

NavBar – *Navigation Bar* – Cajón de Navegación. Menú lateral accesible mediante un swipe derecho en el borde izquierdo de la pantalla.

SQLite3 – Motor de base de datos incluido en Android

Intent – Objeto que representa la intención de realizar una acción. Se utiliza para solicitar una acción de otra componente de la aplicación. Es posible añadir información extra al intent que lanza una activdad, para que la actividad lanzada tenga esta información.

Activity – *Actividad* - Cada actividad representa una pantalla en una aplicación Android.

Fragment – Un Fragment representa una porción de la interfaz de usuario en una Actividad o pantalla. Se pueden combinar múltiples fragments en una misma Actividad o pantalla.

Listener– En java las acciones que puede realizar el usuario se denominan eventos. Estos eventos realizan a su vez una serie de acciones. Los Listeners se encargan de controlar estos eventos. Si sobreescribimos un metodo Listener podemos añadir acciones extra a las acciones que se lanzarían de forma automática con el evento seleccionado. El Listener onClick es el listener que maneja los clicks a un botón, por ejemplo.

Spinner General – Número de veces que hay que repetir el intervalo general de un Hiit (El que engloba al sprint y al descanso)

Spinner Sprint y Spinner Rest– Número de veces que hay que repetir la sección sprint y la sección descanso, respectivamente

Time Sprint y Time Rest– Segundos a realizar la sección sprint y la sección descanso, respectivamente.

9. Bibliografía

Varios Autores (2nd Edition) (Agosto 2015). Android Programming: The Big Nerd Ranch Guide. Amazon Digital Services LLC.

Neil Smyth (1st Edition)(Diciembre 2015). Android Studio Development Essentials – Android 6 Edition. EbookFrenzy.

9.2. Documentación Online

[1] Wikipedia, "App Store (iOS)," Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/App_Store_(iOS).

[2] Statista, "The Statistics Portal," June 2016. [Online]. Available: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[3] MyFitnessPal, "myFitnessPal," UnderArmour, [Online]. Available: https://www.myfitnesspal.com/.

[4] L. Drell, "Mashable," [Online]. Available: http://mashable.com/2015/01/04/connected-fitness-tech-ces/#B8s6nz6WikqP.

[5] CES Consumer Technology Asociation, "CES Consumer Technology Asociation," 2015. [Online]. Available: http://www.ces.tech/Conference/Conference-Tracks/Fitness-Tech.aspx.

[6] E. Clawrence, "Fitness Applications," [Online]. Available: http://emilyclawrence.weebly.com/history.html.

[7] J. Milivojevic, "The HIIT Trend," University Health News, 18 AUG 2016. [Online]. Available: http://universityhealthnews.com/daily/mobility-fitness/the-hiit-trend-high-intensity-interval-training/.

[8] A. Schlinger, "DailyBurn / Life," 2013. [Online]. Available: http://dailyburn.com/life/fitness/high-intensity-hiit-workout/.

[9] SimpleVision, "Google Play," 3 9 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.simplevision.workout.tabata.

[10] G. Regni, "Google Play," 7 12 2015. [Online]. Available: https://play.google.com/store/apps/details?id=com.ihunda.android.hiit.

[11] Caynax, "Google Play," 16 8 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.caynax.hiit.

[12] Runtastic, "Google Play," 16 8 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.runtastic.android.

[13] Pimpim Mobile, "Google Play," 28 5 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.pimpimmobile.atimer.

[14] Zumzet Workouts, "Google Play," 18 9 2015. [Online]. Available: https://play.google.com/store/apps/details?id=com.zumzet.fitness.hiit10min.

[15] Android Developer, "Floating Action Button," Google, [Online]. Available: https://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html.

[16] Google, "Material Design," [Online]. Available: https://material.google.com/.

[17] Android Developer, "Navigation Drawer," Google, [Online]. Available: https://developer.android.com/training/implementing-navigation/nav-drawer.html.

[18] Android Developer, "CardView," Google, [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/CardView.html.

[19] Android Developer, "Coordinator Layout," Google, [Online]. Available: https://developer.android.com/reference/android/support/design/widget/CoordinatorLayout.html.

[20] Android Developer, "Recycler View," Google, [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html.

[21] Android Developer, "Creating Lists and Cards," Google, [Online]. Available: https://developer.android.com/training/material/lists-cards.html.

[22] Android Developer, "RecyclerView.ViewHolder," Google, [Online]. Available: https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html.

[23] Android Developer, "Cursor," Google, 2015. [Online]. Available: https://developer.android.com/reference/android/database/Cursor.html.

[24] StackOverflow, "StackOverflow," [Online]. Available: http://stackoverflow.com/.

[25] W3Schools, "SQL tutorial," [Online]. Available: http://www.w3schools.com/sql/.

[26] D. d. Andres, "github," [Online]. Available: https://github.com/ddandres.

[27] Oracle, "Java API," 2016. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/.