Trabajo De Fin de Máster

Análisis y Desarrollo de una aplicación Android para entrenamiento personal

Máster Universitario en Investigación en Ingeniería de Software y Sistemas Informáticos

Antón Bouzán Cumplido

MÁSTER EN INVESTIGACIÓN EN INGENIERÍA DEL SOFTWARE Y SISTEMAS INFORMÁTICOS

ITINERARIO: Ingeniería del Software

CÓDIGO ASIGNATURA: 31105151

TÍTULO DEL TRABAJO: Análisis y Desarrollo de una aplicación Android para entrenamiento personal

TIPO DE TRABAJO: Tipo B, propuesto por el alumno

ALUMNO: Antón Bouzán Cumplido

DIRECTOR: Ismael Abad Cardiel

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO FIN DE MASTER

Fecha: 15/09/2021

Quién suscribe:

Autor: Antón Bouzán Cumplido
D.N.I/N.I.E/Pasaporte.: 53487083P

Hace constar que es la autor del trabajo:

TRABAJO FINAL DEL MÁSTER
Análisis y Desarrollo de una aplicación Android para entrenamiento personal

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores, se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

- ✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.
- ✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.
- ✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Fdo.

Impreso TFdM05_AutorPbl. Autorización de publicación y difusión del TFM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen

En la actualidad el mundo del deporte y la tecnología están fuertemente ligados. El avance en el tipo de sensores que los dispositivos móviles emplean hacen que el seguimiento del rendimiento y análisis de los deportistas sea cada vez más completo.

Este trabajo nace con la ambición de análizar e investigar qué abanico de aplicaciones y funcionalidades se ofrecen hoy en día con el fin de desarrollar una versión completa de una app de entrenamiento personal.

Palabras Clave

Aplicación, app, entrenamiento personal, acelerómetro, gps, clase, base de datos, servidor, Android, Android Studio, Java, PHP

Índice de contenido

1 Introducción

El presente documento pretende mostrar la investigación y posterior desarrollo de una aplicación Android enfocada al entrenamiento personal.

En la actualidad el mundo del fitness y del entrenamiento personal está en auge. La situación mundial que estamos viviendo, con confinamientos durante meses debido al Covid-19, ha hecho que el número de aplicaciones móviles disponibles haya crecido en este tiempo.

El motivo de este tipo de trabajo es que, a mi parecer, las funcionalidades que se ofrecen en la mayoría de casos son reducidas. Las aplicaciones están enfocadas a un tipo de trabajo: running, pesas, dietas, plannings de ejercicios... pero no existe una aplicación que combine todas estas ideas por lo que los usuarios deben usar varias aplicaciones si quieren disponer de unas funcionalidades completas.

En este trabajo se investigará y analizarán las funcionalidades y tecnologías que emplean las principales aplicaciones del mercado con el objetivo de poder desarrollar una app completa con lo mejor de cada una.

2 Objetivo

El objetivo del trabajo final de fin de máster es investigar, analizar, diseñar y realizar una aplicación completa Android combinando los conocimientos adquiridos en las asignaturas de *Computación Ubicua*, *Arquitectura Orientada a Servicios*, *Desarrollo de software seguro* y *Generación Automática de código*.

La práctica consiste en la realización de una aplicación completa que denominaremos *StayFit* que debe resolver la integración de un sistema de computación ubicua para recoger, procesar y generar información útil para el desarrollo de una actividad deportiva. Antes de llegar al desarrollo de la misma se ha realizado una labor de investigación y análisis de forma minuciosa de la actualidad del mundo del fitness y la tecnología. Analizando las principales apps disponibles para Android en la Play Store que ofrezcan planes entrenamientos.

Se ha desarrollado una aplicación Android con la que el usuario podrá tener un entrenador personal, ofreciendo planes de entrenamiento, dietas etc. El usuario podrá acceder y compartir con aplicaciones externas la información recogida de dichos entrenamientos a través de los diferentes sensores del dispositivo. También cuenta con un perfil en el que tendrá sus datos personales así como un histórico de peso y cálculo de IMC para saber su estado de forma. Todos los datos están recogidos en un servidor externo y el usuario podrá acceder a su perfil desde cualquier terminal (smartphone o tablet) con la aplicación instalada. Además también se incluye un apartado social en el que los usuarios de la aplicación pueden intercambiar mensajes y entrenamientos a través de sus perfiles.

3 Estructura del documento

El trabajo se divide en los siguientes capítulos y contenidos:

Capítulo 4 : Análisis e investigación. Apartado centrado en el estado del arte. En este capítulo se mostrará el trabajo de análisis e investigación realizado sobre las aplicaciones y tecnologías existentes actualmente y se fijarán los objetivos del desarrollo.

Capítulo 5: Desarrollo de la aplicación. A través de diagramas de clases, fragmentos de código y esquemas se mostrará el desarrollo de las principales funcionalidades de la app.

Capítulo 6: Ubicación de los programas fuente. Se muestra la ubicación de los distintos archivos que conforman la aplicación desarrollada.

Capítulo 7: Manual de uso. Se explica el proceso de instalación de la app en el dispositivo Android y el uso de las funcionalidades creadas.

Capítulo 8: Conclusiones. Conclusiones de todo lo aprendido y trabajado durante el desarrollo de este trabajo de fin de máster.

Capítulo 9: Futuras líneas de trabajo. Aspectos en los que el trabajo podría seguir avanzando para obtener unas mejores funcionalidades.

Capítulo 10: Bibliografía. Recursos bibliográficos utilizados para el desarrollo de este trabajo.

4 Análisis e investigación

En este apartado mostraré el trabajo de investigación realizado para un desarrollo completo de la aplicación final. Se mostrarán aplicaciones ya disponibles en el mercado analizando pros, contras, funcionalidades, diseño etc con el objetivo de llegar a una aplicación final completa y optimizada.

4.1 Aplicaciones

Workout Trainer: home fitness coach

Pensado para entrenar en casa sin apenas materiales de gimnasio, incluye tablas de ejercicios en los que se pueden personalizar. Entrenadores registrados con los que podemos interactuar. Opción premium para acceder a más entrenamientos.

Fitnness Builder

App que nos ofrece tablas de ejercicios personalizables. Es muy sencilla, seleccionas grupo muscular/circuito y se muestran los ejercicios. La interfaz está muy sobrecargada de imágenes haciéndola poco atractiva e intuitiva.

The Fitness Lab

Tablas de ejercicios. Incluye la opción de hablar por chat con el entrenador. El usuario puede crear post que se verán en el muro de su perfil. La interfaz es mejorable.

Home Workout - No Equipment

Aplicación muy popular durante la cuarentena. Entrenamientos sin necesidad de material adicional. Nos muestra tablas de ejercicios con el temporizador clásico. La interfaz es agradable e intuitiva.

Fitness & Bodybuilding

Interfaz intuitiva, entrenamientos agrupados por grupos musculares. Posibilidad de crear entrenamientos diarios/semanales.

Runnea Academy

Aplicación orientada al running, nos ofrece distintos niveles de intensidad/dificultad. Dispone de chat con los entrenadores. Podemos acceder a un histórico de los entrenamientos en donde se muestran gráficas con datos del ejercicio.

RunKeeper

Pensada para el running con esta aplicación podemos crear nuestros entrenamientos e invitar a amigos a unirse. Permite conexión a smartwatch para un control de las constantes vitales o mostrar información relevante durante el ejercicio.

Runastic

Aplicación de *Adidas* cuenta con cientos de miles de usuarios. Entrenamientos de running y retos que ayudarán al usuario al ir mejorando.

4.2 Comparativa interfaz/funcionalidades

Tras haber analizado las principales aplicaciones disponibles en el mercado se muestra a continuación una tabla comparativa con puntos a favor y en contra de cada una de ellas con el objetivo de poder añadir a nuestro desarrollo las mejores funcionalidades.

He decidido establecer una comparativa basada en puntos siguiendo los siguientes criterios para así obtener un análisis objetivo de las aplicaciones. Se asignarán 0, 1 ó 2 puntos en cada uno de los puntos principales a analizar y así se obtendrá la puntuación total de cada app. A continuación explico los apartados que se tendrán en cuenta:

- **Interfaz**: Para poder medir de forma objetiva la calidad de la interfaz que nos ofrece la aplicación se han seguido unos criterios aprendidos durante el grado de Ingeniería Informática de la *USC* en la asignatura *Interacción Persona Ordenador*. Se establecen los siguientes criterios:

    - *0 Puntos*: Elección de colores incorrecta según tablas de colores compatibles. Legilibilidad tediosa o nula. Navegación poco intuitiva. Las funcionalidades de los botones no están claras.

    - *1 Punto*: Correcta selección de colores, navegación y funcionalidades correctas.

    - *2 Puntos*: Buena selección de colores, navegación fluida y funcionalidades bien definidas.

- **Tipos de entrenamiento disponibles**: Se trata de tema principal del trabajo. Se analizará cuantos tipos de entrenamientos ofrece cada aplicación asignando los puntos de la siguiente manera:

    - *0 Puntos*: Planes de entrenamiento estáticos (no personalizables).

    - *1 Punto*: Planes de entrenamiento dinámicos (personalizables).

    - *2 Puntos*: Planes de entrenamiento dinámicos de más de un tipo (gimnasio y running, running y salto...).

- **Apoyo al entrenamiento**: Punto importante en una app enfocada al entrenamiento personal. Se establecen los siguientes criterios:

    - *0 Puntos*: No se ofrecen gráficas ni datos históricos de los entrenamientos realizados.

    - *1 Punto*: Se muestran gráficas o datos históricos de los entrenamientos realizados.

- *2 Puntos*: La app cuenta con gráficas y datos históricos d elos entrenamientos realizados.

- **Social**: En los tiempos que corren la interacción con otros usuarios es algo que se valora positivamente para disponer de una aplicación más dinámica y divertida. Se establecen los siguientes criterios:

  - *0 Puntos*: La aplicación no ofrece ningún tipo de funcionalidad que relacione a usuarios.

  - *1 Punto*: Se permite compartir entrenamientos con aplicaciones externas.

  - *2 Puntos*: Se permite compartir entrenamientos con aplicaciones externas. Se dispone de un servicio de mensajería entre usuarios/entrenadores o usuarios/usuarios para compartir entrenamientos o mensajes.

Aplicación	Interfaz	Tipos de entrenamiento	Apoyo al entrenamiento	Social	Total
Workout Trainer	1	1	0	0	**2**
FitnessBuilder	0	1	0	0	**1**
The Fitness Lab	1	1	2	2	**6**
Home Workout	1	1	2	0	**4**
Fitness & Bodybuilding	1	1	2	0	**4**
Runnea Academy	2	1	2	2	**7**
RunKeeper	2	1	2	1	**6**
Runastic	2	1	2	1	**6**

Con este análisis exhaustivo de las aplicaciones, podemos llegar a varias conclusiones clave:

- Ninguna de las aplicaciones más destacadas del mercado incluye entrenamientos en gimnasio junto a entrenamientos de running en exterior.

- Las aplicaciones orientadas al entrenamiento en gimnasio se basan en tablas de ejercicios ya creados en las que como mucho puedes modificar series y repeticiones.

- No existe ninguna funcionalidad que permita a los usuarios de la aplicación interactuar entre ellos.

Por lo tanto el objetivo de StayFit es , además de incluir entrenamientos agrupados por grupos musculares (como todas estas aplicaciones) añadir también entrenamientos en exterior (control por GPS de las distancias y tiempos recorridos) y crear una comunidad de usuarios/entreandores en la que se podrán compartir mensajes y entrenamientos desde el perfil de cada usuario.

De esta forma obtendremos una aplicación completa tanto en el apartado fitness como en el social.

4.3 Comparativa de sensores/permisos

En este apartado se analizan y comparan los elementos hardware y los permisos utilizados en las aplicaciones estudiadas con el objetivo de entender el funcionamiento y poder crear una aplicación desde cero. Para obtener esta información basta con acceder a la ficha de cada aplicación en la *Play Store* de Google donde se detallan los datos.

Aplicación	Sensores	Permisos
Workout Trainer	- GPS	- Almacenamiento - Sensores del wereable - Ubicación - Cámara - Fotos/multimedia/archivos - Control de vibración - Bluetooth - Conexión a internet
FitnessBuilder	- Ninguno	- Calendario - Contactos - Almacenamiento - Fotos/multimedia/archivos - Identidad - Control de vibración
The Fitness Lab	- Ninguno	- Teléfono - Almacenamiento - Fotos/multimedia/archivos - Cámara - ID de dispositivo - Control de vibración - Conexión a internet
Home Workout	- Ninguno	- Almacenamiento - Fotos/multimedia/archivos - Conexión a internet - Control de vibración

Aplicación	Sensores	Permisos
Fitness & Bodybuilding	- GPS	- Almacenamiento - Ubicación - Cámara - Fotos/multimedia/archivos - Conexión a internet - Información sobre la conexión WiFi
Runnea Academy	- GPS	- Historial de aplicaciones y del dispositivo - Contactos - Almacenamiento - Ubicación - Cámara - Fotos/multimedia/archivos - Conexión a internet - Micrófono - Sensores del wereable
RunKeeper	- GPS	- Contactos - Almacenamiento - Ubicación - Cámara - Fotos/multimedia/archivos - Conexión a internet - Sensores del wereable - Bluetooth
Runastic	- GPS	- Historial de aplicaciones y del dispositivo - Contactos - Almacenamiento - Ubicación - Cámara

		- Fotos/multimedia/archivos - Conexión a internet - Micrófono - Sensores del wereable - Bluetooth

Podemos destacar que al haber analizado qué permisos utilizan las aplicaciones estudiadas una gran mayoría solicitan permisos que para nada son necesarios con las funcionalidades que nos ofrecen. En las primeras 5 aplicaciones, que se basan en tablas de ejercicios y temporizadores se requieren permisos de ubicación, cámara, ID del dispositivo e incluso control de llamadas. En las últimas 3, las dedicadas al running, se entiende que necesiten permisos como la ubicación (para poder realizar el trazado de la ruta con GPS).

Los sensores utilizados solo han sido el GPS en todas las aplicaciones que tiene permiso para acceder a la ubicación. No utilizando por ejemplo el acelerómetro o giroscopio que son de gran utilidad en lo que a movilidad física se refiere.

La aplicación que desarrollaremos a continuación solo empleará los sensores y permisos estrictamente necesarios para un funcionamiento óptimo.

5 Desarrollo de la aplicación

En este apartado se explicará la planificación y requisitos, los recursos de programación utilizados, la descripción de la solución propuesta junto al alcance y limitaciones así como los casos de prueba realizados.

5.1 Planificación y requisitos

En este apartado se verá la planificación del proyecto así como el análisis de requisitos propuestos. Como metodología de desarrollo he decidido utilizar el desarrollo en cascada, ya que es una metodología que a he utilizado en multitud de proyectos durante la carrera de ingeniería informática y en las prácticas en empresa.

En primer lugar analizaremos el Diagrama de Gantt:

Nombre de la tarea	Duración	Fecha de inicio
	424h 10m	**2021/07/01 09:00**
Iniciación/Planificación	18h	**2021/07/01 09:00**
Crear Planificación	1h 30m	2021/07/01 09:00
Análisis de requisitos	2h	2021/07/01 10:30
Planificación de Seguridad	3h	2021/07/01 12:30
Instalación IDE	1h	2021/07/05 09:00
Configuración IDE	1h	2021/07/05 10:00

Análisis e Investigación	8h	**2021/07/05 11:00**
Búsqueda de aplicaciones	2h	2021/07/05 11:00
Análisis de las funcionalidades	2h	2021/07/05 15:00
Análisis de los sensores hardware y permisos	2h	2021/07/05 17:00
Investigación de tecnologías empleadas	2h	2021/07/06 09:00

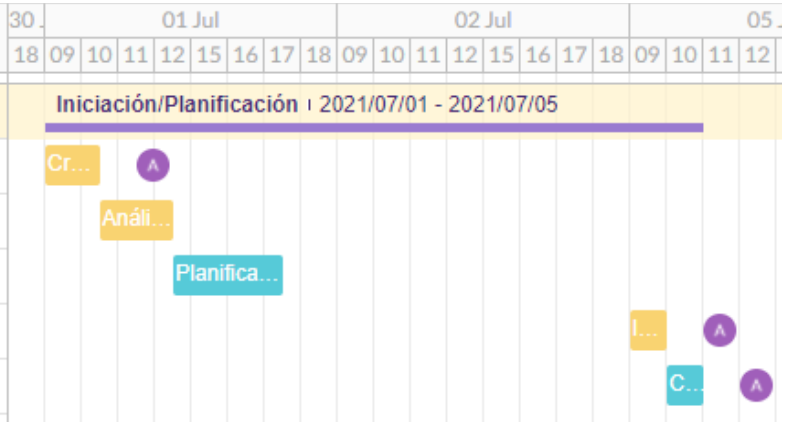
Desarrollo	68h	**2021/07/12 09:00**
Inicio de Sesión	2h 30m	2021/07/12 09:00
Registro	1h	2021/07/12 11:30
Sensor Acelerómetro	10h	2021/07/12 12:30
Sensor GPS	10h	2021/07/14 11:00
Entrenamientos Personalizados	10h	2021/07/15 11:00
Histórico de datos	8h	2021/07/16 09:00
Exportar información	3h	2021/07/19 09:00
Perfil Social	4h	2021/07/20 09:00
Chat de mensajería	12h	2021/07/21 09:00

Casos de Prueba	5h	**2021/07/26 09:00**
Caso de Prueba 1	1h	2021/07/26 09:00
Caso de Prueba 2	1h	2021/07/26 10:00
Caso de Prueba 3	1h	2021/07/26 11:00
Caso de Prueba 4	1h	2021/07/26 12:00
Caso de Prueba 5	1h	2021/07/26 15:00
Añadir una tarea | Añadir un hito		
Documentación	34h 30m	**2021/08/09 09:00**
Redacción de informe	20h	2021/08/09 09:00
Revisión de informe y app	2h 30m	2021/08/13 09:00
Añadir una tarea | Añadir un hito		
Entrega	10m	**2021/09/14 09:00**
Nueva tarea hermana	10m	2021/09/14 09:00

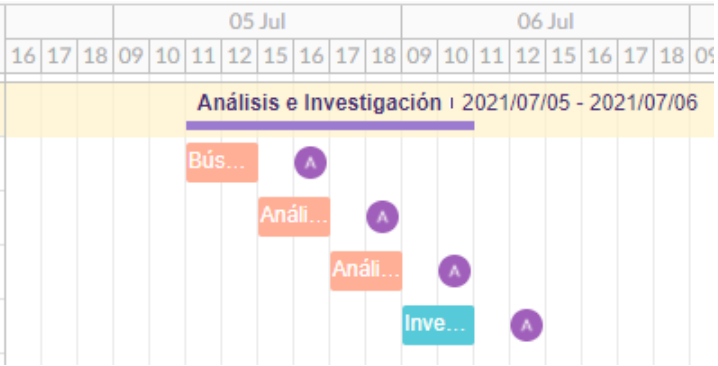
En este apartado se puede ver el desglose de cada una de las actividades con su tiempo de dedicación.

A continuación se muestra la línea temportal del Diagrama de Gantt para tener un mayor detalle:

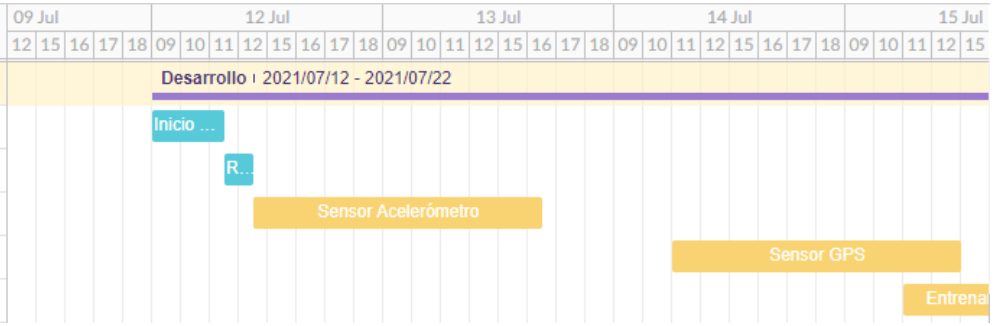
Nombre de la tarea	Duración	Fecha de inicio
Iniciación/Planificación	18h	**2021/07/01 09:00**
Crear Planificación	1h 30m	2021/07/01 09:00
Análisis de requisitos	2h	2021/07/01 10:30
Planificación de Seguridad	3h	2021/07/01 12:30
Instalación IDE	1h	2021/07/05 09:00
Configuración IDE	1h	2021/07/05 10:00



Nombre de la tarea	Duración	Fecha de inicio
Análisis e Investigación	8h	**2021/07/05 11:00**
Búsqueda de aplicaciones	2h	2021/07/05 11:00
Análisis de las funcionalidades	2h	2021/07/05 15:00
Análisis de los sensores hardware y permisos	2h	2021/07/05 17:00
Investigación de tecnologías empleadas	2h	2021/07/06 09:00

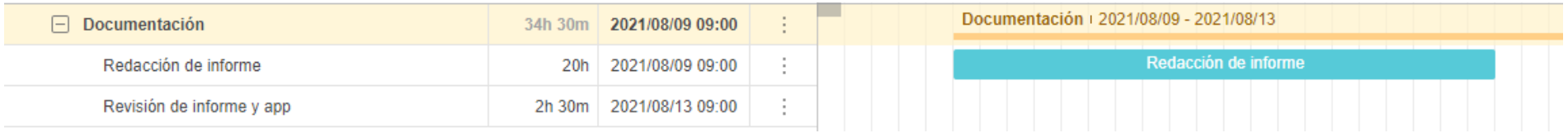


Nombre de la tarea	Duración	Fecha de inicio
Desarrollo	68h	**2021/07/12 09:00**
Inicio de Sesión	2h 30m	2021/07/12 09:00
Registro	1h	2021/07/12 11:30
Sensor Acelerómetro	10h	2021/07/12 12:30
Sensor GPS	10h	2021/07/14 11:00
Entrenamientos Personalizados	10h	2021/07/15 11:00



Histórico de datos	8h	2021/07/16 09:00	⋮
Exportar información	3h	2021/07/19 09:00	⋮
Perfil Social	4h	2021/07/20 09:00	⋮
Chat de mensajería	12h	2021/07/20 17:00	⋮

Nombre de la tarea	Duración	Fecha de inicio	+
Casos de Prueba	5h	**2021/07/26 09:00**	⋮
Caso de Prueba 1	1h	2021/07/26 09:00	⋮
Caso de Prueba 2	1h	2021/07/26 10:00	⋮
Caso de Prueba 3	1h	2021/07/26 11:00	⋮
Caso de Prueba 4	1h	2021/07/26 12:00	⋮
Caso de Prueba 5	1h	2021/07/26 15:00	⋮


Documentación 34h 30m 2021/08/09 09:00
Redacción de informe 20h 2021/08/09 09:00
Revisión de informe y app 2h 30m 2021/08/13 09:00
Documentación 2021/08/09 - 2021/08/13
Redacción de informe


Entrega 10m 2021/09/14 09:00
Nueva tarea hermana 10m 2021/09/14 09:00
Entrega 2021/09/14 - 2021/09/14

Análisis de requisitos

Tras haber analizado en el capítulo 4 las principales aplicaciones del mercado del ámbito del fitness y el running podemos definir los siguientes requisitos para nuestra aplicación:

ID requisito	Requisito
R_1	Correcta selección de colores según tablas de compatibilidad.
R_2	Crear una navegación fluida e intuitiva.
R_3	Ofrecer tipos de entrenamientos con material de gimnasio
R_4	Ofrecer tipos de entrenamientos sin material necesario
R_5	Ofrecer entrenamientos de running
R_6	Seguimiento de las actividades de competición. Análisis del rendimiento
R_7	Seguimiento de constantes vitales (peso, IMC, etc.)
R_8	Integración con planes de nutrición
R_9	Integración de información geo-posicional en los registros y en las planificaciones
R_10	Generación de información estadística de seguimiento de las condiciones personales
R_11	Consulta de datos sobre las actividades realizadas
R_12	Adaptación a las particularidades de una actividad individual o en equipo, o a las particularidades de una determinada actividad deportiva
R_13	Servicios para compartir información con otras aplicación. Exportación de información
R_14	Servicio de mensajería entre usuarios registrados

5.2 Tecnologías empleadas

Este es uno de los apartados más importantes para el desarrollo de un proyecto informático. He analizado las principales tecnologías que se emplean para el desarrollo de una aplicación Android y he llegado a las siguientes conclusiones:

- **La aplicación se tratará de una aplicación nativa**. Existen multitud de herramientas que permiten crear aplicaciones basadas en web o incluso sin conocimiento de programación. Para obtener un rendimiento óptimo al usarse varios de los sensores disponibles en el smartphone, una aplicación nativa es la mejor decisión. El lenguaje de programación empleado será Java.

- **Se utilizará un servidor web Apache**. Este tipo de servidor es muy popular y he trabajo con el durante años, tanto para asignaturas de la carrera y máster como para proyectos personales. Cuenta con una gran comunidad de usuarios de soporte.

- **Base de datos MySQL**. Tras haber trabajo con varios Sistemas de gestión de Bases de Datos (SGDB) como MariaDB, PostgreSQL me decanto por MySQL por la sencillez de instalación, configuración y buen rendimiento que ofrece. Además y de igual forma que sucede con Apache cuenta con una comunidad tan grande que la documentación disponible cubre cualquier tipo de necesidad.

- **Programación del servidor en PHP**. Python es la alternativa que tenía pensado utilizar para el backend del servidor, ambos lenguajes cuentan con un soporte amplio y buen rendimiento. He programado en ambos lenguajes pero tengo más experiencia con PHP para la interacción con bases de datos y aplicaciones Android.

- **Mensajería con JSON, Java y PHP**. Para crear el chat entre usuarios existen varias opciones. Firebase de Google es una plataforma en la nube para el desarrollo de aplicaciones web y móvil. Tenemos la opción de utilizar sus herramientas para crear el chat y enviar las notificaciones. Pero personalmente he decidido no compartir con una empresa externa (Google en este caso) las conversaciones personales de los usuarios por lo que desarrollaré el chat utilizando Java (app Android) y JSON junto a PHP para gestionar mensajes enviados y recibidos.

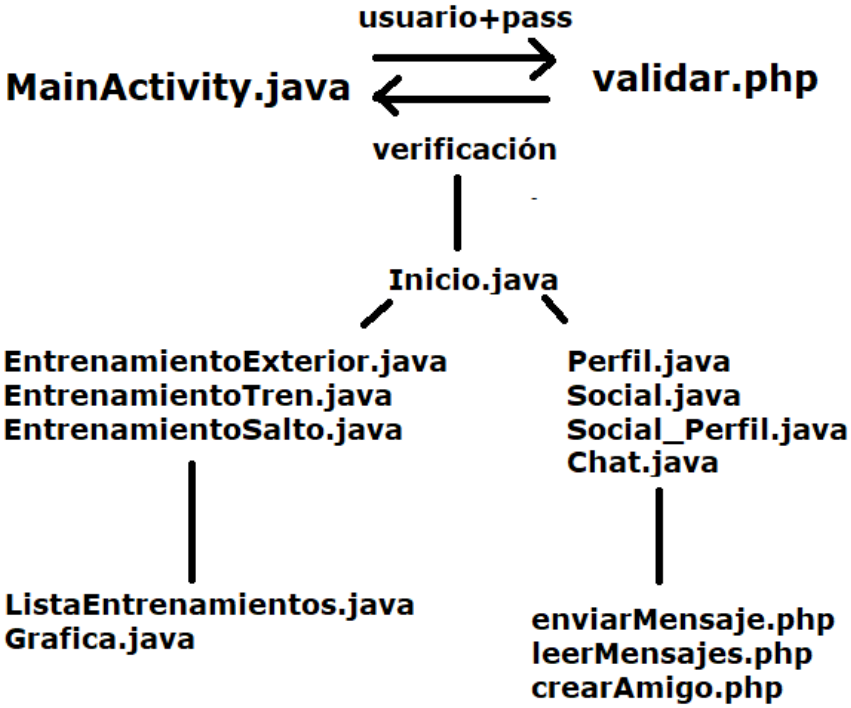
5.3 Recursos de programación utilizados

Tras analizar todas las opciones para poder desarrollar la aplicación, decidí utilizar mis conocimientos en el desarrollo de aplicaciones para Android para proponer la solución al trabajo.

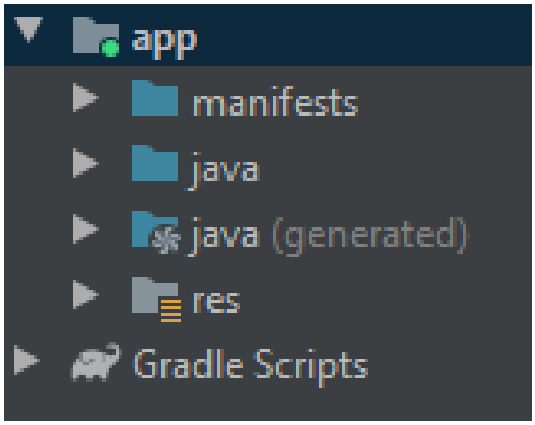
Como IDE he utilizado *Android Studio* en su versión más reciente. Se trata del editor oficial de *Google* para su sistema operativo por lo que su funcionamiento y librerías disponibles son las mejores del mercado. El lenguaje elegido para programar ha sido *Java*. Para ir probando las funcionalidades he empleado la opción que ofrece el IDE para correr la aplicación directamente en mi smartphone, teniendo así una visión realista en vez de utilizar simuladores virtuales.

5.4 Descripción de la solución

A continuación se muestra un diagrama de clases para explicar el funcionamiento de la aplicación:

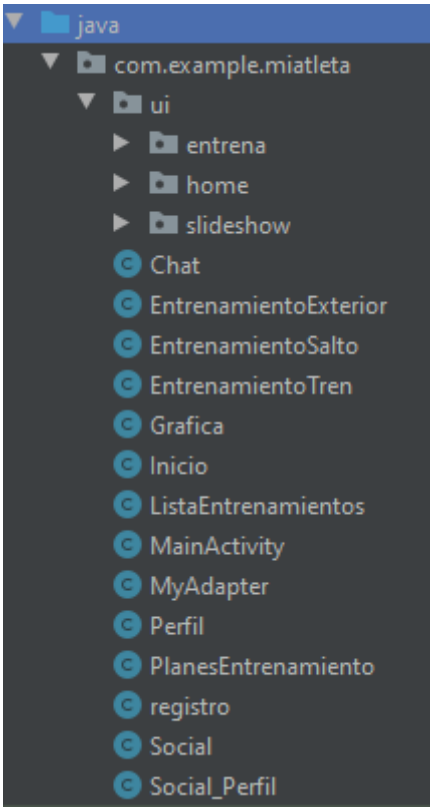


La solución propuesta es una aplicación Android que cumpla los requisitos establecidos en el capítulo anterior. La estructura de la aplicación es la siguiente:



Tres carpetas principales: *manifests, java* y *res.* En el siguiente apartado se explica en detalle cuál es el contenido y objetivo de cada carpeta.

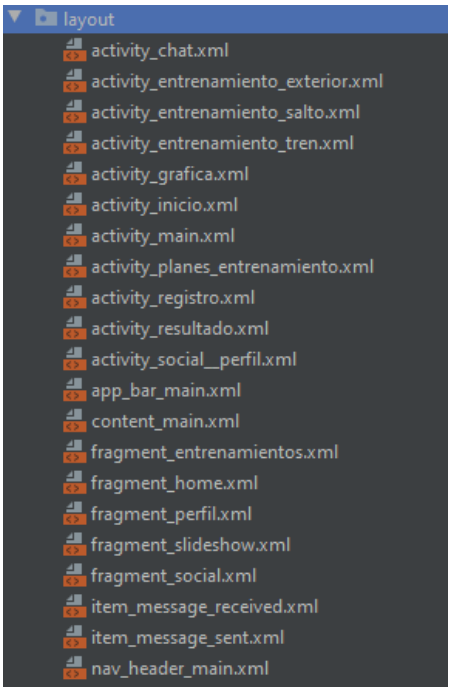
En la carpeta *Java* se encuentran las clases Java que se encargan del funcionamiento de la aplicación.



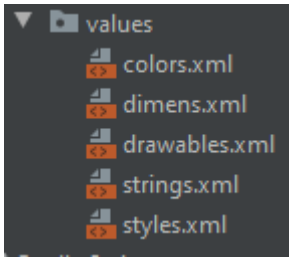
En la carpeta *res* del inglés resources (recursos) se encuentran las carpetas: *drawable, layout* y *values.*

La carpeta *drawable* contiene las imágenes y figuras que se utilizan en la app.

La carpeta *layout* contiene los archivos xml que forman la interfaz de usuario.



La carpeta *values* contiene los archivos xml con los valores de los elementos de la interfaz (colores, texto etc).



En la carpeta *manifests* se encuentra el archivo *AndroidManifest.xml* donde se establecen algunas características del proyecto. Destacar que se incluyen los permisos de localización y vibración:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-feature android:name="android.hardware.location.gps" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

A continuación veremos en detalle y explicaremos el código de cada una de las clases junto a su layout asociado.

MainActivity

Esta clase muestra al usuario una pantalla de login para el inicio de sesión en la aplicación junto a un botón para ejecutar el el registro.

En primer lugar se instancian los objetos java que componen la interfaz para poder interactuar con ellos y se les añade un listener al botón “Iniciar Sesión” y ”Registrar”.

Como se puede ver en el siguiente fragmento de código, cuando el usuario presione el botón de nuevo registro se abrirá una nueva actividad con un formulario para completar los datos.

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

    //Identificacion de elementos del layout
    textUsuario=(EditText)findViewById(R.id.textUsuario);
    textNombre=(EditText)findViewById(R.id.textNombre);
    textEmail=(EditText)findViewById(R.id.textEmail);
    textPass=(EditText)findViewById(R.id.textPass);

    //Listener para el click del boton
    botonRegistrar=(Button) findViewById(R.id.botonRegistrar);
    botonRegistrar.setOnClickListener((view) → {
            Intent intent = new Intent(getApplicationContext(),registro.class);
            startActivity(intent);
    });

    botonLogin=(Button) findViewById(R.id.botonLogin);
    botonLogin.setOnClickListener((view) → {
            ejecutarLogin( URL: ipGlobal+"/conexionmiatleta/validar2.php");
    });
}
```

Por otro lado al presionar el boton de login se llama a la función *ejecutarLogin()*:

```
//metodo para envar las peticiones al servidor
private void ejecutarLogin(String URL){
    //declara una peticion
    StringRequest stringRequest = new StringRequest(Request.Method.POST, URL, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            int exito=0;
            if(!response.isEmpty()){
               int size = response.length();
               //si devuelve [] dos caracteres significa que no hay id con esos datos, login incorrecto
               if(size==2){
                   Toast.makeText(getApplicationContext(),  text: "Usuario o contraseña incorrecto", Toast.LENGTH_LONG).show();
               }else if(size==13) {
                  idUsuario=  Integer.parseInt(response.substring(8,10));
                  exito=1;
               }else{
                   idUsuario=  Integer.parseInt(response.substring(8,9));
                   exito=1;
               }
               //si existe usuario cargamos la actividad principal
               if(exito==1) {
                   //Toast.makeText(getApplicationContext(), String.valueOf(idUsuario), Toast.LENGTH_SHORT).show();
                   Intent intent = new Intent(getApplicationContext(), Inicio.class);
                   startActivity(intent);
               }
```

Esta función recibe como parámetros el contenido de la caja de texto para el nombre de usuario y la contraseña y se los pasa al archivo php del servidor donde se ejecutará una consulta SQL con el objetivo de comprobar si el par usuario-contraseña introducido existe y por lo tanto el inicio de sesión es correcto.

```
               }else{
                   Toast.makeText(getApplicationContext(),  text: "Usuario o contraseña incorrecto", Toast.LENGTH_LONG).show();
               }

           }

           //en caso de que algo salio mal
       }, new Response.ErrorListener() {
           @Override
           public void onErrorResponse(VolleyError error) {
               Toast.makeText(getApplicationContext(), error.toString(), Toast.LENGTH_LONG).show();
           }
       }){
           //indicar los parametros que vamos a enviar

           @Override
           protected Map<String, String> getParams() throws AuthFailureError {
               Map<String, String> parametros = new HashMap<~>();
               //enviar los parametros (tomar en cuenta el orden en como entan en el PHP)
               parametros.put( k: "usuario", textUsuario.getText().toString());
               parametros.put( k: "password", textPass.getText().toString());
               return parametros;
           }
       };
```

El código php que realiza la consulta SQL es muy sencillo:

```php
<?php

include "conexion.php";
$usuario=$_POST['usuario'];
$password=$_POST['password'];

//generamos la consulta
$sql = "SELECT * FROM usuarios where usuario like '".$usuario."' and password like '".$password."' ";

if(!$result = mysqli_query($conn, $sql)) die();
$valores = array(); //creamos un array

if( $row = $result->fetch_assoc()) {
    session_start();
    $id= $row["id"];
    $valores[] = array('id'=> $id);
}

//desconectamos la base de datos
$close = mysqli_close($conn)
or die("Ha sucedido un error inexperado en la desconexion de la base de datos");


//Creamos el JSON
$json_string = json_encode($valores);

?>
```

Nos devuelve un *json* con el id del usuario que acaba de realizar el login correcto por lo que tal y como se muestra en la primera parte del código de la función *ejecutarLogin()* si la respuesta no esta vacía se procede a abrir la actividad con el menú principal de la aplicación.

A continuación se muestra una parte del código xml del diseño de la interfaz del usuario:

Registro

Esta clase nos permite ejecutar el registro de un nuevo usuario en la base de datos. Se instancian los elementos de la interfaz y se añade un listener al botón de registro a través del cual llamamos a la función *ejecutarRegistro()*.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_registro);

    //Identificacion de elementos del layout
    textUsuario=(EditText)findViewById(R.id.textUsuario);
    textNombre=(EditText)findViewById(R.id.textNombre);
    textEmail=(EditText)findViewById(R.id.textEmail);
    textPass=(EditText)findViewById(R.id.textPass);
    textPass2=(EditText)findViewById(R.id.textPass2);

    //Listener para el click del boton
    botonRegistrar=(Button) findViewById(R.id.botonRegistrar);
    botonRegistrar.setOnClickListener((view) -> {
            if(!textPass.getText().toString().equals(textPass2.getText().toString())){
                Toast.makeText(getApplicationContext(),  text: "¡Las contraseñas no coiciden!", Toast.LENGTH_LONG).show();

            }else {
                ejecutarRegistro( URL: MainActivity.ipGlobal+"/conexionmiatleta/insertarDatos.php");
                Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                startActivity(intent);
            }
    });
```

De igual forma que sucede con el login del usuario, se pasa como parámetros al script php los datos introducidos en el formulario:

```
    @Override
    protected Map<String, String> getParams() throws AuthFailureError {
        Map<String, String> parametros = new HashMap<~>();
        //enviar los parametros (tomar en cuenta el orden en como entan en el PHP)
        parametros.put( k: "nombre", textNombre.getText().toString());
        parametros.put( k: "usuario", textUsuario.getText().toString());
        parametros.put( k: "email", textEmail.getText().toString());
        parametros.put( k: "password", textPass.getText().toString());
        return parametros;
    }
};
```

El código php inserta en la tabla usuarios los datos del formulario anterior:

```php
<?php
include 'conexion.php';

$nombre=$_POST['nombre'];
$usuario=$_POST['usuario'];
$email=$_POST['email'];
$password=$_POST['password'];

$consulta="INSERT INTO usuarios (nombre,usuario,email,password) values ('".$nombre."','".$usuario."','".$email."','".$password."')";

mysqli_query($conn,$consulta) or die (mysqli_error());
?>
```

A continuación se muestra una parte del código xml del diseño de la interfaz del usuario:

```xml
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="15dp"
    android:layout_centerHorizontal="true"
    android:text="Registro"
    android:textSize="35dp"
    android:textStyle="bold"
    />

<EditText
    android:id="@+id/textNombre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Nombre completo"
    android:layout_marginLeft="30dp"
    android:layout_marginRight="30dp"
    android:layout_marginTop="80dp" />
```

Inicio

Esta clase nos ofrece el menú lateral y principal de la aplicación a través de la cual el usuario podrá navegar por las diferentes secciones:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_inicio);

    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    FloatingActionButton fab = findViewById(R.id.fab);
    fab.setVisibility(View.INVISIBLE);
    fab.setOnClickListener((view) → {
            Snackbar.make(view, text: "Replace with your own action", Snackbar.LENGTH_LONG)
                    .setAction( text: "Action", listener: null).show();
    });
    DrawerLayout drawer = findViewById(R.id.drawer_layout);
    NavigationView navigationView = findViewById(R.id.nav_view);
    // Passing each menu ID as a set of Ids because each
    // menu should be considered as top level destinations.
    mAppBarConfiguration = new AppBarConfiguration.Builder(
            R.id.nav_inicio, R.id.nav_entrenamientos, R.id.nav_social,R.id.nav_perfil)
            .setDrawerLayout(drawer)
            .build();
    NavController navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment);
    NavigationUI.setupActionBarWithNavController( activity: this, navController, mAppBarConfiguration);
    NavigationUI.setupWithNavController(navigationView, navController);
```

El fragmento *fragment_home* muestra la interfaz principal de la aplicación:

```
<androidx.cardview.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view2"
    android:layout_marginTop="300dp"
    android:layout_marginLeft="15dp"
    android:layout_marginRight="15dp"
    android:layout_width="match_parent"
    android:layout_height="150dp"
    card_view:cardCornerRadius="4dp">

    <TextView
        android:id="@+id/info_text2"
        android:layout_width="match_parent"
        android:text="Conecta con otros usuarios"
        android:textSize="20dp"
        android:textStyle="bold"
        android:layout_marginTop="15dp"
        android:layout_marginLeft="15dp"
```

Entrenamientos

Esta clase nos ofrece un menú con tres tipos de entrenamientos. En primer lugar se instancian los botones de la interfaz para poder interactuar con ellos. Al tratarse de una clase que extiende de la clase *Fragment* tenemos que establecer que *fragment_entrenamientos* será el contenido de la misma:

```
public class EntrenamientoFragment extends Fragment {

    private EntrenamientoViewModel entrenamientoViewModel;

    Button botonClasico,botonExterior,botonSaltos;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        entrenamientoViewModel =
                ViewModelProviders.of( fragment: this).get(EntrenamientoViewModel.class);
        View root = inflater.inflate(R.layout.fragment_entrenamientos, container,  attachToRoot: false);
        //final TextView textView = root.findViewById(R.id.text_gallery);
        entrenamientoViewModel.getText().observe(getViewLifecycleOwner(), (Observer) (s) -> {
            // textView.setText(s);
        });
```

A continuación se crean los *listeners* para cada uno de los botones de los diferentes tipos de entrenamientos. Cada uno de los cuales empezará una actividad diferente dependiendo de cuál sea el botón accionado:

```
//Entrenamiento clasico

botonClasico = (Button) root.findViewById(R.id.botonClasico);
botonClasico.setOnClickListener((v) -> {
        Intent intent = new Intent(getActivity().getApplicationContext(), EntrenamientoSalto.class);
        startActivity(intent);
});

//Entrenamiento en exterior


botonSaltos = (Button) root.findViewById(R.id.botonExterior);
botonSaltos.setOnClickListener((v) -> {
        Intent intent = new Intent(getActivity().getApplicationContext(), EntrenamientoExterior.class);
        startActivity(intent);
});


//Entrenamiento de saltos


botonExterior = (Button) root.findViewById(R.id.botonEntrenamientoSaltos);
botonExterior.setOnClickListener((v) -> {
        Intent intent = new Intent(getActivity(), EntrenamientoSalto.class);
        startActivity(intent);
});
```

Esta parte de código se corresponde con *fragment_entrenamientos.xml donde se establecen los componentes gráficos de la interfaz:*

```
<Button
    android:id="@+id/botonClasico"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_marginTop="40dp"
    android:background="@drawable/img_clasico"
    android:text="" />

<Button
    android:id="@+id/botonExterior"
    android:layout_width="match_parent"
    android:layout_height="200dp"
    android:layout_marginTop="280dp"
    android:background="@drawable/img_exterior"
    android:text="" />

<Button
    android:id="@+id/botonEntrenamientoSaltos"
    android:layout_width="match_parent"
```

EntrenamientoSalto

Para este tipo de entrenamiento se utiliza el sensor **Acelerómetro**. Obteniendo los valores detectados se me ocurrió crear un entrenamiento de tren inferior basado en saltos. Detectando la aceleración del dispositivo podemos detectar cuántos saltos se han realizado.

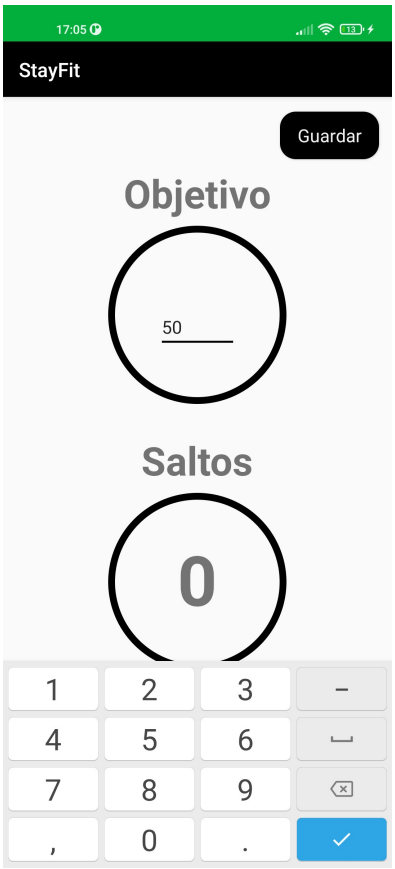
En primer lugar se instancia el acelerómetro accediendo a los sensores del dispositivo. A continuación creo, si no existe, la tabla *historicoSaltos* para almacenar los registros del entrenamiento.

```
//acelerometro
sensorManager=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
acelerometro=sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener( listener: EntrenamientoSalto.this,acelerometro, SensorManager.SENSOR_DELAY_NORMAL);

/// BD
myDB= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE,  factory: null);

myDB.execSQL(
        "CREATE TABLE IF NOT EXISTS historicoSaltos (saltos INT, fecha DATE)"
);
```

Tras registrar un listener para detectar los cambios en el sensor se establece dentro de la función *onSensorChanged* el comportamiento tras detectar movimiento. Lo que he hecho es registrar los valores en cada momento del acelerómetro, y cuando se detecte un cambio superior a 10m/s^2 se incrementará el contador de saltos en una unidad.

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    valorXanterior=valorX;
    valorYanterior=valorY;
    valorZanterior=valorZ;
    //
    textx.setVisibility(View.INVISIBLE);
    texty.setVisibility(View.INVISIBLE);
    textz.setVisibility(View.INVISIBLE);
    //
    valorX=sensorEvent.values[0];
    valorY=sensorEvent.values[1];
    valorZ=sensorEvent.values[2];
    int aux =0;
    if(comenzar==1){
        if(valorX-valorXanterior>10||valorX-valorXanterior<-10){
            contadorSaltos=contadorSaltos+1;
            textoSaltos.setText(""+contadorSaltos);
            aux=objetivo-contadorSaltos;
            if(aux>0) {
                textoFaltaObjetivo.setText("Faltan " + aux + " para el objetivo");
                textoFaltaObjetivo.setTextColor(Color.rgb( red: 244, green: 69, blue: 69));
            }else{
                textoFaltaObjetivo.setText("¡Entrenamiento finalizado!");
                textoFaltaObjetivo.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
                //vibramos para avisar
                Vibrator vibrator = (Vibrator)getApplicationContext().getSystemService(Context.VIBRATOR_SERVICE);
                vibrator.vibrate( milliseconds: 1000);
```

El usuario podrá utilizar el botón superior derecho para editar el objetivo del entrenamiento:

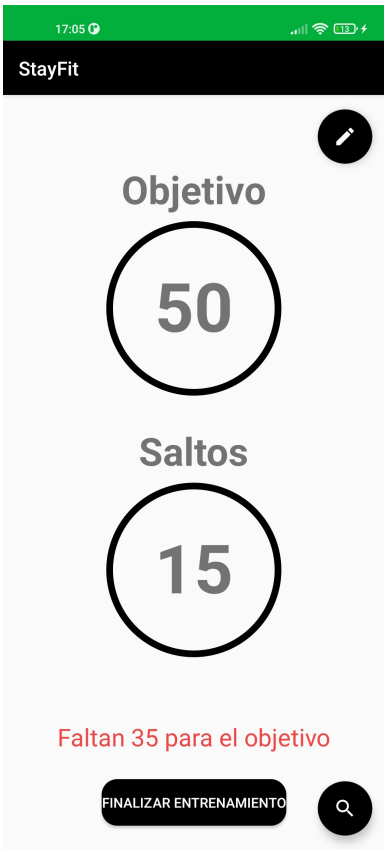


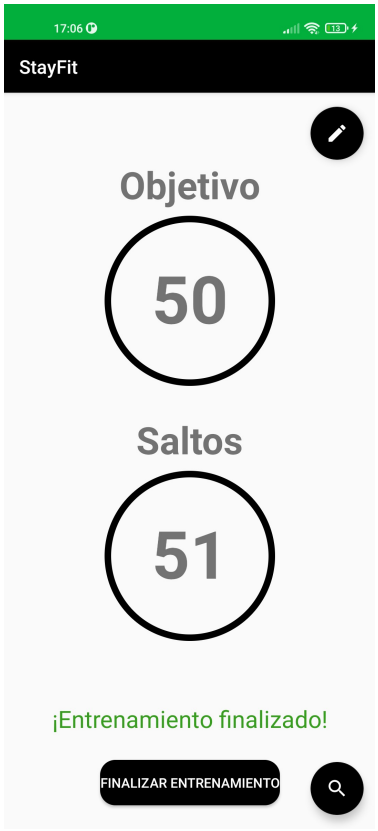
Este es el código correspondiente a la funcionalidad de editar el objetivo y guardarlo:

```
botonEditarSaltos.setOnClickListener((view) → {
        cajaObjetivo.setVisibility(View.VISIBLE);
        textoObjetivo.setVisibility(View.INVISIBLE);
        botonEditarSaltos.setVisibility(View.INVISIBLE);
        botonGuardarObjetivo.setVisibility(View.VISIBLE);
});

botonGuardarObjetivo.setOnClickListener((view) → {
        cajaObjetivo.setVisibility(View.INVISIBLE);
        textoObjetivo.setText(cajaObjetivo.getText().toString());
        botonEditarSaltos.setVisibility(View.VISIBLE);
        botonGuardarObjetivo.setVisibility(View.INVISIBLE);
        textoObjetivo.setVisibility(View.VISIBLE);
        objetivo=Integer.parseInt(textoObjetivo.getText().toString());
});
```

Se basa en actualizar variables y mostrar o no ciertos elementos de la interfaz.

La aplicación muestra información de cuánto falta para finalizar el ejercicio y cuando se llegue al objetivo establecido por el usuario el dispositivo vibrará para notificar el fin del entrenamiento.



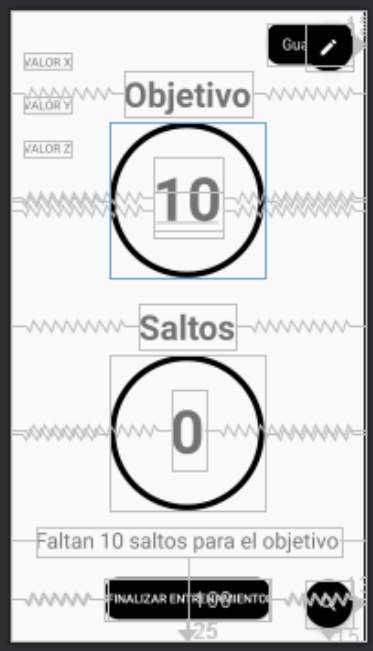


Cuando la aceleración sea superior a 10m/s^2 se actualizan los contadores y variables. Cuando se llega al objetivo se lanza el mensaje y el dispositivo vibra.

```
if(valorX-valorXanterior>10||valorX-valorXanterior<-10){
    contadorSaltos=contadorSaltos+1;
    textoSaltos.setText(""+contadorSaltos);
    aux=objetivo-contadorSaltos;
    if(aux>0) {
        textoFaltaObjetivo.setText("Faltan " + aux + " para el objetivo");
        textoFaltaObjetivo.setTextColor(Color.rgb( red: 244, green: 69, blue: 69));
    }else{
        textoFaltaObjetivo.setText("¡Entrenamiento finalizado!");
        textoFaltaObjetivo.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
        //vibramos para avisar
        Vibrator vibrator = (Vibrator)getApplicationContext().getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate( milliseconds: 1000);
    }
```

El layout de esta actividad se compone de *TextViews* donde se muestran los datos, de *EditText* donde el usuario puede modificarlos y de *Buttons* desde los cuales puede activar la edición, guardar datos o acceder al histórico:

```
<Button
    android:id="@+id/botonSaltar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="25dp"
    android:textColor="#FFFFFF"
    android:background="@drawable/botonredondeado"
    android:text="Comenzar Entrenamiento" />

<Button
    android:id="@+id/botonFinalizar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="25dp"
    android:textColor="#FFFFFF"
    android:background="@drawable/botonredondeado"
```



EntrenamientoCarrera

Para este tipo de entrenamiento se utiliza el sensor **GPS - Ubicación**. El objetivo es crear un entrenamiento registrando la distancia recorrida, el tiempo, y la velocidad a la que se ha realizado. El usuario puede acceder a entrenamientos, compararlos entre sí y ver en detalle las calorías consumidas en el mismo.

En primer lugar creo, si no existe, la tabla *datosEjercicio* para almacenar los registros del entrenamiento.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.setTitle("Comenzar Entrenamiento");

    //creacion de la base de datos
    myDB= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE,  factory: null);
    myDB.execSQL("CREATE TABLE IF NOT EXISTS datosEjercicio (metros FLOAT, velocidad FLOAT,tiempo FLOAT, id INTEGER PRIMARY KEY AUTOINCREMENT)");
```

A continuación se piden los permisos para acceder a la localización del dispositivo cuando el usuario interactúe con el botón *Obtener Ubicación:*

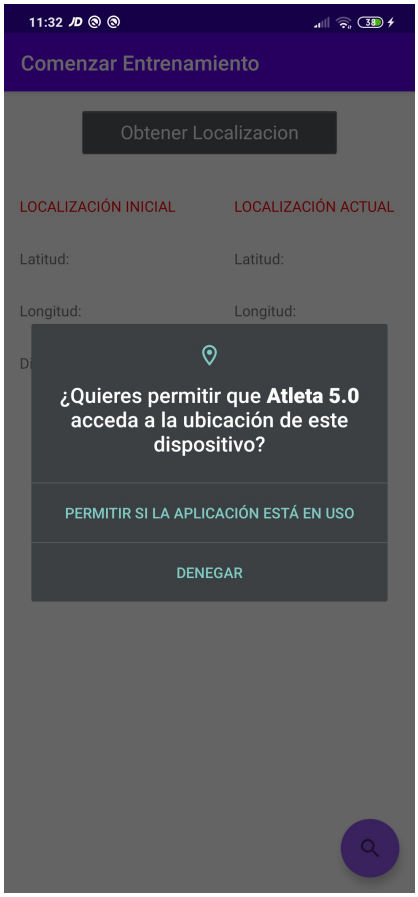
```
// Inicializar localizacion
fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient( activity: this);
boton.setOnClickListener((view) -> {
        //Primero se comprueban permisos
        if (ActivityCompat.checkSelfPermission( context: EntrenamientoCarrera.this, Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            getLocation();
            botonEntrenamiento.setVisibility(View.VISIBLE);
        } else {
            ActivityCompat.requestPermissions( activity: EntrenamientoCarrera.this, new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                    requestCode: 44);
        }

});
```

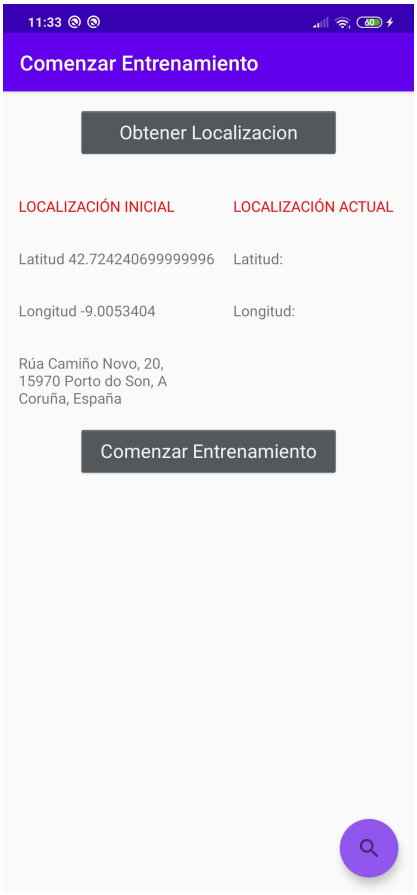
En caso de que el permiso no se haya concedido previamente se le pedirá al usuario su aprobación para acceder a dicho sensor. Si el permiso es concedido se activa el botón Comenzar Entrenamiento y se llama a la función getLocation() que nos mostrará los datos de la ubicación actual.

Se añade un listener a la instancia de la localización creada en la captura anterior. Utilizando la clase *Geocoder* accedemos a los datos de latitud, longitud y dirección y los mostramos en los *TextView* de la interfaz de usuario.

```
//obtener localizacion
private void getLocation() {
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageMa
        return;
    }
    fusedLocationProviderClient.getLastLocation().addOnCompleteListener((task) -> {
            //inicializar ubicacion

            Location location = task.getResult();
            if (location != null) {

                try {
                    //inicializar geocoder
                    Geocoder geocoder = new Geocoder( context: EntrenamientoCarrera.this, Locale.getDefault());
                    List<Address> addresses = geocoder.getFromLocation(location.getLatitude(),
                            location.getLongitude(),
                             maxResults: 1);
                    t1.setText("Latitud " + addresses.get(0).getLatitude());
                    t2.setText("Longitud " + addresses.get(0).getLongitude());
                    t3.setText("" + addresses.get(0).getAddressLine( index: 0));
                    //actualizo variables
                    latitudInicial = addresses.get(0).getLatitude();
                    longitudInicial = addresses.get(0).getLongitude();


                } catch (IOException e) {
                    e.printStackTrace();
                }
```

El resultado a nivel de interfaz sería el siguiente (primero se piden permisos):





El botón *Comenzar Entrenamiento* cuenta con un listener, que en caso de ser pulsado inicializará el cronómetro del ejercicio y obtendrá la localización del dispositivo llamando a la función *getLocation2()*:

```
//Boton comenzar entrenamiento
botonEntrenamiento.setOnClickListener((view) → {
        crono.setBase(SystemClock.elapsedRealtime());
        crono.start();
        getLocation2();
        botonFinalizar.setVisibility(View.VISIBLE);
        textoMetros.setVisibility(View.VISIBLE);
        textoVelocidad.setVisibility(View.VISIBLE);
        crono.setVisibility(View.VISIBLE);
});
```

Se obtiene la localización y se actualizan los valores mostrados en la pantalla cada vez que se detecte un cambio de ubicación para tener los datos actualizados en todo momento.

```
private void getLocation2() {
    LocationManager locationManager = (LocationManager) EntrenamientoCarrera.this.getSystemService(Context.LOCATION_SERVICE);
    LocationListener locationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            t4.setText("Latitud " + location.getLatitude());
            t5.setText("Longitud " + location.getLongitude());
            getDistancia(location.getLatitude(), location.getLongitude());
        }
```

También se invoca a la función *getDistancia()* que calcula y muestra en la pantalla los metros recorridos y la velocidad actual:

```
//Se obtiene los datos de distancia y velocidad
private void getDistancia(double latitud, double longitud) {
    float results[];

    results = new float[4];
    Location.distanceBetween(latitudInicial, longitudInicial, latitud, longitud, results);
    metros = results[0];
    t7.setText("" + results[0]);
    long milisegundos = SystemClock.elapsedRealtime() - crono.getBase();
    segundos = milisegundos / 1000;
    velocidad = results[0] / segundos;
    t6.setText("" + velocidad);

}
```

Cuando el usuario decide finalizar el entrenamiento y toca el boton *Finalizar Entrenamiento* se ejecuta el código dentro del listener asociado:

En primer lugar se detiene el cronómetro y se registra el tiempo para tenerlo guardado. Después se inserta en la base de datos, en la tabla *datosEjercicio* los valroes metros, velocidad y tiempo:

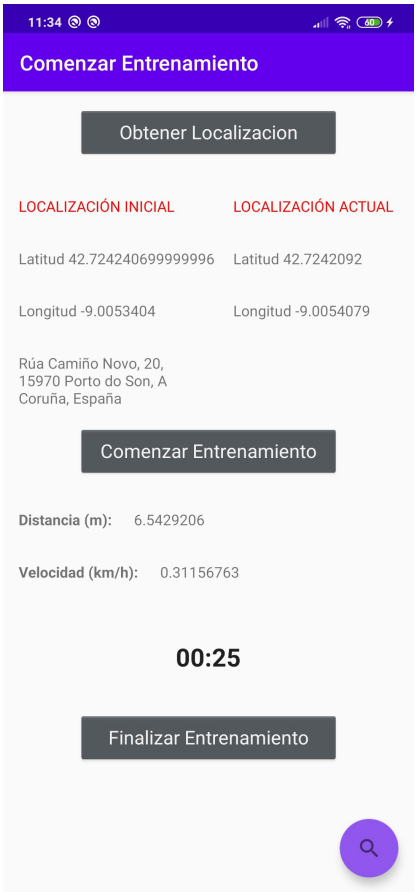
```
botonFinalizar.setOnClickListener((view) → {
        crono.stop();
        long milisegundos = SystemClock.elapsedRealtime() - crono.getBase();
        segundos = milisegundos / 1000;

        // Guardo datos en la BD
        ContentValues row1 = new ContentValues();
        float m = Float.parseFloat(t7.getText().toString());
        float v = Float.parseFloat(t6.getText().toString());
        float t = segundos;
        row1.put("metros", m);
        row1.put("velocidad", v);
        row1.put("tiempo", t);
        myDB.insert( table: "datosEjercicio",  nullColumnHack: null, row1);
        Toast.makeText(getApplicationContext(),  text: "Valores guardados", Toast.LENGTH_LONG).show();
        //
      obtenerHistorico();
});
```

Tras mostrar el mensaje de *Valores Guardados* se invoca a la función *obtenerHistorico()* que nos muestra una nueva pantalla con todos los valores de los entrenamientos previos:

```
private void obtenerHistorico(){
    //Recorro la base de datos y creo el array para mostrar los datos
    ArrayList<String> arrayDatos = new ArrayList<~>();
    ArrayList<Integer> arrayDatosID = new ArrayList<~>();
    EntrenamientoCarrera main = new EntrenamientoCarrera();
    SQLiteDatabase myDB = main.getBaseDatos();
    myDB= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE,  factory: null);
    Cursor myCursor = myDB.rawQuery( sql: "Select * from datosEjercicio",  selectionArgs: null);
    DecimalFormat formato1 = new DecimalFormat( pattern: "#.##");
    while (myCursor.moveToNext()) {
        String te = Float.toString(myCursor.getFloat( i: 2));
        arrayDatos.add(formato1.format(myCursor.getFloat( i: 0)) + " metros  ---  " + formato1.format( number: myCursor.getFloat( i: 1) * 3.6) +
                " km/h  ---  " + te + " segundos");
        arrayDatosID.add(myCursor.getInt( i: 3));
    }
    //llamo a la nueva ventana
    Intent intent = new Intent(getApplicationContext(), ListaEntrenamientos.class);
    intent.putExtra( name: "datos", arrayDatos);
    intent.putExtra( name: "idEntrenamiento", arrayDatosID);
    startActivity(intent);
}
```

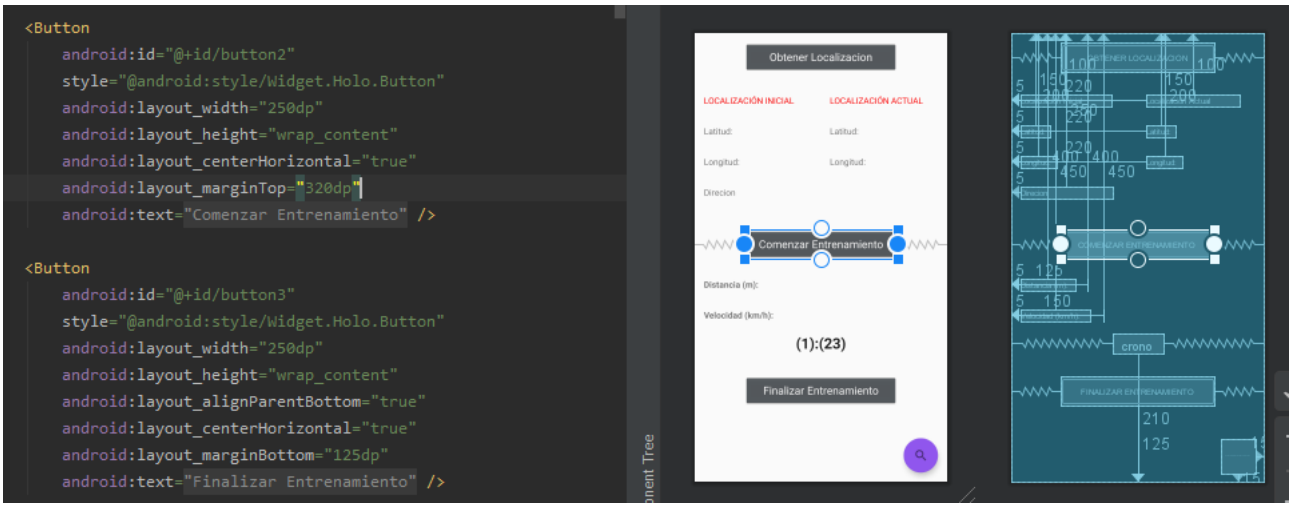
Creo dos *ArrayList* en los que incluyo los datos en formato texto y los ID asociados a cada entrenamiento. Recorriendo la tabla añado la información con el formato al array para después llamar a la nueva actividad pasándole los arrays con *.putExtra()*. De esta forma, la nueva actividad (ventana para el usuario) dispondrá del histórico de datos.

Las interfaces gráficas asociadas son las siguientes. A la izquierda podemos ver la pantalla de entrenamiento con los datos de las localizaciones y datos asociados al entrenamiento (distancia recorrida, velocidad y tiempo). Una vez finalizado el entrenamiento accedemos a la pantalla de la derecha (*ListaEntrenamientos*) donde observamos los datos de cada entrenamiento realizado:



En esta nueva pantalla tenemos dos funcionalidades disponibles: En la lupa inferior derecha accederemos a una gráfica en la que se muestran los datos de todos los entrenamientos para tener una comparativa más visual. Como segunda funcionalidad si se toca un entrenamiento de la lista se accederan a datos más concretos del entrenamiento. Estas dos funcionalidades se explicarán en los siguientes puntos.

El layout de esta actividad se compone de *TextViews* donde se muestran los datos y de *Buttons/FloatingActionButton* desde los cuales se comienza/finaliza el entrenamiento o accede al historial de los mismos.



Perfil

En este apartado el usuario dispone de sus datos personales. Podrá también acceder a un histórico del su peso para poder tener un control. Entre los datos tenemos su nombre, edad, altura e IMC (Índice de Masa Corporal) que nos indica su estado de forma.

En el código trabajamos con SQLite en su versión para Android. Este SGBD nos permite trabajar con bases de datos en local de una forma sencilla y eficaz:

```
//base de datos

myDB= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE,  factory: null);

myDB.execSQL(
        "CREATE TABLE IF NOT EXISTS historicoPeso (peso FLOAT, fecha DATE)"
);

this.setTitle("Mi Perfil");
SharedPreferences myPreferences = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
String nombre = myPreferences.getString( s: "Nombre",  s1: "");
String edad = myPreferences.getString( s: "Edad", s1: "");
String altura = myPreferences.getString( s: "Altura", s1: "");
String peso = myPreferences.getString( s: "Peso", s1: "");
String imc = myPreferences.getString( s: "IMC", s1: "");
```

Se crea, en caso de que no exista, la tabla *historicoPeso* que almacenará todos los pesos que el usuario registre. Para acceder a los datos de *nombre, edad* y *altura* utilizo otra forma de almacenar datos que nos ofrece Android: *SharedPreferences*. Se tratan de pares clave-valor que nos permiten acceder de forma rápida a datos del usuario.

Para calcular el IMC se llama a esta función que utiliza el peso y altura para determinar si se tiene un peso insuficiente, normal o sobrepeso. La información se muestra de forma muy visual en la pantalla.

```
private void calcularImc(String imc) {
    textoResultadoIMC.setVisibility(View.VISIBLE);
    if(Float.parseFloat(imc)<18.5){
        textoResultadoIMC.setText("Peso insuficiente");
        textoResultadoIMC.setTextColor(Color.rgb( red: 241, green: 203, blue: 30));

    }else if (Float.parseFloat(imc)>24.9){
        textoResultadoIMC.setText("Sobrepeso");
        textoResultadoIMC.setTextColor(Color.rgb( red: 244, green: 69, blue: 69));
    }else{
        textoResultadoIMC.setText("Peso normal");
        textoResultadoIMC.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
    }
}
```

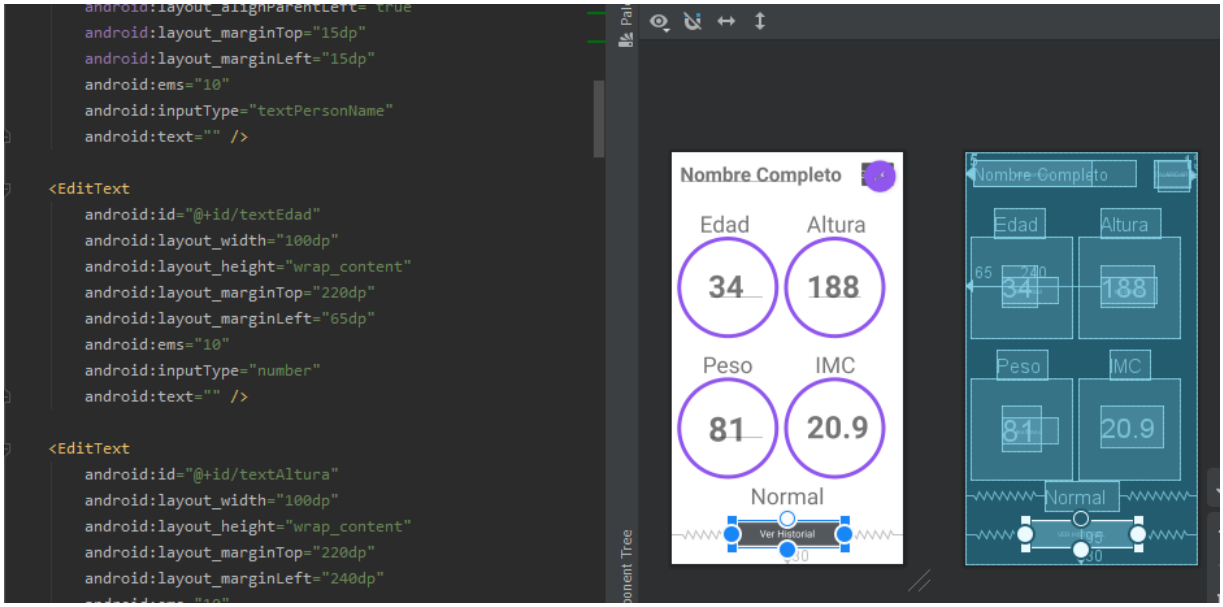
Cuando el usuario edita sus datos y le da a *Guardar* se introducen los datos en la tabla de la base de datos:

```
//guardar datos en la bd
ContentValues row = new ContentValues();
row.put("peso", Float.parseFloat(cajaPeso.getText().toString()));
myDB.insert( table: "historicoPeso",  nullColumnHack: null, row);

Toast.makeText(getApplicationContext(),  text: "Datos guardados", Toast.LENGTH_LONG).show();
```

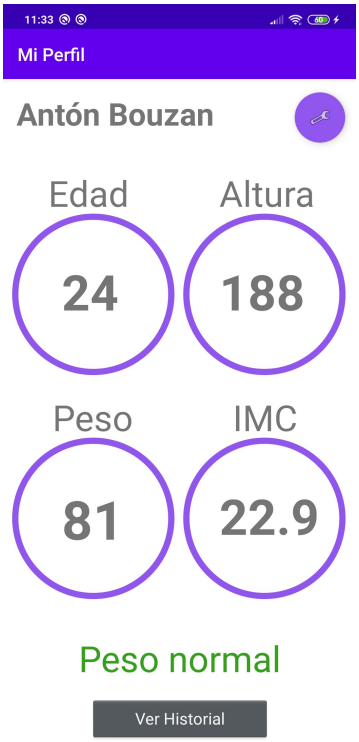
Si el usuario selecciona la opción *Ver Histórico* se invoca a la actividad *Grafica* que muestra una gráfica con el histórico de datos:

```
botonHistorico.setOnClickListener((view) → {
        Intent intent = new Intent(getApplicationContext(), Grafica.class);
        intent.putExtra( name: "opcion",  value: 2 );
        startActivity(intent);
});
```

El layout del perfil del usuario se compone de *TextViews* donde se muestran los datos, de *EditText* donde el usuario puede modificarlos y de *Buttons* desde los cuales puede activar la edición, guardar datos o acceder al histórico:



El resultado final de la actividad corriendo en el smartphone es el siguiente:



En la parte superior izquierda se muestra el nombre del usuario. En la superior derecha un botón que activa la edición de los campos. En el centro se muestran los datos personales introducidos por el usuario y el IMC con su mensaje generados por la aplicación. Por último el botón para acceder al historial:

ListaEntrenamientos

Esta actividad utiliza como elemento principal un *ListView* para mostrar los datos de los entrenamientos que han sido obtenidos y pasados desde la actividad anterior (*EntrenamientoCarrera*).

Para ello creo dos *ArrayList*, uno de String para obtener los datos en el formato que quiero mostrarlos y otro de Integers para obtener los ids asociados a los entrenamientos.

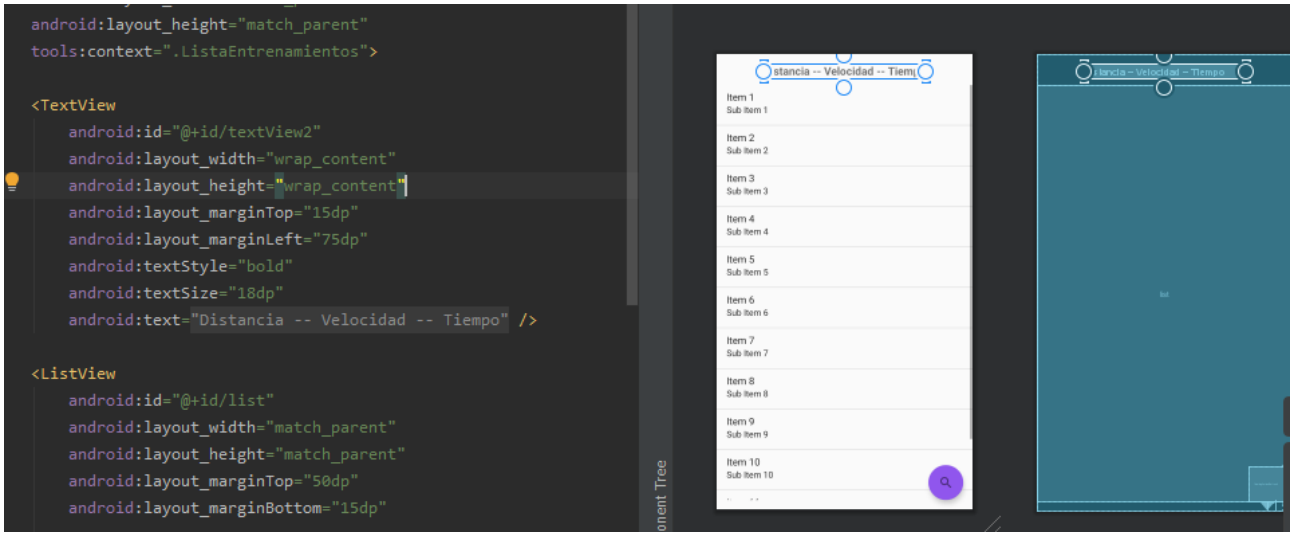
Establezco el adaptador del *ListView* pasándole la lista de String que acabo de crear:

```
//Obtengo los datos del activity anterior
lv = (ListView) findViewById(R.id.list);
datos=getIntent().getExtras();
idEntrenamiento= getIntent().getExtras();
final ArrayList<String> lista = (ArrayList<String>) datos.getSerializable( key: "datos");
final ArrayList<Integer> listaID = (ArrayList<Integer>) idEntrenamiento.getSerializable( key: "idEntrenamiento");
//Establezco el adaptador
adaptador=new ArrayAdapter( context: this,android.R.layout.simple_list_item_1,lista);
lv.setAdapter(adaptador);
lv.setOnItemClickListener((adapterView, view, i, l) -> {
        //llamo a la nueva ventana
        Intent intent = new Intent(getApplicationContext(), Entrenamiento.class);
        String aux = lista.get(i);
        intent.putExtra( name: "datos", aux);
        intent.putExtra( name: "idEntrenamiento", listaID.get(i));
        startActivity(intent);

});
//Llamo a la ventana con la grafica
floatingActionButton.setOnClickListener((view) -> {
        Intent intent = new Intent(getApplicationContext(), Grafica.class);
        intent.putExtra( name: "opcion",  value: 1 );
        startActivity(intent);
});
```

Cuando se toca un elemento de la lista se llama a una nueva actividad (*Entrenamiento)* y, al igual que antes, se pasan los datos y el id del entrenamiento.

Interactuando con la lupa se accede a la gráfica comparativa que se explicará posteriormente.

El layout de esta actividad se compone de un *TextView* que indica el título, un *ListView* donde se muestran los datos de los entrenamientos y un *FloatingActionButton* que nos lleva a una nueva actividad con una gráfica comparativa.



Entrenamiento

Cuando el usuario selecciona uno de los entrenamientos que se muestra en la lista de históricos se invoca a esta nueva actividad donde se verá en detalle los datos del mismo.

Esta actividad recibe los datos del entrenamiento (distancia,velocidad y tiempo) así como su ID asociado. La pantalla se compone principalmente de una gráfica de barras en la que se compara la distancia recorrida en el ejercicio seleccionado con respecto a la media recorrida.

```
//Obtener datos
datos=getIntent().getExtras();
idEntrenamiento=getIntent().getExtras();
int id = (int) idEntrenamiento.getSerializable( key: "idEntrenamiento");
cajaEntrenamiento.setText(datos.getString( key: "datos", defaultValue: "No hay datos"));
cajaEntrenamiento.setVisibility(View.INVISIBLE);

//grafica

grafico = (BarChart)findViewById(R.id.graficaIndividual);
BarDataSet linea = new BarDataSet(addValores(id), label: "Metros Recorridos VS Media Recorrida");
linea.setColor(Color.parseColor( colorString: "#9056ED"));
ArrayList<IBarDataSet> dataSets = new ArrayList<>();
dataSets.add(linea);
BarData linea2 = new BarData(dataSets);
grafico.setData(linea2);
grafico.invalidate();
```

Tras recoger los datos se crea la gráfica y se invoca a la función *addValores()* a la que pasamos como argumento el ID del ejercicio seleccionado. Esta función que explicaremos a continuación devuelve los valores de los datos de la tabla. A continuación se disponen los datos en la gráfica estableciendo colores y estilo.

La función *addValores()* recibe el identificador del ejercicio para poder acceder así a sus datos. Nos conectamos a la base de datos y realizamos una sentencia *sql* a través de la cuál obtenemos los datos del ejercicio guardándolos en las variables m (distancia),v (velocidad) y t (tiempo). Como la velocidad se registra en m/s se multiplica por 3,6 para obtener los km/h. Añadimos a *dataValue* una nueva barra con la distancia recordida. Por último se añaden a los *TextView* asociados parasu presentación en la interfaz.

```
private ArrayList<BarEntry> addValores(Integer idEn){
    ArrayList<BarEntry> dataValue = new ArrayList<~>();
    EntrenamientoCarrera main = new EntrenamientoCarrera();
    SQLiteDatabase mydb= main.getBaseDatos();
    mydb= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE,  factory: null);
    Cursor myCursor = mydb.rawQuery( sql: "Select * from datosEjercicio where id = "+idEn+" ;",  selectionArgs: null);
    while (myCursor.moveToNext()) {
        dataValue.add(new BarEntry( x: 0,myCursor.getFloat( i: 0)));
        m=myCursor.getFloat( i: 0);
        v=myCursor.getFloat( i: 1);
        t=myCursor.getFloat( i: 2);
        cajaMetros.setText(""+m);
        cajaVelocidad.setText(""+v*3.6);
        cajaTiempo.setText(""+t);
        met=1;
        v=v*3.6f;
        t=t/60;
```

Pasamos ahora quizá a los más interesante de la función. Y es que, **gracias a los datos obtenidos por el GPS** (tercera funcionalidad de sensor: velocidad, saltos, kcal) podemos determinar las kilocalorías que han sido consumidas en el ejercicio de carrera. Para ello he consultado webs de referencia en el mundo de la preparación física en la que indican la forma de realizar el cálculo. La fórmula para obtener el número de kilocalorías quemadas durante un ejercicio físico es la siguiente:

Kcal/min= MET x 0,0175 x peso(kg)

Por ejemplo en ejercicio de 60 minutos:

Kcal = MET x 0,0175 x peso(kg) x 60

El dato del peso del usuario lo tenemos disponible en su perfil, y guardado en la base de datos por lo que podemos acceder a su valor de forma rápida y sencilla.

Para saber el valor MET nos valemos de la siguiente tabla:

Actividad	METs
Pasear - 4,5km/h	2
Caminar ligero – 5,3 km/h	3,8
Caminar rápido – 6,4 km/h	5
Correr - 8,4 km/h	9
Correr - 9,6 km/h	10
Correr - 10,8 km/h	11
Correr > 10,8 km/h	14

Al disponer de la velocidad del ejercicio en la base de datos podemos fácilmente trabajar con los valores de la tabla y así determinar el número de kilocalorías consumidas de acuerdo con la fórmula:

```
//calcular las kcal gracias al valor de MET
if(v>0 && v<4.5){ //Pasear
    met=2;
}else if(v>=4.5 && v<6.4){ //caminar ligero
    met =3.3f;
}else if(v>=6.4 && v<8.4){ //caminar rapido
    met=5;
}else if(v>=8.4 && v<9.6){ //correr 1
    met=9;
}else if(v>=9.6 && v<10.8){ //correr 2
    met=10;
}else if(v>=10.8 && v<11.3){ //caminar rapido
    met=11;
}else{
    met = 14;
}
kcal = met*0.0175f*peso*t;
```

De esta forma tenemos guardado en la variable kcal el valor de:

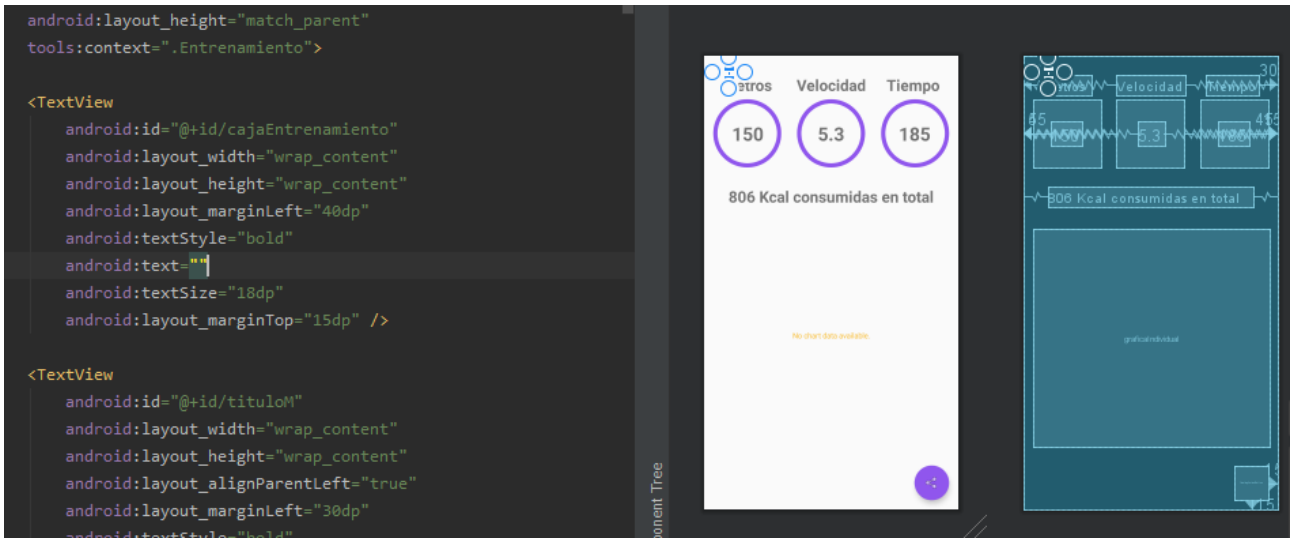
$$kcal = met*0,0175*peso*t$$

Añadimos al *TextView* el número de kcal consumidas durante el ejercicio y le añadimos un color verde para que destaque. A continuación obtenemos los datos de todos los ejercicios para obtener el total de metros recorridos y así obtener la media y añadimos de nuevo a *dataValue* otra barra con el dato de la media de distancia. Finalizamos la función devolviendo la variable *dataValue.*

```
        cajaCalorias.setText(""+kcal+" Kcal consumidas");
        cajaCalorias.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
    }
    myCursor = mydb.rawQuery( sql: "Select * from datosEjercicio ;", selectionArgs: null);
    float totalMetros=0;
    int contador=0;
    while (myCursor.moveToNext()) {
        contador++;
        totalMetros=myCursor.getFloat( i: 0)+totalMetros;
    }
    float mediaMetros = totalMetros/contador;
    dataValue.add(new BarEntry( x: 1,mediaMetros));
    return dataValue;
}
```

Este es el aspecto de la interfaz gráfica que ve el usuario. En los círculos superiores se representan los valores de distancia velocidad y tiempo de entrenamiento. Debajo la cantidad de Kcal consumidas (116,16 en este caso) y debajo una gráfica comparativa en la que vemos a la izquierda la distancia recorrida en el entrenamiento actual, y a la derecha la media de todo nuestro historial. Con el botón situado en la esquina inferior derecha el usuari podrá compartir los datos de su entrenamiento con aplicaciones externas.

El layout de esta actividad se compone de cuatro *TextView* en los que se muestra la información del ejercicio y de un elemento de la librería *AndroidChart* para crear la gráfica de barras comparativa. También cuenta con un elemento *FloatingActionButton* para compartir los datos del entrenamiento con aplicaciones externas.



Grafica

Esta actividad es invocada cuando se quieren comparar datos históricos de entrenamientos de carrera, de salto o de peso. Recibe un int con un número de opción para saber en qué tabla de la base de datos consultar:

```
//Obtengo datos de la actividad previa
datos=getIntent().getExtras();
opcion= (int) datos.getSerializable( key: "opcion");
//obtener datos de la bd
ArrayList<Float> datos = new ArrayList<>();
if (opcion==3){
    sql = "Select * from historicoSaltos";
    titulo ="Número de saltos";
    this.setTitle("Entrenamiento de saltos");
}else if (opcion==1) {
    sql = "Select * from datosEjercicio";
    titulo="Metros recorridos";
    this.setTitle("Gráfica de entrenamientos");
}else{
    sql = "Select * from historicoPeso";
    titulo="Peso historico";
    this.setTitle("Histórico de Peso");
}
datos = obtenerDatos();
```

Dependiendo de la opción se crea la sentencia sql asociada a la tabla adecuada y después se invoca a la función *obtenerDatos():*

```
private ArrayList<Float> obtenerDatos(){

    ArrayList<Float> datos = new ArrayList<>();
    EntrenamientoCarrera main = new EntrenamientoCarrera();
    SQLiteDatabase myDB = main.getBaseDatos();
    myDB= openOrCreateDatabase( name: "datos.db", MODE_PRIVATE, factory: null);
    Cursor myCursor = myCursor = myDB.rawQuery(sql, selectionArgs: null);

    while (myCursor.moveToNext()) {
        datos.add(myCursor.getFloat( i: 0));
    }
    return datos;
}
```

Se devuelve la variable *datos* que contiene los datos consultados a la base de datos.

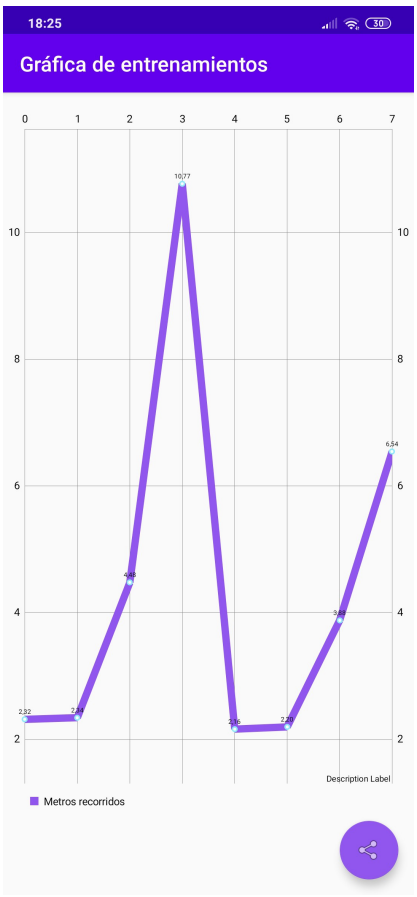
Ahora se crea la instancia de la gráfica de líneas y se pasa como argumento a la función *addValoresLinea* el ArrayList de floats que acabamos de obtener con la consulta anterior.

```
// Grafica
grafico2 = (LineChart)findViewById(R.id.grafica);
LineDataSet linea = new LineDataSet(addValoresLinea(datos),titulo);
linea.setColor(Color.parseColor( colorString: "#9056ED"));
linea.setLineWidth(7);
ArrayList<ILineDataSet> dataSets = new ArrayList<>();
dataSets.add(linea);
LineData linea2 = new LineData(dataSets);
grafico2.setData(linea2);
grafico2.invalidate();
```

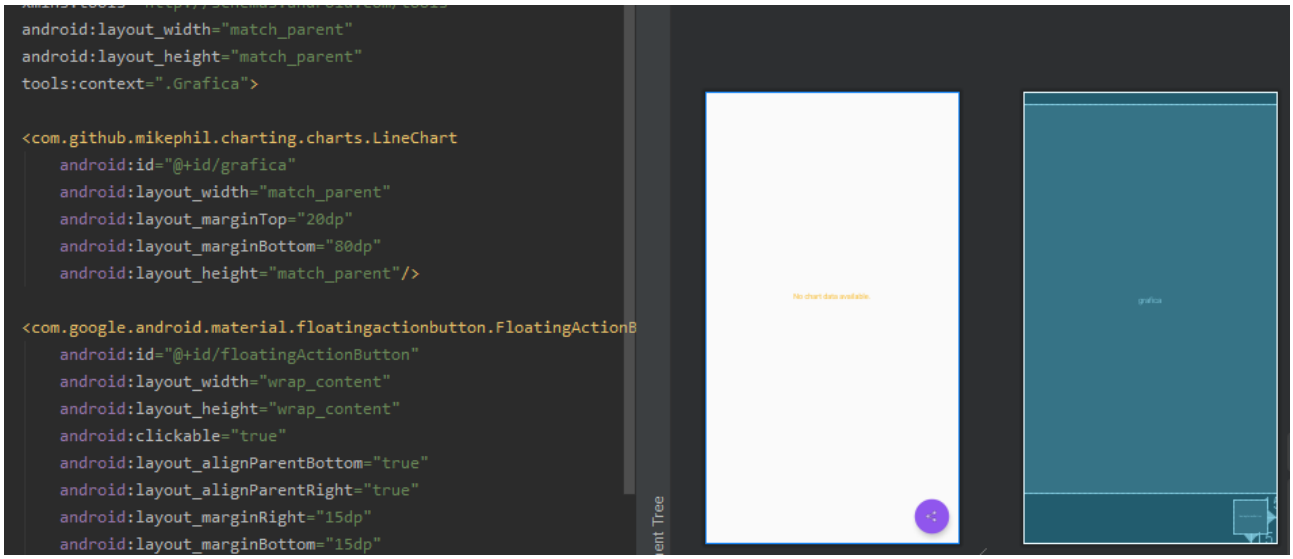
La función *addValoresLinea()* devuelve el tipo de dato necesario para que los datos puedan ser representados en la gráfica:

```
private ArrayList<Entry> addValoresLinea(ArrayList<Float> datos){
    ArrayList<Entry> dataValue = new ArrayList<~>();
    for(int i=0;i<datos.size();i++){
        dataValue.add(new BarEntry(i,datos.get(i)));
    }
    return dataValue;
}
```

Este es el aspecto de la interfaz gráfica que ve el usuario. La gráfica representa la evolución de la distancia recorrida en cada uno de los entrenamientos registrados. Con el botón situado en la esquina inferior derecha el usuari podrá compartir los datos de su entrenamiento con aplicaciones externas.



El layout de esta actividad se compone de un elemento de la librería *AndroidChart* para crear la gráfica de líneas comparativa. También cuenta con un elemento *FloatingActionButton* para compartir los datos del entrenamiento con aplicaciones externas.



PlanesEntrenamiento

Esta actividad se corresponde con la opción en el menú principal que nos ofrece planes de entrenamiento alternativos a la carrera y el salto. Cuenta con dos botones principales a través de los cuales se carga la tabla correspondiente con un entrenamiento para tren superior o inferior respectivamente.

```
botonSuperior.setOnClickListener((view) → {
        imagenInferior.setVisibility(View.INVISIBLE);
        imagenSuperior.setVisibility(View.VISIBLE);
        botonComenzar.setVisibility(View.VISIBLE);
        botonPersonalizar.setVisibility(View.VISIBLE);
        tipo=0;
});

botonInferior.setOnClickListener((view) → {
        imagenSuperior.setVisibility(View.INVISIBLE);
        imagenInferior.setVisibility(View.VISIBLE);
        botonComenzar.setVisibility(View.VISIBLE);
        botonPersonalizar.setVisibility(View.VISIBLE);
        tipo=1;
});
```

Cuando el usuario presiona el boton *Comenzar Entrenamiento* se llama a la neuva actividad pasando por defecto un entrenamiento de 5 series con una duración de 30 segundos el ejercicio:

```
botonComenzar.setOnClickListener((view) → {
        Intent intent = new Intent(getApplicationContext(), EntrenamientoTren.class);
        intent.putExtra( name: "tipo", tipo );
        intent.putExtra( name: "numSeries",  value: 5);
        intent.putExtra( name: "tiempo",  value: 30 );
        startActivity(intent);
});
```

El usuario tiene la posibilidad de personalizar el entrenamiento eligiendo el número de series y la duración de cada ejercicio haciendo uso del botón *Personalizar*. Se habilitará la opción de edición de los campos para que el usuario pueda introducir los valores que desee:

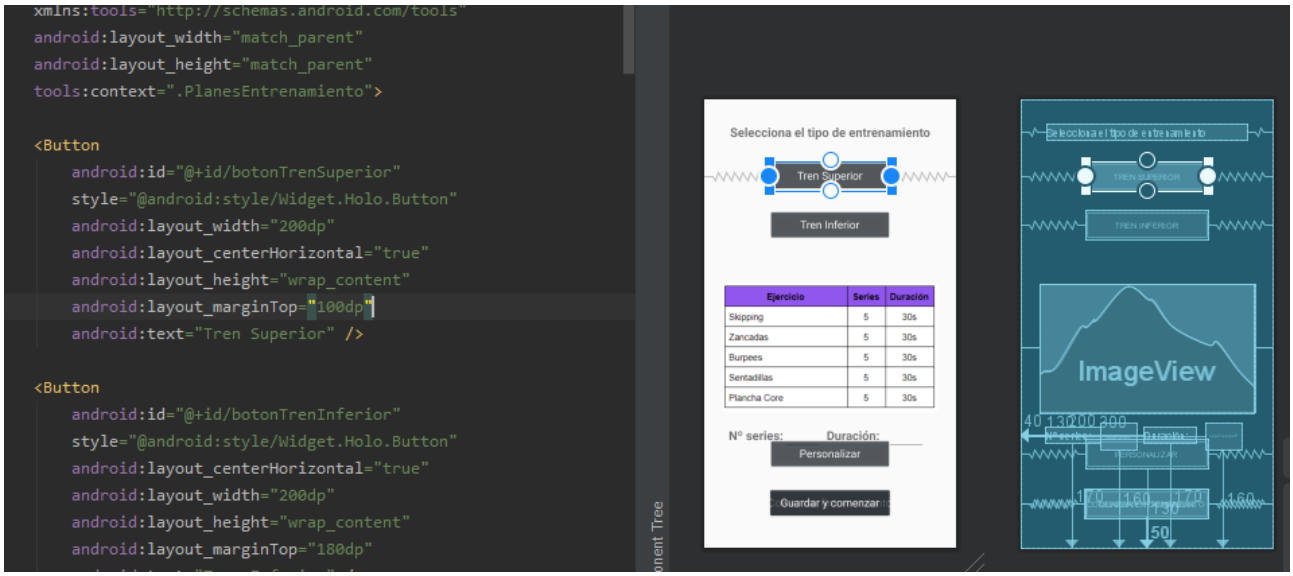
```
botonPersonalizar.setOnClickListener((view) → {
        botonComenzar.setVisibility(View.INVISIBLE);
        botonGuardar.setVisibility(View.VISIBLE);
        botonPersonalizar.setVisibility(View.INVISIBLE);

        textoSerie.setVisibility(View.VISIBLE);
        cajaSerie.setVisibility(View.VISIBLE);
        textoTiempo.setVisibility(View.VISIBLE);
        cajaTiempo.setVisibility(View.VISIBLE);
});
```

Tras personalizar el entrenamiento el usuario presionará el botón *Guardar y comenzar*, se recogeran los valores de los campos y se invocará una nueva actividad pasándole los datos *numSeries* y *tiempo* que guardan los valores introducidos por el usuario:

```
botonGuardar.setOnClickListener((view) → {
        numSeries = Integer.parseInt(cajaSerie.getText().toString());
        tiempo = Integer.parseInt(cajaTiempo.getText().toString());
        Intent intent = new Intent(getApplicationContext(), EntrenamientoTren.class);
        intent.putExtra( name: "tipo", tipo );
        intent.putExtra( name: "numSeries", numSeries);
        intent.putExtra( name: "tiempo", tiempo );
        startActivity(intent);
});
```

Este es el aspecto de la interfaz gráfica que ve el usuario. Los botones superiores muestran la tabla del entrenamiento mientras que los inferiores nos dan la opción de la personalización del mismo y de comenzar.

El layout de esta actividad se compone de elementos *Button* e *ImageView*. Dos botones principales que cargan la tabla del entrenamiento adecuado y otros dos botones: *Personalizar* y *Comenzar Entrenamiento*.



EntrenamientoTren

Tras haber elegido el tipo de entrenamiento y el número de series y tiempo de duración se invoca a esta actividad. En primer lugar se recogen y guardan en variables los datos necesarios previamente detallados:

```
//cojo datos
datos=getIntent().getExtras();
tipo= (int) datos.getSerializable( key: "tipo");
seriesTotales = (int) datos.getSerializable( key: "numSeries");
tiempoTotal = (int) datos.getSerializable( key: "tiempo");
numeroSerie=1;
numeroEjercicio=1;
tiempo.setText(""+tiempoTotal+"s");
```

Dependiendo del tipo de ejercicio seleccionado (*Tren Superior/Tren Inferior*) se muestra el título y tabla correspondiente (por defecto está establecido *Tren Superior*). Se añade un listener al botón *Comenzar Entrenamiento* que invoca a la función *empezarContador()*:

```
if(tipo==0) {
    this.setTitle("Tren Superior");
    tituloEjercicio.setText("Flexiones");
}else {
    this.setTitle("Tren Inferior");
    tituloEjercicio.setText("Skipping");
    imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagenskipping));
}

botonComenzar.setOnClickListener((view) -> {
        botonComenzar.setVisibility(View.INVISIBLE);
        botonSiguiente.setVisibility(View.VISIBLE);
        empezarContador();
});
```

Esta función crea una instancia de un objeto *CountDownTimer* y comienza la cuenta atrás del timepo introducido por el usuario. Cuando el tiempo finaliza *(onFinish)* se muestra un mensaje en color verde y el dispositivo vibra para notificar el fin del ejercicio sin que tenga que mirarse la pantalla:

```
public void empezarContador(){

    contador=new CountDownTimer( millisInFuture: tiempoTotal*1000,  countDownInterval: 1000) {
        public void onTick(long millisUntilFinished) {
            tiempo.setText("" + millisUntilFinished / 1000 + "s");
        }
        public void onFinish() {
            tiempo.setText("Hecho!");
            tiempo.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
            //vibramos para avisar
            Vibrator vibrator = (Vibrator)getApplicationContext().getSystemService(Context.VIBRATOR_SERVICE);
            vibrator.vibrate( milliseconds: 1000);
        }
    };
    contador.start();

}
```

Cuando el entrenamiento comienza y el temporizador empieza a contar se activa el botón *Siguiente* que permite al usuario avanzar al siguiente ejercicio. Cuando este botón es presionado, en primer lugar se comprueba si ya existe una instancia del temporizador y en caso de que así sea se cancela y comienza uno nuevo:

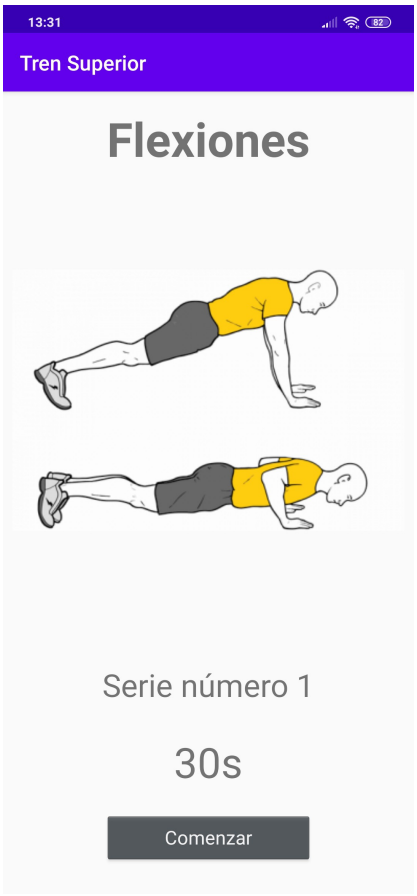
```
botonSiguiente.setOnClickListener((view) -> {
        if (contador != null) {
            contador.cancel();
            empezarContador();
        }
```

Después comprobamos que tipo de entrenamiento ha seleccionado y vamos avanzando en el número de ejercicio gracias a las variables que contienen el número actual de serie, de ejercicio y el objetivo:

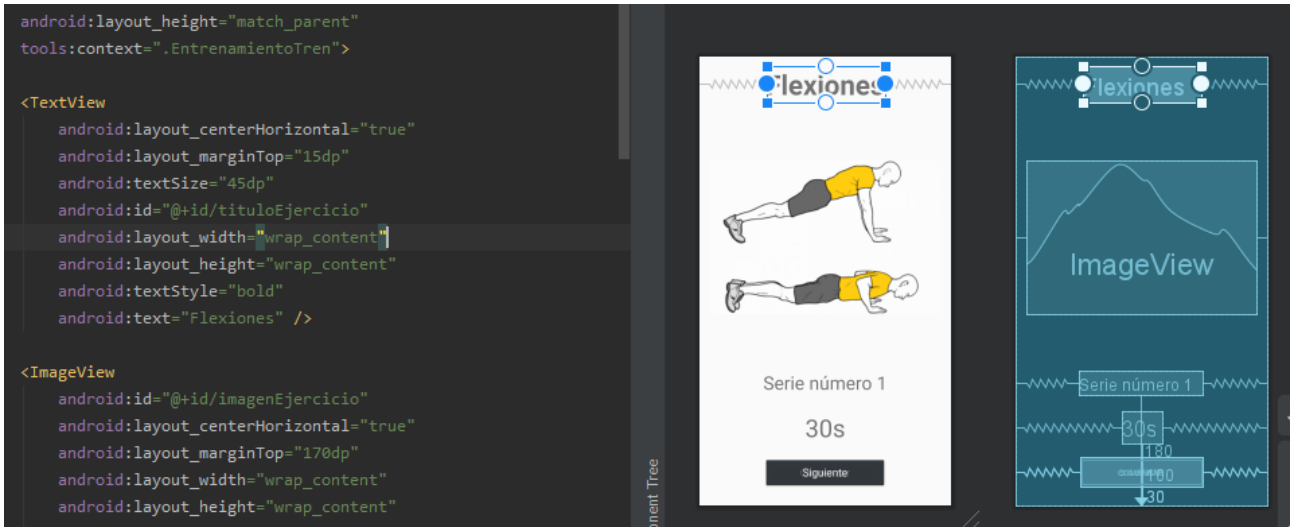
```
if (tipo == 0) {
    if(numeroSerie<=seriesTotales) {
        textoSerie.setText("Serie número "+numeroSerie);
        if (numeroEjercicio==10){
            tituloEjercicio.setText("Flexiones");
            imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagenflexiones));
            numeroEjercicio=1;
        }
        else if (numeroEjercicio == 1) {
            tituloEjercicio.setText("Tríceps");
            imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagentriceps));
            numeroEjercicio++;
        } else if (numeroEjercicio == 2) {
            tituloEjercicio.setText("Bíceps");
            imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagenbiceps));
            numeroEjercicio++;
        } else if (numeroEjercicio == 3) {
            tituloEjercicio.setText("Remo");
            imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagenremo));
            numeroEjercicio++;
        } else if (numeroEjercicio == 4) {
            tituloEjercicio.setText("Flexiones juntas");
            imagenEjercicio.setImageDrawable(getResources().getDrawable(R.drawable.imagenflexiones2));
            numeroEjercicio=10;
            numeroSerie++;
        }
    }else{
        textoSerie.setText("¡Entrenamiento finalizado!");
        textoSerie.setTextColor(Color.rgb( red: 56, green: 158, blue: 31));
```

En esta función se carga la imagen que ilustra el ejercicio correspondiente así como los títulos del mismo. Cuando se llega al final se muestra el mensaje *"¡Entrenamiento finalizado!"* en color verde.

Este es el aspecto de la interfaz gráfica que ve el usuario. En la parte superior se muestra el título del ejercicio actual, debajo la imagen representativa del mismo. El número de serie actual está indicado encima del temporizador. Por último el botón nos permite comenzar y avanzar en el entrenamiento.



El layout de esta actividad se compone de tres elementos *TextView* que muestran el título, número de serie y temporizador; de un elemento *ImageView* que muestra la imagen del ejercicio y de un elemento *Button* que nos permite avanzar en el entrenamiento:

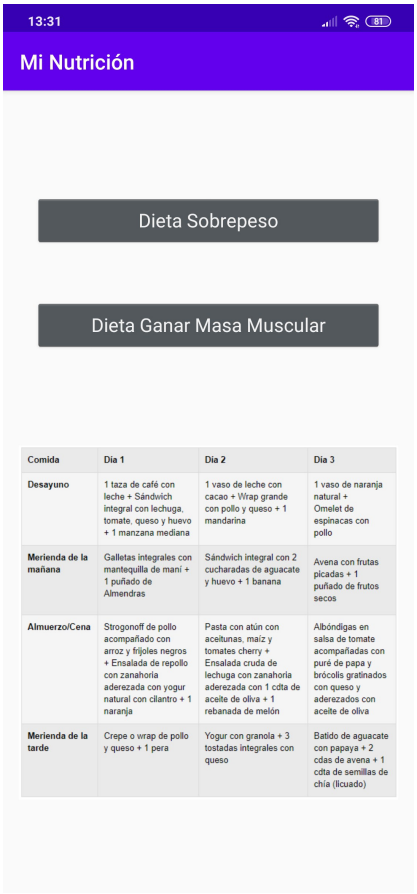


Nutrición

Esta actividad se corresponde con la opción en el menú principal que nos ofrece planes de nutrición. Cuenta con dos botones principales a través de los cuales se carga la tablas con la información nutricional correspondiente.

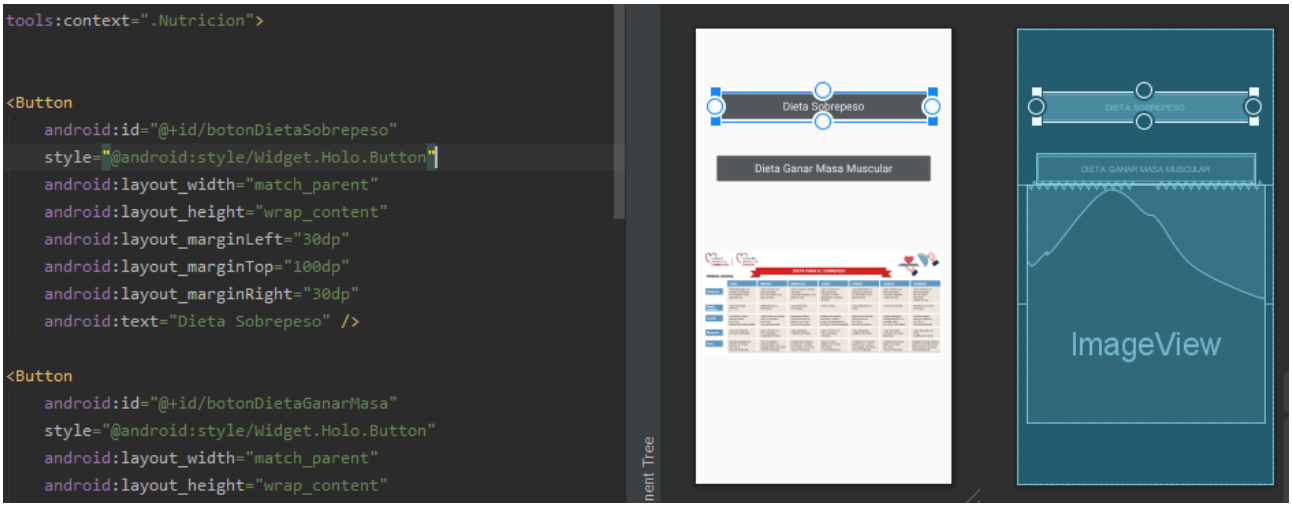
```
botonDieataMasa.setOnClickListener((view) → {
        imagenDietaSobrepeso.setVisibility(View.INVISIBLE);
        imagenDietaMasa.setVisibility(View.VISIBLE);
});

botonDietaSobrepeso.setOnClickListener((view) → {
        imagenDietaMasa.setVisibility(View.INVISIBLE);
        imagenDietaSobrepeso.setVisibility(View.VISIBLE);
});
```

Este es el aspecto de la interfaz gráfica que ve el usuario. A través de los botones se muestra la tabla de dieta asociada a las opciones:



Comida	Dia 1	Dia 2	Dia 3
Desayuno	1 taza de café con leche + Sándwich integral con lechuga, tomate, queso y huevo + 1 manzana mediana	1 vaso de leche con cacao + Wrap grande con pollo y queso + 1 mandarina	1 vaso de naranja natural + Omelet de espinacas con pollo
Merienda de la mañana	Galletas integrales con mantequilla de mani + 1 puñado de Almendras	Sándwich integral con 2 cucharadas de aguacate y huevo + 1 banana	Avena con frutas picadas + 1 puñado de frutos secos
Almuerzo/Cena	Strogonoff de pollo acompañado con arroz y frijoles negros + Ensalada de repollo con zanahoria aderezada con yogur natural con cilantro + 1 naranja	Pasta con atún con aceitunas, maiz y tomates cherry + Ensalada cruda de lechuga con zanahoria aderezada con 1 cdta de aceite de oliva + 1 rebanada de melón	Albóndigas en salsa de tomate acompañadas con puré de papa y brócolis gratinados con queso y aderezados con aceite de oliva
Merienda de la tarde	Crepe o wrap de pollo y queso + 1 pera	Yogur con granola + 3 tostadas integrales con queso	Batido de aguacate con papaya + 2 cdas de avena + 1 cdta de semillas de chía (licuado)

El layout de esta actividad se compone de dos elementos *Button* que nos permiten seleccionar el tipo de dieta y de un elemento *ImageView* que nos muestra la tabla con la información nutricional:



Chat

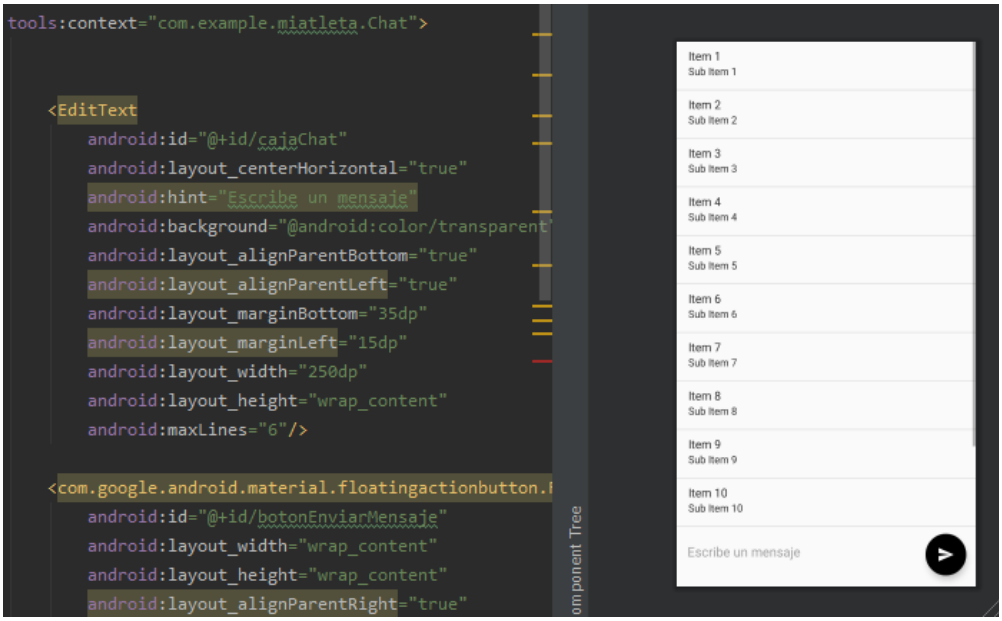
El apartado del chat utiliza json para compartir los mensajes entre el cliente y el servidor. EL usuario envía el mensaje que se inserta en la base de datos y a través de la siguiente función se realiza una consulta a la base de datos en la que se obtienen los mensajes en los que interviene el usuario identificado en la sesión:

```
//recorre el array
for (int i = 0; i < jsonArray.length(); i++) {

    //obtiene el elemento json del array
    JSONObject obj = jsonArray.getJSONObject(i);
    //condicional para mostrar solo mensajes en los que haya intervenido
    if (obj.getString( name: "id1").equals(String.valueOf(MainActivity.idUsuario))
            && obj.getString( name: "id2").equals(String.valueOf(aux2)) || obj.getString( name: "id1").equals(String.valueOf(aux2))
            && obj.getString( name: "id2").equals(String.valueOf(MainActivity.idUsuario))){
        heroes.add(obj.getString( name: "mensaje"));
    }
}
```

Con la función de *actualizarMensajes()* se refresca la lista de la interfaz en la cuál se muestran los mensajes:

```
Runnable actualizarMensajes = () -> {
        // Esto se ejecuta en segundo plano una única vez
        while (true) {
            try {//actualizamos cada 3 segundos
                Thread.sleep( millis: 3000);
                leerMensajes( urlWebService: MainActivity.ipGlobal+"/conexionmiatleta/leerMensajes.php");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
};
```

Este es el aspecto de la interfaz donde se muestra una caja de texto , un botón para el envío y una lista donde se muestran los mensajes:

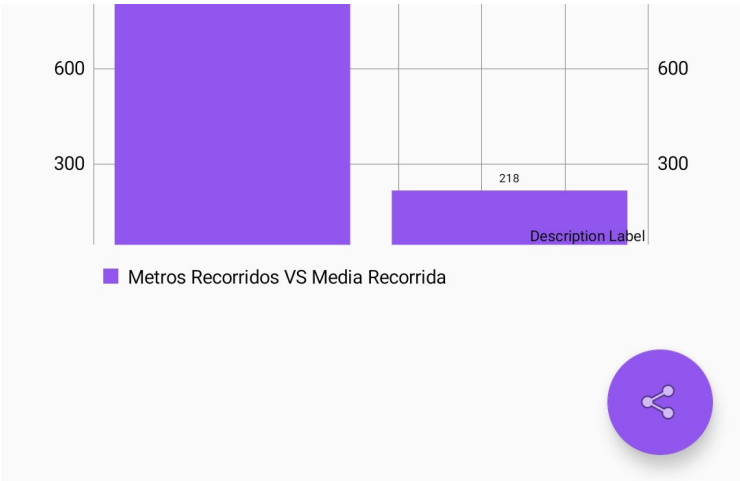


Perfil Social

Apartado dedicado a conectar a los diferentes usuarios de la plataforma:

Compartir información con aplicaciones externas

El requisito *R_13* nos indica que la aplicación debe permitir compartir información con aplicaciones externas. Para ello en varias actividades existe un botón flotante en la esquina inferior izquierda que nos permite hacerlo:



Cuando el botón es presionado se ejecuta el siguiente código. Se crea un mensaje con los datos del entrenamiento y la actividad de compartir información se activa:

```
//boton compartir datos
botonExportar.setOnClickListener((view) → {
        try {
        Intent i = new Intent(Intent.ACTION_SEND);
        i.setType("text/plain");
        i.putExtra(Intent.EXTRA_SUBJECT, "Atleta 5.0");
        String aux = "Te paso mi entrenamiento!\n";
        aux = aux + datos.getString( key: "datos", defaultValue: "No hay datos");
        i.putExtra(Intent.EXTRA_TEXT, aux);
        startActivity(i);
    } catch (Exception e) {
    }
});
```

Esta es la interfaz que el usuario verá cuando presione el botón flotante:

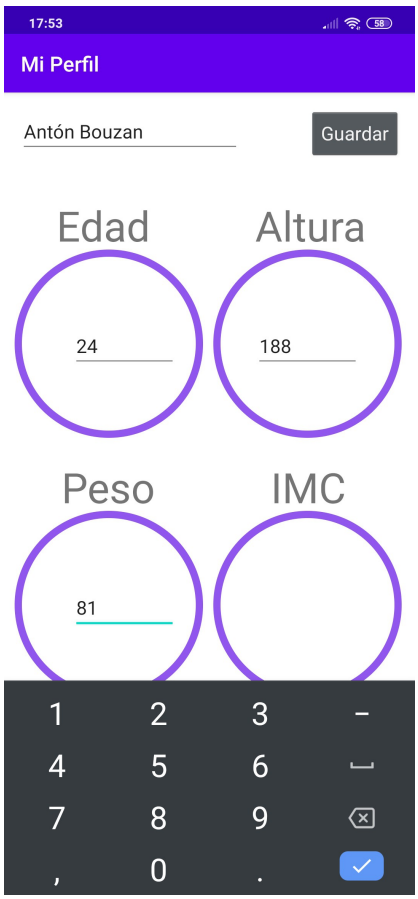
Deberá elegir con que aplicación quiere compartir los datos del entrenamiento *(WhatsApp,Twitter,Instagram* etc):

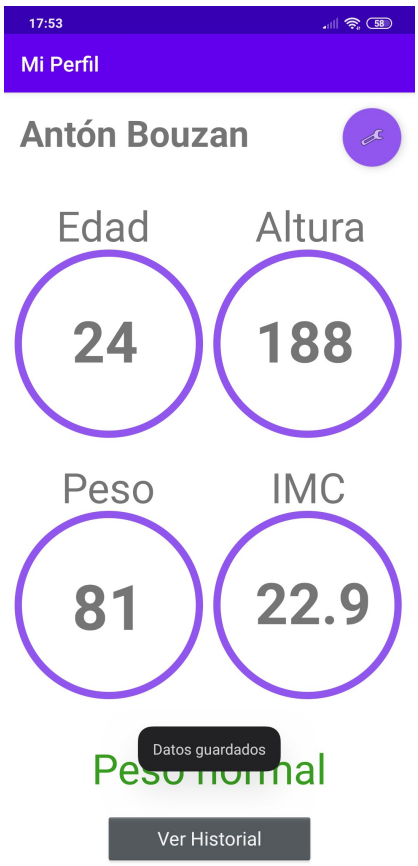
5.5 Descripción de los casos de prueba

Caso de prueba 1

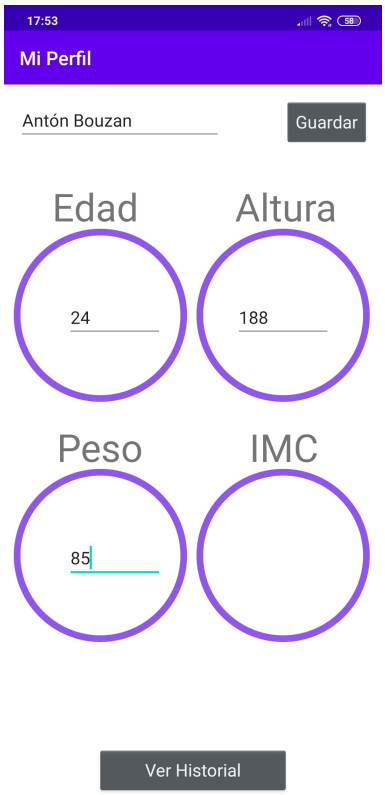
En el primer caso de prueba completaremos los datos de nuestro perfil y después modificaremos el peso varias veces para comprobar que el registro histórico es correcto.

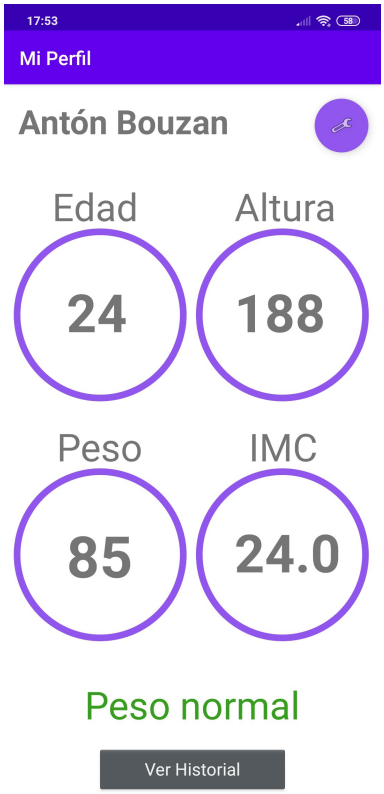
En primer lugar rellenamos los datos con nuestras medidas y nombre (imagen izquierda). Tras presionar el botón *Guardar* los valores se actualizarán y se mostrará el IMC añadiendo un mensaje que nos indica su nivel (imagen derecha):



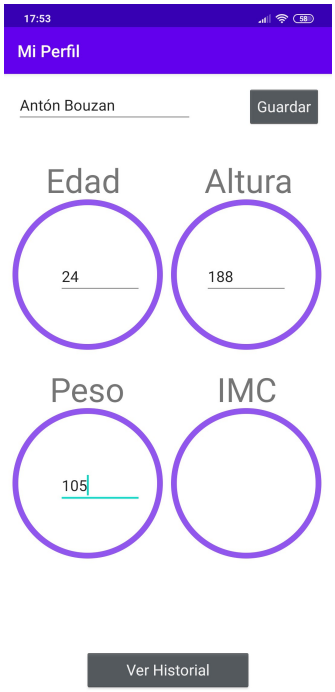


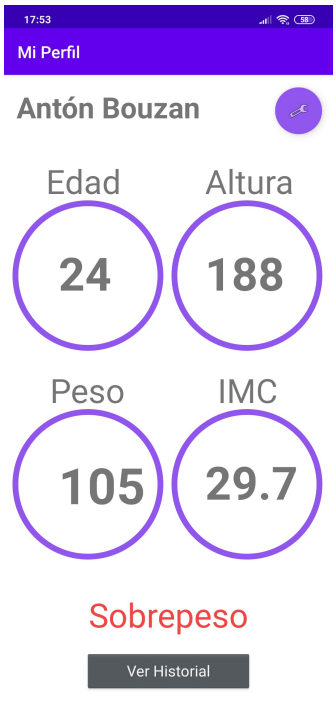
Pasamos ahora a actualizar nuestro peso, el resto de valores se quedan guardados (imagen izquierda). De igual forma que antes se actualizarán los datos y se nos mostrará el nuevo IMC de acuerdo con nuestro peso y altura (imagen derecha):



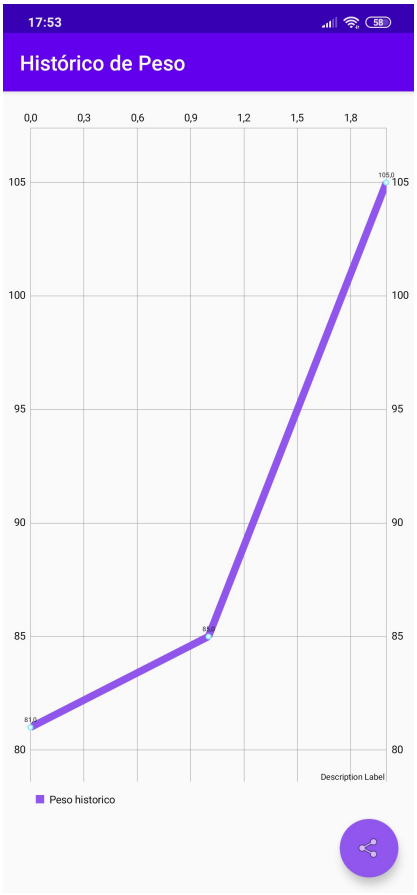


Actualizamos de nuevo nuestro peso (imagen izquierda). De igual forma que antes se actualizarán los datos y se nos mostrará el nuevo IMC de acuerdo con nuestro peso y altura (imagen derecha), esta vez mostrando sobrepeso:





Utilizando el botón *Histórico* accedemos a la gráfica comparativa de todos los pesos que registramos en la aplicación. De este modo tenemos un control de nuestra evolución:

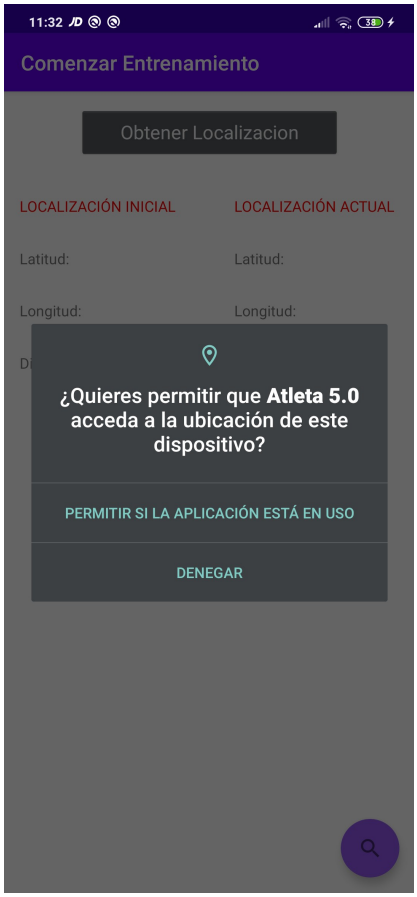


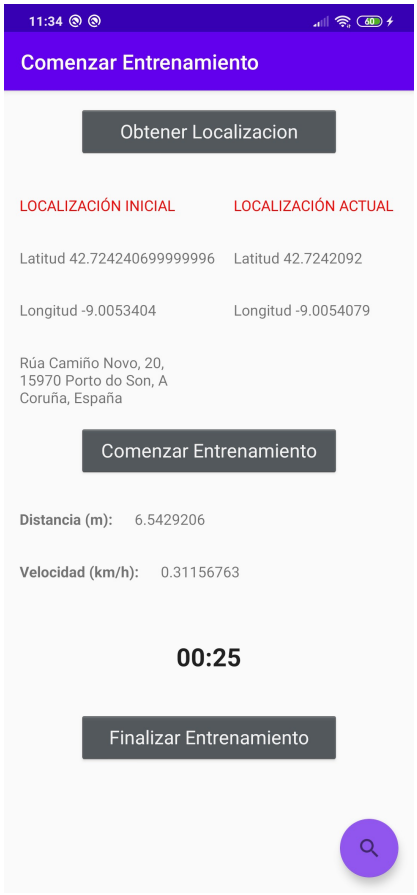
Con el botón flotante en la parte inferior derecha podemos compartir nuestra evolución con aplicaciones externas.

Caso de prueba 2

En el segundo caso de prueba realizaremos un entrenamiento de carrera para después comprobar cuántas Kcalorías hemos quemado y compartiremos nuestro entrenamiento por *WhatsApp.*

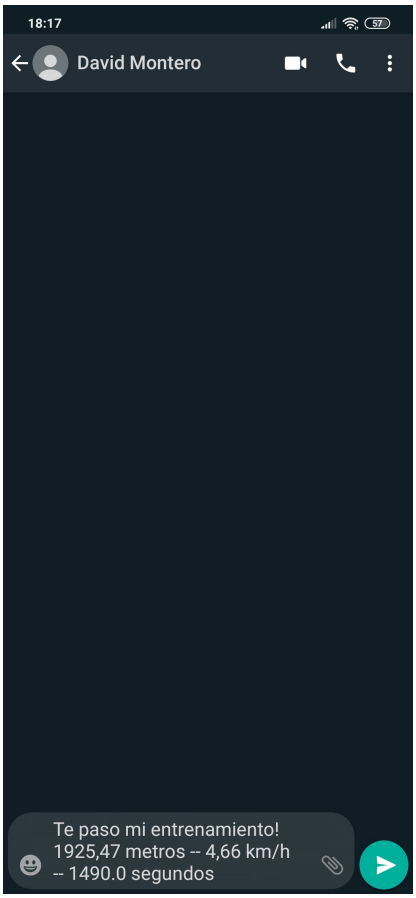
Seleccionamos en el menú principal la opción *Entrenamiento de carrera.* Con el botón superior *Obtener Localización* obtendremos nuestra localización inicial. Se habilitará el botón *Comenzar Entrenamiento* y se mostrarán los datos de ubicación actual, distancia recorrida y velocidad. Cuando el usuario pulse *Finalizar Entrenamiento*, se registrarán automáticamente los datos del mismo en la base de datos y se nos mostrará una nueva ventana (imagen derecha).





En la nueva ventana se nos ofrecen los datos de una forma más visual además de incluirse la cantidad de Kcal consumidas durante el entrenamiento. En la gráfica de barras podemos ver una comparativa entre el entrenamiento actual (barra izquierda) y la media del resto de entrenamientos (barra derecha).

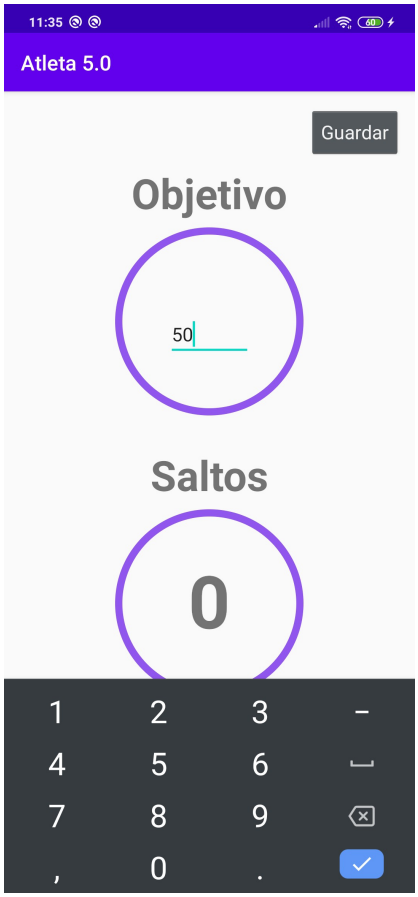
Gracias al botón flotante que se encuentra en la parte inferior derecha tenemos la opción de compartir nuestro entrenamiento a través de otras aplicaciones. Se abrirá un menú en la parte inferior donde podremos elegir qué aplicación utilizar (imagen izquierda). En este caso seleccionamos *WhatsApp* y tras elegir el contacto se creará el mensaje de forma automática para ser enviado (imagen derecha).

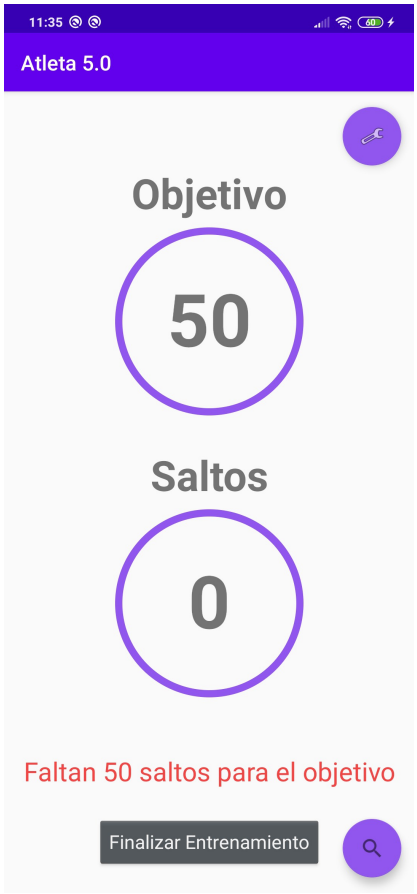


Caso de prueba 3

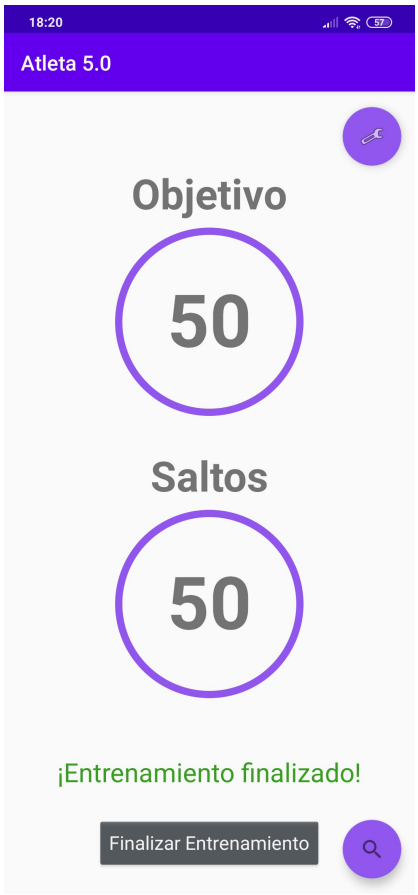
En el tercer caso de prueba realizaremos dos entrenamientos de salto estableciendo objetivos diferentes y después comprobaremos que el registro histórico es el correcto.

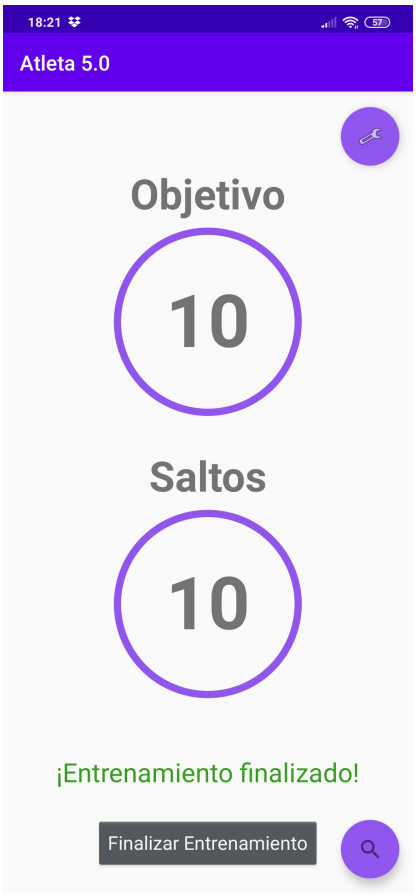
Seleccionamos en el menú principal la opción *Entrenamiento de salto*. En el botón de la esquina superior derecha podremos editar el objetivo (imagen izquierda). Una vez pulsado el botón *Guardar* se establece dicho objetivo y podremos comenzar el entrenamiento (imagen derecha).





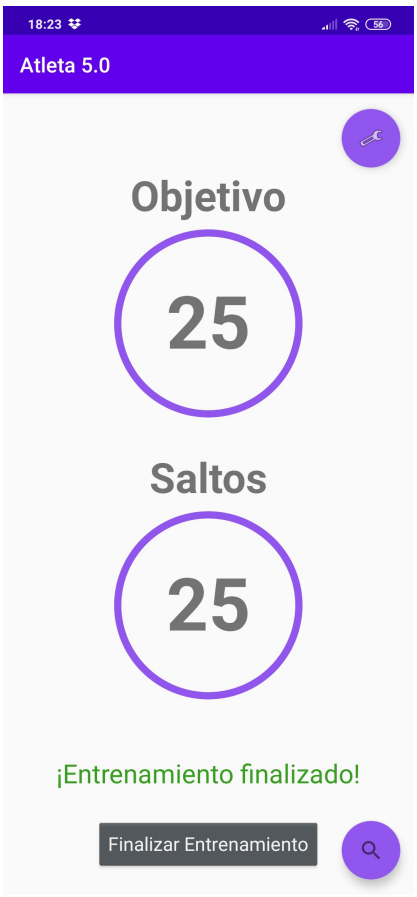
Cuando el usuario realice un salto el contador incrementará y se actualizará el número de saltos restantes para alcanzar el objetivo. En el momento en el que se consiga aparecerá un mensaje en la pantalla y el dispositivo vibrará para notificar al usuario.

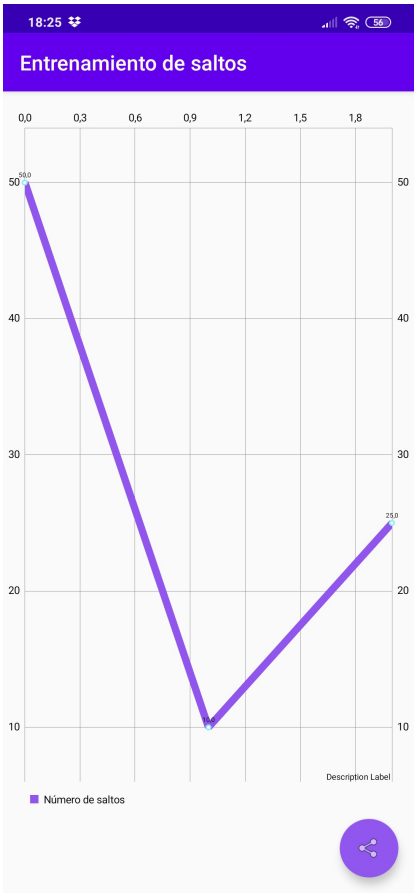




En la imagen de la izquierda podemos ver que se ha llegado al objetivo establecido de 50 saltos, mientras que en la derecha el objetivo fueron 10.

Nos marcamos un nuevo objetivo, esta vez de 25 saltos, y finalizamos el entrenamiento cuando llegamos a ese número (imagen izquierda). Tras la finalización de cada ejercicio se registran los datos para poder acceder al histórico. Pulsando sobre la lupa en la esquina inferior derecha accedemos a una gráfica comparativa que nos muestra la evolución de todos los entrenamientos de salto que hayamos realizado (imagen derecha).



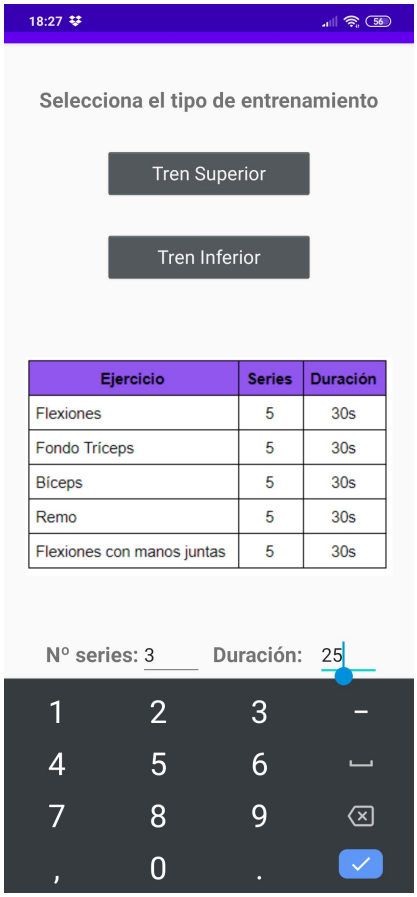


En la gráfica se observa el primer entrenamiento registrado, 50 saltos, seguido de 10 y acabando con los últimos 25.

Caso de prueba 4

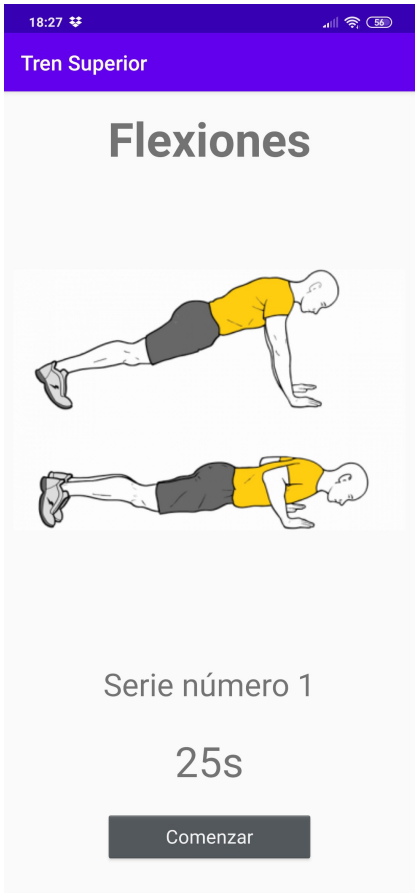
En el cuarto caso de prueba realizaremos un entrenamiento de tren superior utilizando los planes que nos ofrece la aplicación. Personalizaremos el número de series y la duración de los ejercicios antes de comenzar.

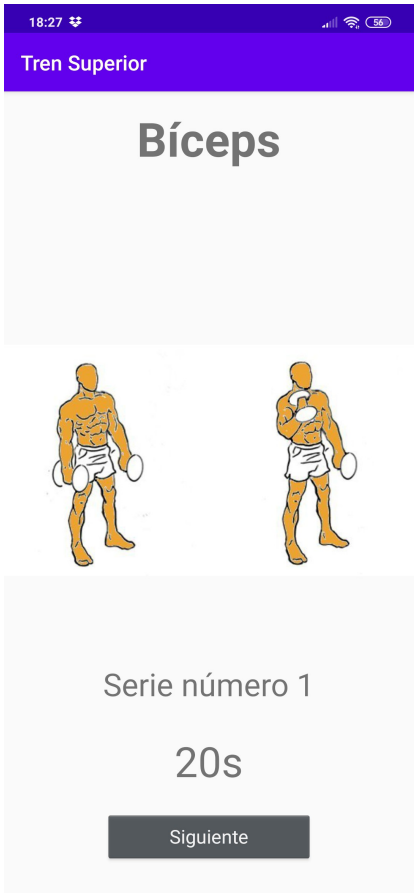
Seleccionamos en el menú principal la opción *Planes de entrenamiento*. Se nos muestra en la pantalla dos botones en la partes superior: *Tren Superior*, *Tren Inferior*. Presionamos en este caso el primer botón y se cargará en la parte central una tabla con un planning de entrenamiento donde se detallan tipos de ejercicio, número de series y duración (imagen izquierda).



Se activarán dos botones en la parte inferior: *Personalizar*, *Comenzar Entrenamiento*. Haciendo uso del primero podremos editar a nuestro gusto el número de series que queremos realizar y la duración de cada ejercicio (imagen derecha). Tras rellenar los datos presionamos el botón *Guardar y comenzar* para comenzar el entrenamiento.

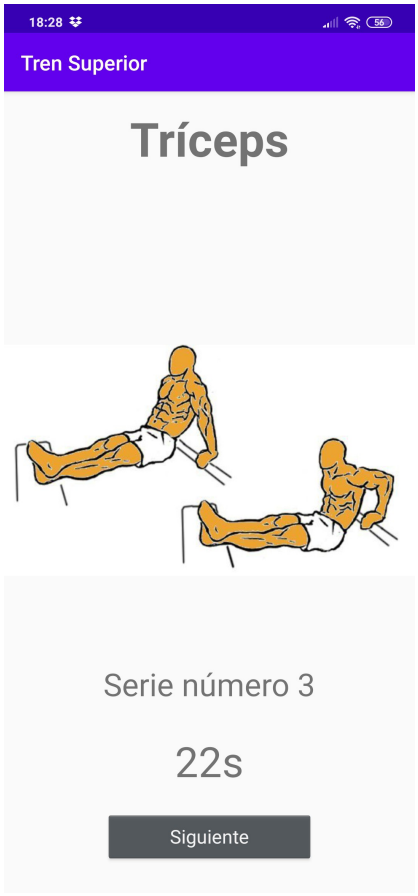
Comenzaremos entonces el entrenamiento y en la pantalla se nos mostrará una imagen explicativa del ejercicio que debemos realizar junto a la información del número de serie y el tiempo restante del ejercicio.

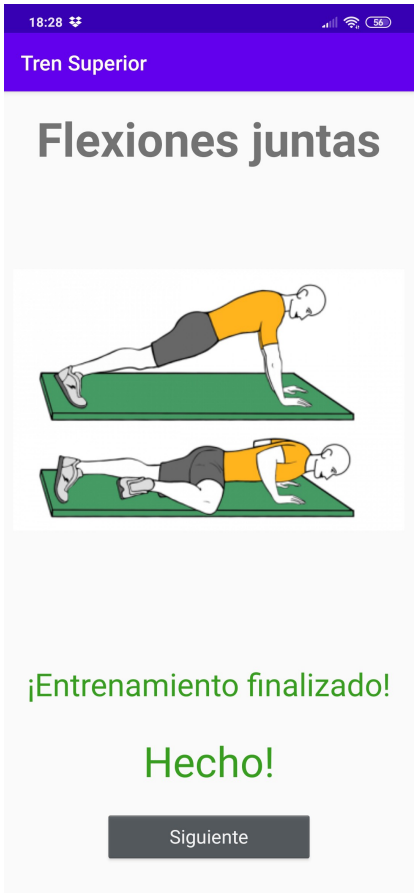




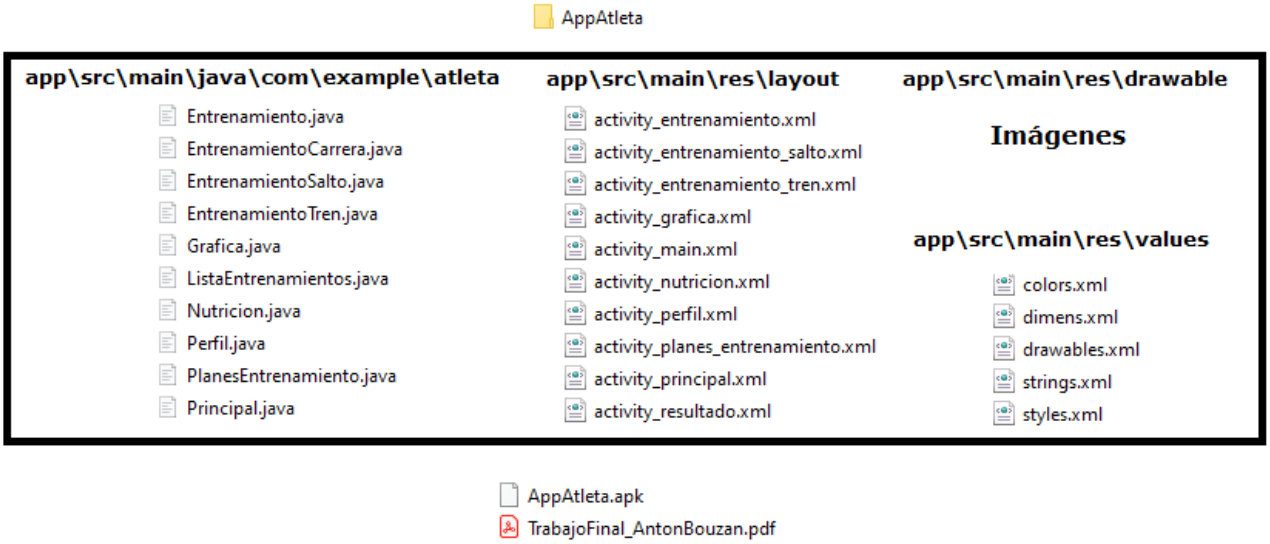
En el primer ejercicio el botón inferior *Comenzar* hará que el temporizador arranque (imagen izquierda). Cuando el tiempo finalice pasaremos al siguiente ejercicio (imagen derecha).

Seguimos avanzando en los ejercicios y la información mostrada por pantalla irá cambiando (imagen izquierda). Cuando acabemos el último ejercicio del entrenamiento (último ejercicio de la última serie) se mostrará un mensaje destacado en la pantalla y el teléfono vibrará (imagen derecha).





6 Ubicación de los programas fuente



La carpeta entregada contiene un directorio que forma el proyecto en Android Studio. Los archivos principales se encuentran ubicados en el esquema superior. También se entrega el archivo *.apk* para su instalación en un smartphone Android y este informe del proyecto.

7 Manual de uso e instalación

Para utilizar la aplicación en un smartphone basta con instalar el archivo *AppAtleta.apk* en el dispositivo. Otra opción sería utilizar la carpeta del proyecto en Android Studio y emplear un simulador para ejecutar la aplicación. Las funcionalidades están explicadas en el apartado *2.5 Descripción de casos de prueba*.

8 Conclusiones

El proceso de investigación y análisis para llegar al desarrollo de una aplicación Android que aúne las mejores características y funcionalidades me resultó muy interesante y enriquecedor a nivel profesional. Establecer los criterios que analizar para obtener una valoración objetiva de la aplicación me resultó útil para ver que realmente el desarrollo de una aplicación real abarca una gran cantidad de elementos funcionales y de diseño y que es indispensable que que todas las partes se hagan de forma correcta.

Como suele pasar en este tipo de proyectos, la planificación inicial y la final sufrió alguna variación debido a percances de tiempo para llegar a los hitos establecidos. De todas formas se ha conseguidos desarrollar en tiempo y forma la gran mayoría de elementos plasmados en el diagrama de Gantt.

Con los casos de prueba hemos comprobado que las funcionalidades establecidas en los requisitos iniciales se han cubierto de forma correcta.

Tras haber desarrollado aplicaciones para cursos y asignaturas de este propio máster quería llevarlo al siguiente nivel creando una app más completa tras haber investigado y analizado las opciones del mercado.

Está claro que cada vez existen más dispositivos IoT, pero el smartphone es y será el principal dispositivo que todo el mundo tenga en el día a día, por lo que el desarrollo de aplicaciones está cobrando un peso cada vez más importante.

9 Futuras líneas de trabajo

En este apartado se explican diferentes frentes en los que el el trabajo actual puede seguir desarrollándose para llegar a un mayor alcance.

Como líneas de futuras investigaciones podemos incluir:

- Integración con dispositivos wereables. Sería una opción muy interesante sincronizar y conectar el smartphone con la app instalada para poder obtener datos biométricos del usuario.

- Muestra de la ruta recorrida en los entrenamientos exteriores. Bastaría con guardar las coordenadas cada x metros para poder dibujar al finalizar el entrenamiento el recorrido que el usuario realizó durante el ejercicio.

- Implementación de Firebase Cloud Messaging. Para una cantidad grande de usuarios sería interesante implementar el chat de la aplicación en este servicio para así poder exprimir las características que ofrece (notificaciones a usuarios, envío de mensajes de texto, documentos o imágenes a usuarios o grupos etc

- Creación de entrenamientos conjuntos. Plantear la opción para que el usuario pueda si así lo desea compartir un entrenamiento con otro usuario de la plataforma y así entrenar de forma paralela. Esto hará que la app sea más colaborativa y divertida.

- Optimización del diseño para tablets. Debido al tamaño de pantalla de las tablets sería una buena idea crear un diseño optimizado para este tipo de dispositivos que hará que el usuario disfrute de una experiencia más agradable.

10 Bibliografía

- Web Oficial de Android Studio, https://developer.android.com/studio

- Sensores de movimiento-Android Developers, https://developer.android.com/guide/topics/sensors/sensors_motion?hl=es

- Usando acelerómetro en Android – CodePlus, https://code.tutsplus.com/es/tutorials/using-the-accelerometer-on-android--mobile-22125

- Dirección de una ubicación-Android Developers, https://developer.android.com/training/location/display-address?hl=es#java

- *Daniel Manrique Lucas*, *Desarrollo de una aplicación móvil Android para la gestión de citas de fisioterapia y la asignación de ejercicios* .TFM Universidad de Valladolid. 2017

- Tablas MET-Real Fitness, https://www.realfitness.es/calculadoras/aprende-utilizar-tablas-met-calcular-calorias-quemadas/

- Imágenes de los planes de entrenamiento – Entrenamiento.com, https://www.entrenamientos.com/

- Cómo firmar tu APP – AndroidStudio, https://developer.android.com/studio/publish/app-signing?hl=es-419

- *Japan Smartphone Security Asociaction, Android Aplicaction Secure Design.* Secure Coding Working Group. 2019

- *Chryssa Aliferi , Android Programming Cookbook*. Exelixis Media P.C. 2016

- Curso UNED - 3.0 Android Studio: Desarrollo de aplicaciones profesionales para móviles y smartwatches ,

  https://formacionpermanente.uned.es/tp_actividad/idactividad/10412

- *Dawn Griffiths & David Griffiths, Head First Android Development: A Brain-Friendly Guide.* O'Reilly. 2015