DESARROLLO DE APLICACIONES MULTIPLATAFORMA II

TRABAJO DE FIN DE GRADO

Andújar Cuenca, Francisco

Tutor: Arroyo, Manuel

IES Enrique Tierno Galván

15 de diciembre de 2021

Índice

Índice de Figuras

1. Introducción

Para la realización de este trabajo de fin de grado se plantearon distintas posibilidades, desde una aplicación de juegos, a una de compras como Amazon o AliExpress. Finalmente, me decidí por algo diferente “una aplicación de gimnasio”.

Yo suelo ir al gimnasio con una frecuencia media de dos o tres días por semana. Normalmente, voy con un amigo que conoce rutinas y ejercicios y me aconseja. Cuando me planteé cómo orientar mi Trabajo de Fin de Grado a algo que fuera práctico, me surgió una duda: si fuera sin compañía al gimnasio, ¿cómo podría establecer mis rutinas y seguimiento sin conocimiento alguno?

Investigué previamente sobre el tema y si bien hay personas que cuentan con entrenadores personales, existen muchos casos que no pueden permitirse económicamente este recurso. Esa era la idea principal del proyecto, crear una aplicación de apoyo para la actividad física y al alcance de todo el mundo.

Sangre de Espartano es una APP, que toda persona apuntada al gimnasio sin conocimiento previo pueda utilizar. En ella se puede consultar tu historial de calorías y el tiempo de entrenamiento de cada día. Además, te permitirá elegir entre múltiples rutinas que entrenan distintos grupos musculares pudiendo graduar la dificultad que más se adapte al usuario. Durante la sesión contará con un cronómetro que fomenta el esfuerzo, también se podrá visualizar un ejemplo visual a través de las animaciones de entrenamiento, lo que facilita su comprensión.

¿Pero por qué una aplicación? Actualmente, el móvil es una herramienta casi imprescindible en nuestras vidas, incluso dentro del gimnasio mucha gente lo usa para escuchar música. Por ello, la plataforma más útil para cumplir con el objetivo del proyecto era una app que funcione offline, ya que gracias al uso de SQLite los datos se almacenan internamente. El lenguaje de programación escogido es java, ya que es un lenguaje más extendido de lo que lo son sus competidores en desarrollo de aplicaciones, como lo es Kotlin. Además, para facilitar el desarrollo y la gestión de archivos de la app se usará un IDE, en concreto se usará Android Studio que da múltiples facilidades a la hora de desarrollar. Dentro de la aplicación se podría destacar la utilización de la biblioteca Glide, importada desde GitHub, que facilitara la realización de las animaciones, ya que te permite añadir Gifs animados a la aplicación.

2. Planificación

La planificación de este proyecto se llevará a cabo a través de la fusión de Kanban y Gantt. Gantt permite un buen planteamiento de cuánto tiempo te puede llevar la realización del trabajo, y así conocer el tiempo del que se dispone, ya que en todo momento se tiene en cuenta el tiempo, y Kanban te permite ver en qué fase están las tareas y subtareas del proyecto, por lo que juntando ambas te muestra la fecha fija de la finalización del proyecto y cómo van las tareas para ir viendo por dónde va el proyecto. Al ser un trabajo de fin de grado hay fecha límite, por lo tanto, esto me parece un método adecuado de planificación. A continuación, se muestra un ejemplo de diagramas de Kanban-Gantt del proyecto.

Figura 1. *Diagrama de Kanban-Gantt en desarrollo*

		25-31 Oct	1-7 Nov	8-14 Nov	15-21 Nov	22-28 Nov	29-5Dic
Activity Entrada	7h						
Eleccion y creacion del Diseño	4h						
Creacion De la Animacion	2h						
Musica	1h						
Activity Interfaz	7h						
Eleccion y creacion del Diseño	2h						
Editar De las imagenes	5h						
Activity Popup previsualizar	13h						
Eleccion y creacion del Diseño	5h						
obtener de gifs	8h						
Activity Visualizar Rutina	10h						
Eleccion y creacion del Diseño	8h						
Crear Cronometro Redondo	2h						
Activity Historial	8h						
Eleccion y creacion del Diseño	6h						
crear logica de almacenamiento	2h						
silbato	5min						
Creacion y automatizacion con BDs	20h						
Crear Clases Conexiones	3h						
Investigacion y creacion de rutinas	6h						
crear las bd con las rutinas	6h						
hacer que el codigo funcione con Bds	5h						
Exxtras de la app	1h						
quitar action bars	30min						
añadir icon	20min						
fijar Activitys	10min						
Documentacion:	19h						
Total:	85h						
		Completado	Incompleto	No iniciado			

Este sería un diagrama de los inicios del proyecto. Como se puede observar se visualiza el tiempo planeado que tienes para realizar cada tarea, el proceso por el que va, si se han completado o no las tareas que debía realizar y todo lo que ofrece este método de planificación.

Figura 2. *Diagrama de Kanban-Gantt finalizado*

		25-31 Oct	1-7 Nov	8-14 Nov	15-21 Nov	22-28 Nov	29-5Dic
Activity Entrada	7h						
Eleccion y creacion del Diseño	4h						
Creacion De la Animacion	2h						
Musica	1h						
Activity Interfaz	7h						
Eleccion y creacion del Diseño	2h						
Editar De las imagenes	5h						
Activity Popup previsualizar	13h						
Eleccion y creacion del Diseño	5h						
obtener de gifs	8h						
Activity Visualizar Rutina	10h						
Eleccion y creacion del Diseño	8h						
Crear Cronometro Redondo	2h						
Activity Historial	8h						
Eleccion y creacion del Diseño	6h						
crear logica de almacenamiento	2h						
silbato	5min						
Creacion y automatizacion con BDs	20h						
Crear Clases Conexiones	3h						
Investigacion y creacion de rutinas	6h						
crear las bd con las rutinas	6h						
hacer que el codigo funcione con Bds	5h						
Exxtras de la app	1h						
quitar action bars	30min						
añadir icon	20min						
fijar Activitys	10min						
Documentacion:	19h						
Total:	85h						
	Completado	Incompleto	No iniciado				

El diagrama de arriba sería una vez completo. Se puede destacar que la documentación, como se hace a medida que el proyecto avanza, está en todas las fechas.

3. Desarrollo

Sangre de espartano es una aplicación desarrollada con el lenguaje de programación Java y en el entorno de desarrollo Android Studio, se aprovecha de una biblioteca de GitHub llamada Glide para sus animaciones y usa bases de datos internas SQLite.

3.1. El flujo entre activities

A la hora de diseñar la idea se pensó en una estructura sencilla,con el fin de facilitar la comprensión al usuario. Este consta de una entrada, una interfaz de elección del ejercicio y un *activity* que muestre la rutina.

Figura 3. *Flujo entre activities en su planificación.*

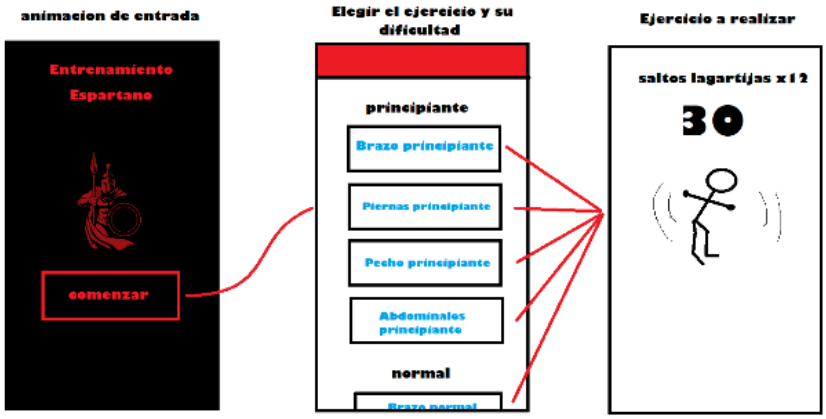
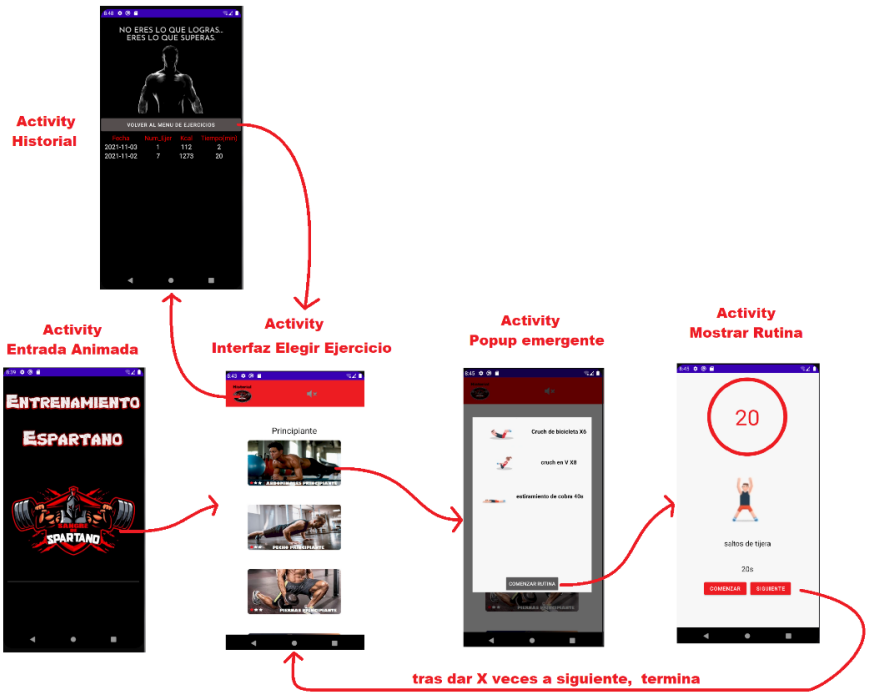


Figura 4. *Flujo entre activities una vez desarrollada la aplicación*



El flujo de la aplicación ha cambiado un poco, pero conserva la esencia. La aplicación comienza en la *activity* “Entrada Animada”, desde ella se podrá acceder a la “Interfaz Elegir Ejercicio” pulsando la imagen del espartano con la pesa, o esperando tiempo determinado. Una vez llegado a la “Interfaz Elegir Ejercicio”, se podrán observar las doce opciones de rutina en las que se puede elegir entre tres dificultades y cuatro grupos musculares.

Se puede optar entre dos flujos o caminos, ambos igual de importantes: un camino es ir a ver el “Historial” y el otro seleccionar una rutina. Para poder acceder a “Historial”, se pulsará en la imagen del logo de la aplicación arriba a la izquierda y tras pulsarlo se abrirá la *activity* “Historial”. En ella, se encontrará una tabla con el registro del ejercicio realizado: día que se realizó, número de rutinas, las calorías quemadas y el tiempo. Para volver a la “Interfaz Elegir Ejercicio” se toca el botón gris de debajo de la imagen de la silueta del hombre.

El otro flujo que permite la *activity* es pulsando una de las rutinas. Entonces saldrá un “Popup Emergente” donde se visualizarán los ejercicios que contiene esa rutina. Para acceder a la siguiente *activity* se debe pulsar el botón “comenzar rutina”, el cual permanece estático en la parte inferior. Tras pulsar el botón llegarás a esta *activity* en la que se puede observar un cronómetro, una animación y un par de botones. El botón “siguiente” cambiará al próximo ejercicio de la rutina, al pulsar varias veces llegará el momento que terminen las rutinas para realizar y aparezca el botón “finalizar”. Una vez terminada la actividad se cerrará e irá otra vez a la *activity* “Interfaz Elegir Ejercicio”.

Ahora se procederá a explicar el código de cada una de las *Activities* que hay de forma más detallada, mirando cómo funciona internamente.

3.2. Entrada Animada

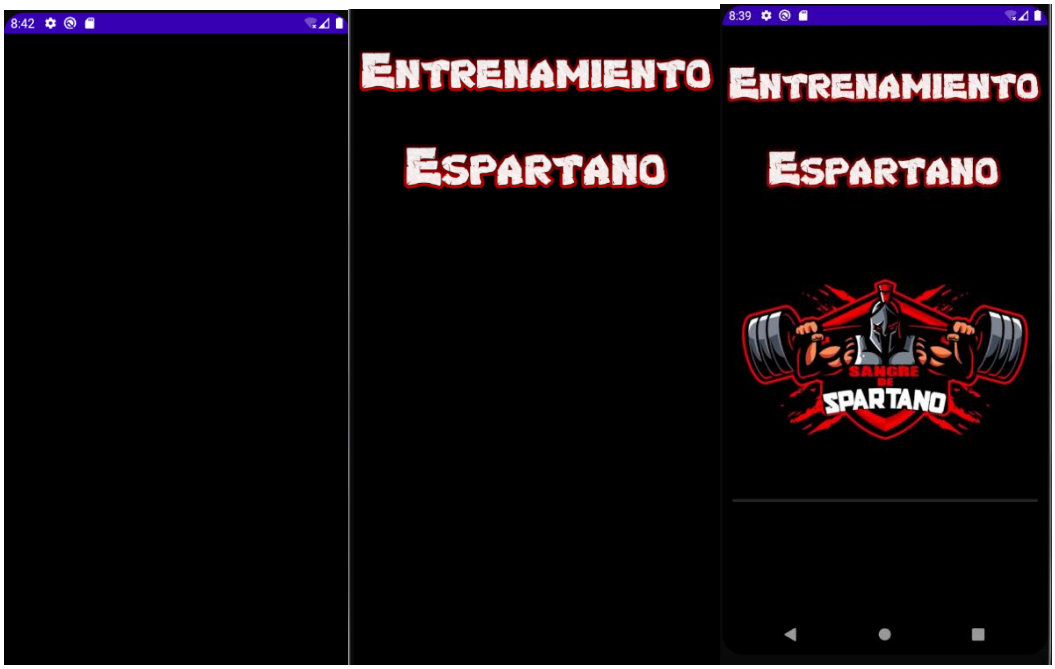
La *activity* “Entrada Animada” principalmente consta de tres imágenes y un botón imagen. La actividad inicia con una canción que suena hasta que termine (o lo apaguen en la siguiente actividad), para ello se usa la clase MediaPlayer que permite la reproducción de sonido. Como se quería poder parar en la siguiente actividad y que no solo se escuchara en la entrada, se creó una clase llamada “controladorMusica” a la cual se la debe pasar el contexto e inicia la música.

Ver Anexo 6.1. Controlador de música.

Además, se creó una animación alrededor de la música en la que por cada golpe de la música, aparece una imagen que parpadea hasta que se completa la actividad. Para su realización se utilizó Handler ya que permite interactuar con el hilo principal de la actividad y así, poder modificar la opacidad de las imágenes de cero a uno y viceversa, dando la sensación de que aparece y desaparece parpadeando. Para ello se usan varios Handler para que cuando pase una cantidad de tiempo, realicen la animación de aparecer y desaparecer (para el destello se usan varios muy pegados en tiempo).

Ver Anexo 6.2. Temporizadores de la animación.

Figura 5. *Activity Entrada Animada (animación).*



En esta actividad se crean las bases de datos de los ejercicios, en el caso de que sea la primera vez de uso de la aplicación en tu dispositivo. Para ello se realiza una comprobación de si existen datos previos, porque si no se hiciera esto, cada vez que se abriese la aplicación y no se comprobase, se insertarían y duplicarían en las bases de datos.

Ver Anexo 6.3. Comprobación de la existencia de bases de datos.

Para la creación de las bases de datos se han usado métodos estáticos, todos ellos dentro de una misma clase para facilitar su visualización. Los métodos al ser estáticos pueden ser llamados desde cualquier clase, lo que facilita la creación, ya que se permite en el momento que se crea más conveniente sin previa planificación. El método requiere de un contexto que deberá ser pasado, en el “Anexo 6.4” se puede observar uno de los ejemplos de estos métodos estáticos.

Ver Anexo 6.4. Creación de las bases de datos.

Como se puede observar en el “Anexo 6.4” este método contiene otra clase llamada ConexionSQLiteHelperAbp (cada rutina tiene la suya). Esta clase extiende de SQLiteOpenHelper con sus métodos heredados, los cuales permiten crear, gestionar y actualizar, la base de datos fácilmente.

Ver Anexo 6.5. Clase que extiende de “ConexionSQLiteHelper”.

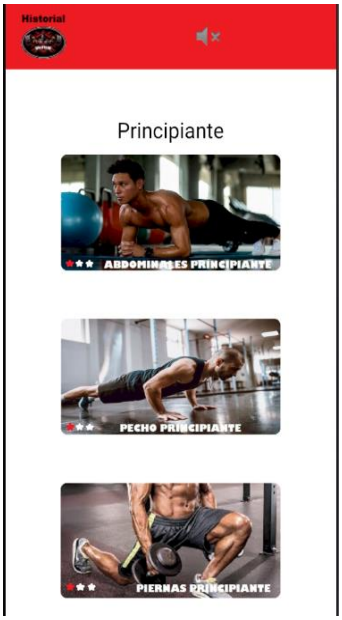
Además, como era recomendado, se ha creado una clase “Utilidades”, la cual permite la fácil reutilización de código, sin dar lugar a equivocación. La clase “utilidades” es sobre todo usada por “ConexionSQLiteHelperAbp”, ya que contiene las sentencias SQL que crean las tablas, sus filas y los nombres de los campos.

Ver Anexo 6.6. Clase Utilidades.

Una vez creadas las tablas, contendrán los siguientes campos: el nombre del gif del ejercicio, el nombre del ejercicio, las repeticiones y el tiempo. Como se puede observar, destaca sobre todo el nombre gif, al ser un dato de tipo String. Esto se debe a que a la hora de la planificación de la creación de la base de datos, se consideró más sencillo el guardar un dato tipo String, que haga referencia al elemento que se necesita recuperar, que al elemento completo. El nombre del gif luego se transformará en un drawable que animará Glide, como se mencionará más adelante.

3.3. Interfaz Elegir Ejercicio

Figura 6. *Activity Interfaz Elegir Ejercicio.*



En la interfaz podemos observar que hay catorce botones con imágenes. Para ver todas se tendrá que deslizar hacia abajo, ya que este *activity* permite hacer “scroll” vertical. Los 2 primeros botones están en la franja roja, uno es el botón de historia, el cual permite acceder al *activity* historial y el otro es el que detiene la música de la primera actividad, llamando al método *parar()* de la clase

propia “ControladorMusica”. Esta clase al tener guardada la música en su variable estática, permite el acceso desde cualquier otra *activity.* La música está guardada en la carpeta raw como se indica en Android Developer.

Ver Anexo 6.7. Método parar de la clase ControladorMusica .

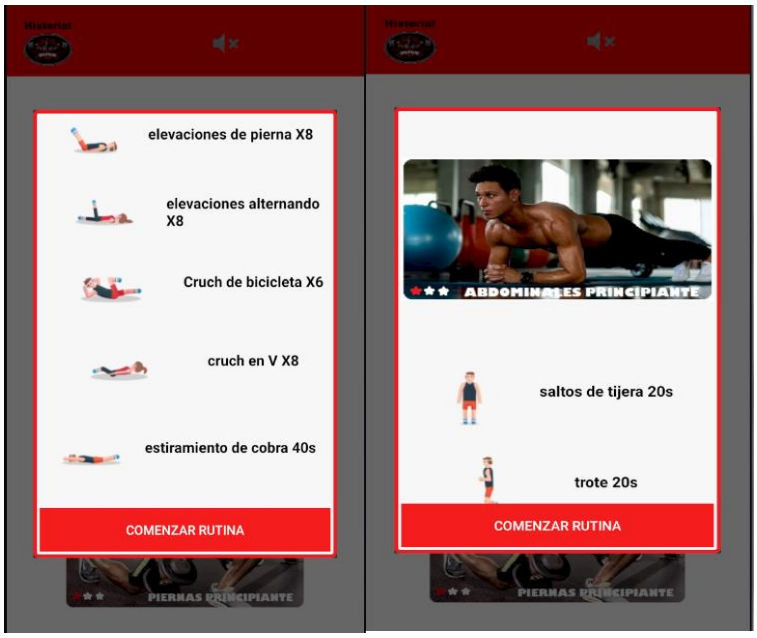
Los otros doce botones son las rutinas entre las que poder elegir. Estas abren la actividad de “Popup” a través de un intent, pero se le añadirá en el intent un Bundle que llevará los dos siguientes datos, un distintivo de la rutina elegida y las kilocalorías que quema esa rutina. La cantidad de calorías es la del conjunto de la rutina, por lo que para quemarlas y que se añadan en el historia, se debe completar la rutina.

Este *Bundle* permite recuperar estos datos en la siguiente rutina, permitiendo así saber cuál fue la rutina escogida.

Ver Anexo 6.8. Método del botón “Popup”.

3.4.Popup Emergente

Figura 7. *Activity Popup Emergente.*



La clase “*Popup*” permite la visualización rápida de las rutinas antes de iniciarlas. Todas las rutinas se cargan en este mismo *activity*, la única diferencia de una opción a otra, son las diferentes rutinas que se cargan.

El “*Popup*” es una *activity* normal a la cual se le ha modificado la apariencia, mostrándose como una ventana emergente. Este estilo de *activity* es óptimo para este caso, ya que da la sensación de no haber comenzado la rutina,

únicamente muestra los datos. Además, pulsando fuera del “Popup”, se le permite rápidamente al usuario volver atrás, lo que ofrece un sistema de uso sencillo. Para la creación de este tipo de *activity* lo primero que debe cambiarse es el tamaño, para que este no ocupe la totalidad de la pantalla.

Ver Anexo 6.8. Cambiar la resolución de la pantalla.

El código del “Anexo 6.8” coge los píxeles del ancho y alto de la pantalla. Después, se multiplican por el porcentaje del tamaño que se quiere. Este previamente ha sido dividido entre 100, lo que hará que cambie la resolución del *activity*.

Además de cambiarle la resolución, se deben modificar otros aspectos como que el fondo sea translúcido, que al tocar fuera del activity “*Popup”* este se cierre (se realiza modificando la resolución, si no se cambia la resolución previamente, no se podría tocar fuera, ya que abarcaría toda la pantalla), que sea flotante y que no aparezca el título del proyecto en la parte superior. Para ello, se ha creado un estilo en el XML de Theme.

El estilo que usa una *activity* está especificado en el Android Manifest, por lo que para su uso se modificará ahí.

Ver Anexo 6.9. Estilo Propio y Androd Manifest.

En el “Popup” se obtienen los datos de la base de datos. Estos datos se guardarán en cuatro ArrayList respectivos para guardar el nombre de los gifs, el nombre del ejercicio, el número de repeticiones y el tiempo que se debe tardar en hacer esa rutina. Además, para no escribir de uno en uno todos los ejercicios, se producirá un bucle que rellenará todos los datos. También, habrá otros dos ArrayList uno que va a contener los ID de los imagen-gifs, y otro que contendrá los ID de los TextView, reduciendo así la cantidad de código, lo que lo hace más visible y comprensible.

Ver Anexo 6.10. ArrayLists contenedores.

El Popup al ser el mismo *activity* para todos, se le debe pasar en el Intent un objeto Bundle que contiene un distintivo. Este distintivo permite diferenciar la rutina que se ha escogido y así, saber qué base de datos se debe recuperar.

El motivo por el cual se decidió crear un solo *activity* desde el que cargar las diferentes rutinas, es porque facilita la legibilidad del proyecto al no tener que crear doce *activities*. Estas *activities* contendrían cada una su propia rutina ya hardcodeada. Aunque a nivel de programación resulta más sencillo a la hora de programarlo, es mucho más complicado cuando el proyecto se debe modificar y entender.

Ver Anexo 6.11. Distintivo entre Popups.

Para comparar el distintivo recuperado se usa un *switch*, este es más elegante y legible que un conjunto de “if-else” anidados. Dentro de cada caso del *switch,* se creará una conexión a la base de datos correspondiente, gracias al uso de una clase propia que permite gestionar y acceder a una de las bases de datos. Un ejemplo de una de estas clases es “ConexiónSQLHelperAbP”, que permite acceder a la base de datos de abdominales para principiantes.

Una vez encontrada y obtenida la base de datos correspondiente, se procederá a rellenar con los datos recuperados de la base de datos, los respectivos ArrayLists que contendrán esos datos del ejercicio. Una vez guardados los datos de la rutina, se crea un bucle que rellena las imágenes y textos vacíos almacenados a través de su ID en ArrayLists.

Ver Anexo 6.12. Switch de obtención de datos.

Ver Anexo 6.13. Bucle de cargado Popup.

En el “Anexo 6.13” se puede ver por primera vez la biblioteca Glide. Para poder usar Glide, se debe especificar en el constructor de la aplicación las bibliotecas que se van a utilizar, llamado *Gradle*. Glide se encuentra en GitHub, por lo que se descarga automáticamente al añadir estas dos líneas de código en las dependencias.

Ver Anexo 6.14. Dependencias Glide.

La biblioteca Glide permite que al agregar un elemento .gif a un ImageView, este siga siendo un gif con movimiento y no se convierta en una imagen estática. Además, cabe explicar que para su uso se debe dar un contexto, un drawable y un recurso donde cargarlo.

La aplicación usa un método peculiar e ingenioso para guardar las imágenes-gifs que se usan en las rutinas. Se optó por su sencillez a la hora de guardar los datos, que solo se guardase el nombre del gif en formato String, consiguiendo a posteriori, con múltiples métodos, el elemento drawable a través del nombre del gif.

Ver Anexo 6.15. Ejemplo explicado de cómo usar Glide.

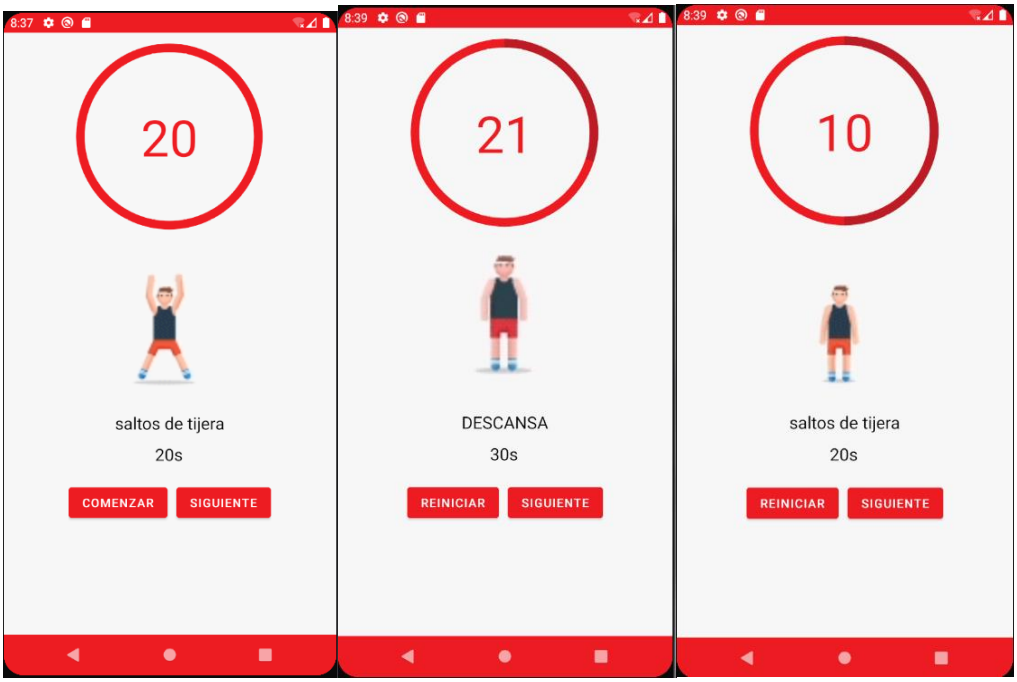
Por motivos de diseño el botón de comenzar se mantendrá en la parte inferior del “Popup”, debido a que es más intuitivo para el usuario saber así donde pulsar para continuar. Se planteó la opción de que estuviera fijo abajo del todo del “scroll”, obligando así a ver todo el ejercicio que quiera realizar.

El botón “comenzar rutina” realiza la función de recoger las calorías que se queman la rutina, las cuales fueron recuperadas del anterior Bundle y manda estas calorías más los cuatro ArrayList al siguiente *activity*. Sin esta función, sería poco eficiente tener que mandar otro distintivo, para luego volver a recuperar los datos de la base de datos de nuevo en la siguiente actividad. Por esto, una vez recuperados los datos para la visualización rápida en el “Popup”, se aprovechan si se acaba escogiendo la rutina. Todo esto mostraría el “Popup” que se ve en la “Figura.7”.

Ver Anexo 6.16. Botón comenzar rutina

3.5. Mostrar Rutina

Figura 8. *Activity Mostrar Rutina.*



Esta *activity* consta de un cronómetro, una imagen-gif, un texto y dos botones. La actividad comienza recuperando a través de Bundle, los ejercicios que aparecerán y se guardarán en sus respectivos ArrayList. El cronómetro explicado de forma sencilla tiene un hilo que funciona cuando la variable booleana es “verdadero”. Esta variable se pone en verdadero cuando tocas el botón “Comenzar”. Dentro del hilo hay un método *sleep,* el cual dormirá al hilo durante un segundo y, cuando este tiempo pase, sucederán dos cosas:

La primera restará un segundo a la variable que guarda los segundos y llevará el cronómetro y, la segunda, crea un Handler que modifica el texto del tiempo que lleva el cronómetro hasta llegar a 0 segundos, ya que como se ha

mencionado anteriormente, este tipo de hilo permite interactuar con el principal y modificar los elementos visuales.

Esta forma sería la más sencilla. Sin embargo, el cronómetro de esta aplicación es mucho más complejo, ya que permite comenzar, reiniciar y pasar a siguiente. Una vez se termina el ejercicio llegando a los 0 segundos, volverá a arrancar el cronómetro para iniciar el descanso. Tras terminarse el descanso, pasará automáticamente al siguiente ejercicio. Se puede en todo momento reiniciar el ejercicio o pasar a siguiente ejercicio, hasta que se terminen todos estos.

Además, el cronómetro tiene un ProgressBar circular. Este se consigue haciendo dos drawables con XML, uno de un color determinado, cuando está el progreso a cero, y otro de distinto color cuando esté completo. Para permitir que el elemento XML se pueda usar en el ProgressBar, se debe usar la propiedad “useLevel”.

Ver Anexo 6.17.XML del círculo sin progreso

El botón “Siguiente” carga en la *activity* el siguiente ejercicio, el cual se obtiene de los cuatro ArrayList. Todo esto hasta que se acaben los ejercicios.

Una vez que los ejercicios se acaban, el botón pondrá “Finalizar” y tras pulsarse se procederá a guardar la información recogida a la base de datos del historial. Primero obtendrá la fecha de hoy, y después pedirá dicha información a la base de datos e irá comparando en bucle si hay alguno con esa fecha. En caso de que exista, recogerá los datos y un booleano dirá que existe la fecha, de lo contrario, dentro de ese bucle no hará nada y el booleano dirá que no existe.

Si existe procederá a hacer un *update* con los datos obtenidos, al que se le sumarán los datos del ejercicio realizado. En el caso de que no exista, en vez de un *update* se hará un *insert*, al no haber datos recuperados, solo tendrá los de este ejercicio y los insertará en la tabla por primera vez. Si no se comprobase y se hiciera directamente *insert* se podrían duplicar los datos y habría uno por cada ejercicio hecho en el día.

Los datos que se insertan en la tabla son los siguientes:

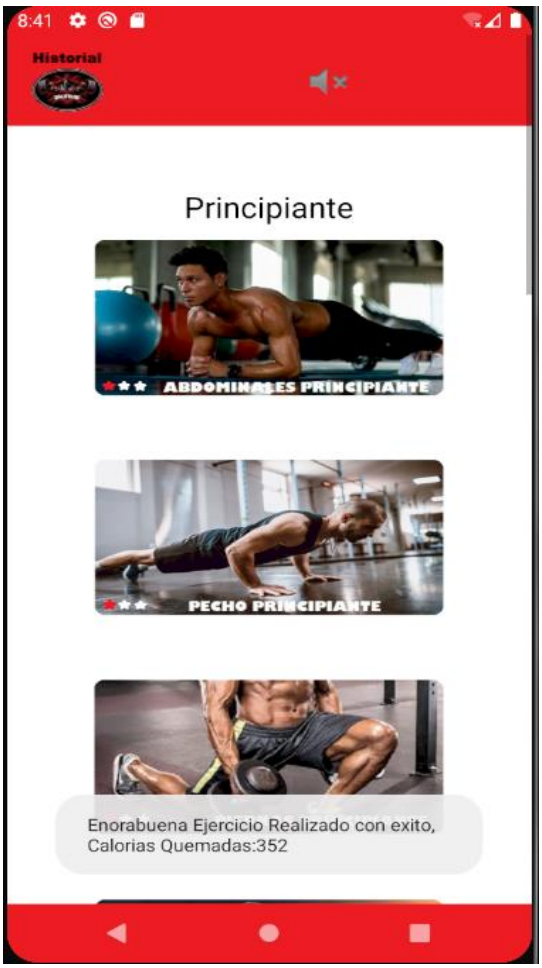
- La **fecha**, que será la del día.
- **Número de rutinas**, es decir, se le sumará +1 al terminarla.
- Las **calorías**, las cuales se llevan pasando desde la actividad “Interfaz Elegir Ejercicio”.

- El **tiempo** que transcurre desde el inicio del ejercicio. Este dato se obtiene del segundo hilo que tiene la actividad, el cual hace lo mismo que el cronómetro. Este, sin embargo, no se para en ningún momento desde que inicia la actividad y, cada vez que pasa un minuto, o dicho de otra manera, duerme 60.000 milisegundos, sumará 1 al tiempo. Este tiempo es el que luego se añade. Se eligió el minuto por que es la unidad de medida más apropiada a la hora de medir el ejercicio realizado.

Tras terminar todos los ejercicios se vuelve a “Interfaz Elegir Ejercicio” mostrando las calorías quemadas. Cabe mencionar que fue necesario crear un método para conseguir el contexto porque no se podía obtener a través “this”.

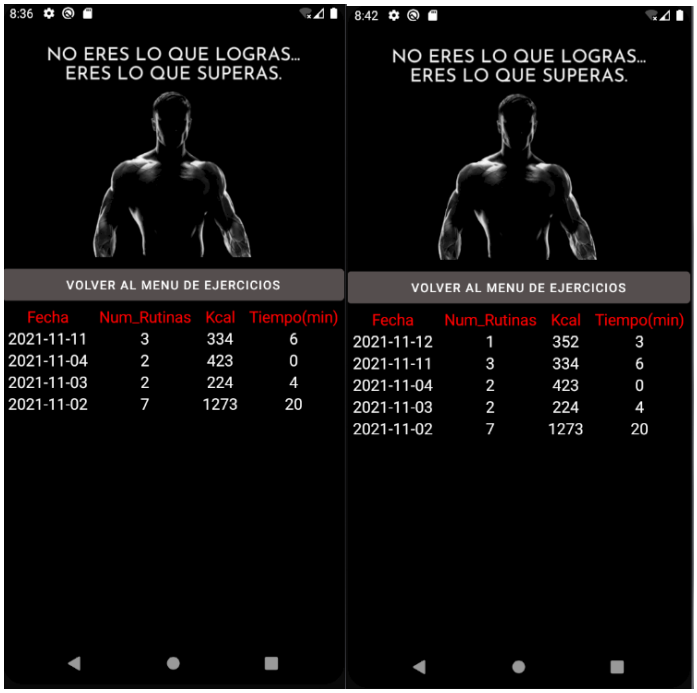
Ver Anexo 6.18. Obtención de un contexto.

Figura 9. *Regreso al Activity Interfaz tras realizar el ejercicio.*



3.6. Historial

Figura 10. *Activity Historial.*



La *activity* historial se encarga de visualizar la información que se obtenga de la base de datos, ya que todos los procesos de actualización y creación ocurren al terminar una de las rutinas por completo.

Al tratarse de una cantidad de datos desconocida, este deberá generarse dinámicamente. Para ello, se usará una tabla que se rellenará automáticamente. La tabla es una clase propia creada que se llama “TableDynamic”, a la cual hay que proporcionar un objeto tabla y el contexto, dicha clase contiene los métodos de crear tabla, crear cabecera, crear fila, crear columna, etc.

Para crear una nueva tabla dinámica se le proporcionará el objeto tabla de la *activity* y el contexto. Posteriormente, en el método cabecera se le debe dar un array de Strings con los nombres de las columnas y para rellenar la tabla con datos debe recibir un ArrayList de arrays de Strings.

Ver Anexo 6.19. Creación de una tabla dinámica.

3.7. Otras cosas a tener en cuenta

Hay aspectos generales de la aplicación fuera de las actividades que se deben tener en cuenta. Algunos de los elementos son la creación del icono de la aplicación, la orientación de la pantalla, etc.

3.7.1. El icono

En el icono de la aplicación viene por defecto un logo de Android con un fondo verde, eso no resulta atractivo para la aplicación y no permitiría la diferenciación entre aplicaciones si todos mantuvieran el mismo.

Por ello, para cambiarlo, lo primero que se tiene que hacer es borrar todas las imágenes del icono que están en res/mipmap. Cómo está mapeado para diferentes tamaños de móviles, hay que eliminar varias imágenes. Una vez borradas las imágenes, si uno se sitúa en el visualizador del proyecto en la carpeta res y da clic derecho sobre él, aparece la opción “Image Asset”, el cual, permite generar un icono mapeado. Una vez es clicado, se abrirá una ventana que ayudará a la creación de ese icono. El único elemento que se debe cambiar es el path/dirección de la imagen de la que se desee obtener los iconos. Después solo se deberá dar a next y finish, completando así la creación de los iconos. Si se cambia el nombre saldrá un error ya que en los XML apuntan a los elementos borrados al principio, simplemente se deberá cambiar el nombre en los XML por el nombre que se le puso al recurso.

3.7.2. La orientación

La orientación en el móvil es compleja. El móvil contiene un elemento llamado giroscopio que permite detectar en qué posición se encuentra, puede encontrarse en horizontal o vertical lo que hace que cambie el ancho y el largo de la pantalla. El desarrollador para adaptarse a ese cambio tiene dos opciones: la primera opción es crear un estilo/diseño para cada posición y la segunda opción es fijar la orientación a la que el desarrollador crea más conveniente.

Para esta aplicación se optó por la segunda opción: un fijado vertical en toda la aplicación, debido a que no se tiene la necesidad de poner el móvil en horizontal. La orientación horizontal es más habitual en aplicaciones de juegos o visualización de series como lo es Netflix. La orientación del móvil se fijó en el Android Manifest, en cada actividad se debe de indicar que se cambiará la configuración de la orientación y la orientación que se desea fijar.

Ver Anexo 6.20. Fijar orientación.

3.7.3. El KeepScreenOn

El mantener la pantalla encendida es un atributo muy importante, sobre todo en esta aplicación, debido a que el móvil si no es utilizado en un tiempo se va oscureciendo la pantalla hasta apagarse. Si no fuera por este atributo, a la hora de realizar ejercicios largos de más de 70 segundos, al no haber interacción con el móvil este se apagaría la pantalla.

Por recomendación de Android Developer esto solo se debe usar en la actividad, nunca en un proceso o componente. Este atributo se puede realizar de dos maneras que son equivalentes: la primera a través del XML o la segunda a través del código de programación dentro del OnCreate(), siendo este último más útil debido a que puedes cambiar en un futuro esta cualidad y que se pueda poner en reposo. Sin embargo, a pesar de ser más útil, en el caso de esta aplicación fue usado el atributo XML, debido a que no se requiere denegarlo en ningún momento.

Ver Anexo 6.21. Mantener pantalla encendida.

3.7.4. El StatusBar y NavegationBar

El StatusBar y NavegationBar son dos herramientas que se crean por defecto en toda aplicación. La barra de estado se encuentra en la parte superior y te dice la hora, batería, cobertura, etc. y la barra de navegación te permite regresar a la actividad anterior o visualizar todas las aplicaciones en funcionamiento, por lo que fue considerado poco aconsejable eliminarlas de la aplicación por su gran utilidad.

Estas herramientas vienen con unos colores por defecto que la aplicación crea como colores primarios de la aplicación, los cuales pueden no ser los que se planean usar para la aplicación. Hay dos opciones: modificarlo desde el XML que muestra dichos colores primarios o como se realizó en el caso de esta aplicación, cuando se crea la actividad, modificarles el color a través del código de programación de la aplicación. Los colores de esta aplicación son rojo, blanco y negro, por lo que se usa el color predominante de la actividad en estas dos herramientas para resaltar los colores de la aplicación.

Ver Anexo 6.22. Seleccionar color del StatusBar y NavegationBar.

4. Conclusiones

En primer lugar, mencionar que los objetivos principales del proyecto de desarrollo de la aplicación se han cumplido. Especialmente la parte visual, que puede resultar más costosa. El proyecto ha contado con numerosos puntos de vista de amistades que han utilizado la app. Estos, han hecho posible mejoras en problemas concretos, que solo podrían conocerse al utilizar la aplicación durante la actividad física.

El proyecto aporta diversos conocimientos útiles para futuras aplicaciones, como la creación de bases de datos internas con SQLite, el cual resulta muy útil para la persistencia de datos sobre todo la que se genera dentro

de la aplicación como lo es el historial. Las rutinas podrían estar hard-codeadas y no pasaría nada porque es información que no se altera ni modifica.

En el desarrollo de la aplicación, he aprendido que es necesario comprobar si existen bases de datos previas, haciendo una conexión, ya que si estas existen y las vuelves a insertar los datos se duplicarían. Además, he aprendido que se debe usar “Web Servicies” si lo que deseas es tener acceso a una base de datos externa, ya que te proporciona información de forma ligera, siempre que tengas permisos de acceso a internet.

He de destacar la importancia del ciclo de vida, porque no todo tiene porqué suceder cuando se crea el *activity*. Se puede parar y al reanudar querer hacer ciertas cosas, para gestionar estos casos hay que utilizar los métodos del ciclo de vida.

Como conclusión del proyecto quería destacar algunos nuevos conocimientos adquiridos durante el desarrollo de la aplicación y que me resultan muy enriquecedores para el futuro.

Por ejemplo, la importancia del Handler. Si se emplea un hilo normal, este no tiene acceso a modificar elementos del *activity*, al no pertenecer al hilo principal. Handler es un tipo de hilo con el cual poder acceder a modificar esos elementos. Por otro lado, hasta este proyecto tampoco había creado antes un elemento drawable XML, y es muy conveniente en casos con cronómetro para modificar la forma de la barra de progreso. Otra cuestión podría ser la creación de una tabla dinámica.

Las bibliotecas de GitHub son sencillas de instalar y ofrecen muchas facilidades, en ella encontré Glide, el cual facilitó mucho la animación del ejercicio. Gracias a Glide se posibilitó la aparición de gifs, algo imprescindible para dinamizar la aplicación. No resultó tarea fácil porque los gifs animados encontrados en internet venían en un bloque y era necesario separarlos uno a uno. Posteriormente, investigando descubrí que hay páginas web que te permiten recortarlos de forma online, sin embargo, al recortarlos algunos elementos aparecían negros en el gif. Porbé con programas que graban y crean gifs y el resultado fue el mismo, hasta que, finalmente, descubrí que era por la resolución. Por lo que, quedaba recortar los 90 gifs seleccionados, con los que seguidamente creé las 12 rutinas.

5. Visión de futuro

Una vez finalizado el proyecto, es necesario realizar un análisis de las posibles mejoras de la aplicación resultante. Objetivamente, al no tener estudios específicos sobre la materia de diseño gráfico, las animaciones no son propias y

en el proceso de desarrollo de la aplicación han podido perder cierta calidad. De esta manera, una posible mejora de cara al futuro podría ser la estética general de la aplicación de modo que resulte más llamativa y original para los usuarios.

Otras aplicaciones, cuentan con un historial de informes sincronizado a un correo electrónico, un apartado que te permite establecer recordatorios, pestañas que permiten crear un plan de entrenamiento personalizado para el usuario de la aplicación y la posibilidad de elegir entre varios idiomas, permitiendo tener una aplicación global con la cual acceder a mayor número de usuarios.

Una idea interesante sería que la aplicación te de información de la meteorología y dependiendo de esta recomendarte un entrenamiento en exterior o interior. Sin embargo, no consideré la idea en su momento ya que el API resulta costoso y requiere de recursos de los que no dispongo. Las versiones gratuitas solo permiten 50 llamadas al día, de tal forma, pensé en hacer mi propio *web servicie* que distribuyera esta información. Llegado a este punto solo podrían utilizar esta característica los de una zona en concreto, de modo que, no lo consideré relevante para el objetivo de la aplicación.

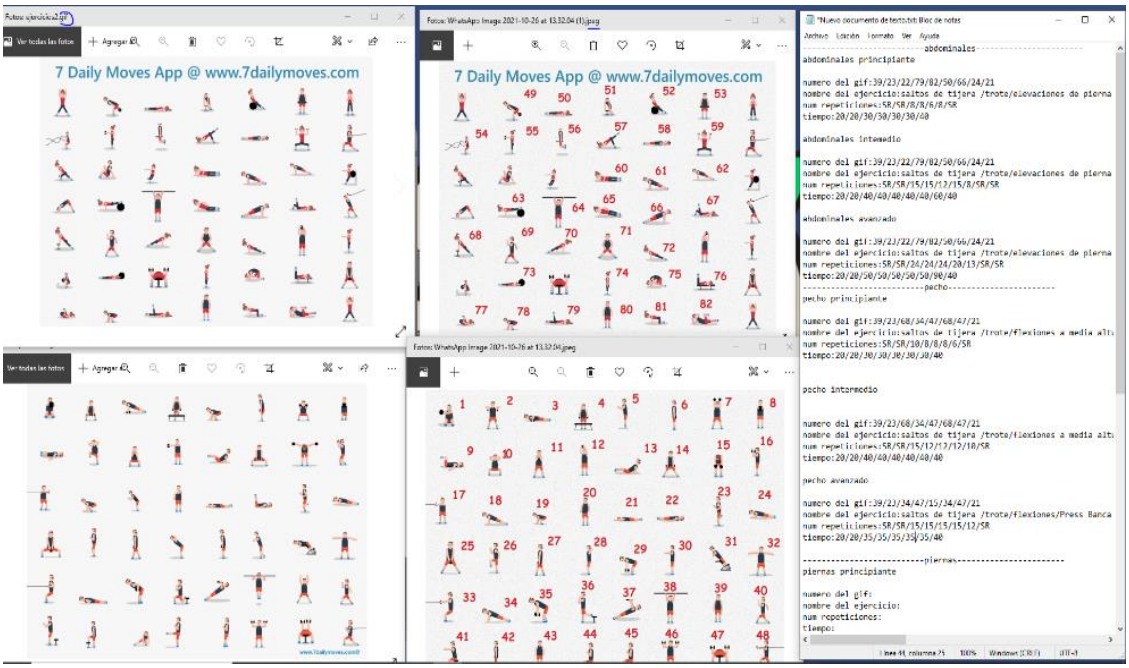
Otra posibilidad era la de añadir un mapeo de imágenes para poder abarcar todas las resoluciones existentes. Actualmente esta aplicación no está mapeada, solo el icono de la aplicación, ya que la mayoría de los elementos están en *constraint layout* que soluciona bastantes problemas de escalado. También, intento usar sobre todo *match_parent* y *wrap_content*. En caso de dar una medida, uso dp y sp que, también, solucionan problemas de escalado porque es “por densidad de píxeles”. El dp se usa para elementos y el sp para texto, además en la mayoría de *acitivities* se permite el deslizamiento hacia abajo, por lo que puedes ver todos los elementos independientemente de la resolución de la pantalla.

He considerado que, aunque se pudiera mapear con herramientas, solo lo haría manualmente y con los recursos necesarios (gifs de diferentes tamaños), ya que este tipo de herramientas destrozan los formatos gifs convirtiéndolos en imágenes estáticas. De este modo podría, en un futuro, adaptarlo para dispositivos muy pequeños o que pueden doblarse. En este momento, la aplicación permite su utilización en todos los móviles actuales del mercado.

Considero también, cambiar las rutinas creadas, debido a que no están respaldadas por la opinión de ningún experto. La aplicación entrena un grupo muscular a través de una rutina previamente seleccionada, sin embargo, al ser una aplicación orientada a un trabajo de fin de grado, se prioriza más la estética que la actividad funcional, por ello, las cantidades varían mucho. Por ejemplo, a mayor dificultad más tiempo, cuando profesionalmente, debería ser al contrario,

muchas repeticiones. En caso de que fuese con peso, se requieren menos repeticiones porque la idea, al aumentar la dificultad, es ganar mayor masa muscular, lo cual se consigue con más peso y menos repeticiones. De forma resumida, para perfeccionar la aplicación sería imprescindible la ayuda de un experto en actividades físicas y rutinas deportivas.

Figura 11. *Creación de las rutinas.*



Concluyendo el proyecto, de cara al futuro, el objetivo sería registrarme como desarrollador para poder subir la aplicación a *Play Store* y poder monetizarla por número de descargas.

6. Anexo Códigos

6.1. Controlador de música

```
public class ControladorMusica {
   public static MediaPlayer;
   public ControladorMusica () {

 }
   public static void empezar (Context context){
      mediaPlayer = MediaPlayer.create(context, R.raw.musicagym);
      mediaPlayer.start();
   }
   public static void parar(){
      if(mediaPlayer!=null) {
         mediaPlayer.stop();
      }
   }
}
```

6.2. Temporizadores de la animación.

```
int tiempoTranscurrir1 = 1000;
Handler handler1 = new Handler ();
handler1.postDelayed(new Runnable () {
   @Override
   public void run () {
      ImagenTitulo1.animate().alpha(1f).setDuration(0);
      handler1.removeCallbacks(null);
   }
}, tiempoTranscurrir1);
```

6.3. Comprobación de la existencia de bases de datos.

```
SQLiteDatabase checkDB = null;
String path="/data/data/com.example.sangredeespartano/databases/bd_abp";
try {
   checkDB = SQLiteDatabase.openDatabase(path, null,
SQLiteDatabase.OPEN_READONLY);
   checkDB.close();

} catch (SQLiteException e) {
   CreacionIncialDeLasBDs.crearBdAbP(this);
   CreacionIncialDeLasBDs.crearBdAbI(this);
```

6.4. Creación de las bases de datos.

```
public static void crearBdAbP(Context context){
    ConexionSQLiteHelperAbP connAbP=new
ConexionSQLiteHelperAbP(context,"bd_abp",null,1);
    SQLiteDatabase dbAbP=connAbP.getWritableDatabase();
    String[]
nombreGif={"gifejercicio39","gifejercicio23","gifejercicio22","gifejercicio79","gifej
ercicio82","gifejercicio50","gifejercicio21"};
    String[] nombreEjercicio={"saltos de tijera","trote","elevaciones de
pierna","elevaciones alternando","Cruch de bicicleta","cruch en V","estiramiento
de cobra"};
    String[] numRepeticiones={"20s","20s","X8","X8","X6","X8","40s"};
    int[] tiempos={20,20,30,30,30,30,40};
    ContentValues values=new ContentValues();

    for(int i=0;i< nombreGif.length;i++) {
        values.put(Utilidades.CAMPO_NOMBRE_GIF, nombreGif[i]);
        values.put(Utilidades.CAMPO_NOMBRE_EJERCICIO, nombreEjercicio[i]);
        values.put(Utilidades.CAMPO_NUM_REPETICIONES, numRepeticiones[i]);
        values.put(Utilidades.CAMPO_TIEMPOS, tiempos[i]);
        dbAbP.insert(Utilidades.TABLA_ABP, Utilidades.CAMPO_NOMBRE_GIF,
values);
        values.clear();
    }
}
```

6.5. Clase que extiende de "ConexionSQLiteHelper".

```
class ConexionSQLiteHelperAbP extends SQLiteOpenHelper {

    public ConexionSQLiteHelperAbP(@Nullable Context, @Nullable String name,
@Nullable SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(Utilidades.CREAR_TABLA_ABP);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int Vantigua, int vnueva) {
        db.execSQL("DROP TABLE IF EXISTS "+Utilidades.TABLA_ABP);
        onCreate(db);
    }
}
```

6.6. Clase Utilidades.

```
public static final String TABLA_ABP="abdominalesprincipiante";
public static final String CREAR_TABLA_ABP="CREATE TABLE
"+TABLA_ABP+" ("+CAMPO_NOMBRE_GIF+" TEXT
,"+CAMPO_NOMBRE_EJERCICIO+" TEXT ,"+CAMPO_NUM_REPETICIONES+"
TEXT ,"+CAMPO_TIEMPOS+" INTEGER)";

public static final String CAMPO_NOMBRE_GIF="nombregif";
public static final String CAMPO_NOMBRE_EJERCICIO="nombreejercicio";
public static final String CAMPO_NUM_REPETICIONES="numrepeticiones";
public static final String CAMPO_TIEMPOS="tiempos";
```

6.7. Método parar de la clase ControladorMusica.

```
        public static void parar () {
   if(mediaPlayer!=null) {
      mediaPlayer.stop();
  }
}
```

1.8. Método del botón “Popup”.

```
Bundle bolsa=new Bundle ();

//  Es PeP porque es PEcho Principiante
   public void botonPopupPeP (View view){

      bolsa.putString("ejercicioElegido","PeP");
      bolsa.putInt("kcal",110);
      Intent ir1=new Intent(this,PopupPreComenzarEjercicio.class);
       ir1.putExtras(bolsa);

           startActivity(ir1);
   }
```

6.8. Cambiar la resolución de la pantalla.

```
DisplayMetrics medidasVentana=new DisplayMetrics();
getWindowManager(). getDefaultDisplay().getMetrics(medidasVentana);

int ancho=medidasVentana.widthPixels;
int alto=medidasVentana.widthPixels;

getWindow().setLayout((int)(ancho*0.85),(int)(alto*1.25));
```

6.9. Estilo Propio y Androd Manifest.

Estilo Propio

```
<style name="Theme.AppCompat. TemaPopup">
   <item name="android:windowIsFloating">true</item>
   <item name="android:windowIsTranslucent">true</item>
   <item name="android:windowCloseOnTouchOutside">true</item>
   <item name="windowNoTitle">true</item>
</style>
```

Android Manifest

```
<activity
   android:name=". PopupPreComenzarEjercicio"
   android:configChanges="orientation"
   android:screenOrientation="portrait"
   android:exported="true"
   android:theme="@style/Theme.AppCompat.TemaPopup" />
```

6.10. ArrayLists contenedores.

```
ArrayList<String> nombresDeGifs=new ArrayList<>();
ArrayList<ImageView> idImagenes=new ArrayList<>();
ArrayList<TextView> idTexto=new ArrayList<>();
ArrayList<String> nombreDeEjercicios=new ArrayList<>();
ArrayList<String> numeroRepeticiones=new ArrayList<>();
ArrayList<Integer> tiempos=new ArrayList<>();

idImagenes.add((ImageView)findViewById(R.id.imgVaciaEjercicio1));…

idTexto.add((TextView) findViewById(R.id.textVacioEjercicio1));…
```

6.11. Distintivo entre Popups.

```
Bundle bolsaR=getIntent().getExtras();
datosQueCargar= bolsaR.getString("ejercicioElegido");
kcaloriasRecuperadas= bolsaR.getInt("kcal");
```

6.12. Switch de obtención de datos.

```
switch (datosQueCargar){
   case "AbP":

imagenEjercicioElegido.setImageResource(R.drawable.imgabdominalesprincipiant
e);
```

```
        ConexionSQLiteHelperAbP connAbP=new
ConexionSQLiteHelperAbP(this,"bd_abp",null,1);
        SQLiteDatabase dbAbP=connAbP.getReadableDatabase();
        Cursor cursorAbP;
        cursorAbP = dbAbP.query(Utilidades.TABLA_ABP, null, null, null, null, null,
null);
        while(cursorAbP.moveToNext()){
        nombresDeGifs.add(cursorAbP.getString(0));
        nombreDeEjercicios.add(cursorAbP.getString(1));
        numeroRepeticiones.add(cursorAbP.getString(2));
        tiempos.add(cursorAbP.getInt(3));
        }
       break;
    case "Abl":…
```

6.13.bucle de cargado Popup.

```
for(int i=0;i<nombresDeGifs.size();i++){
    Drawable =
getResources().getDrawable(getResources().getIdentifier(nombresDeGifs.get(i),
"drawable",
            getPackageName()));

    Glide.with(this).load(drawable).into(idImagenes.get(i));
    idTexto.get(i).setText(nombreDeEjercicios.get(i)+"
"+numeroRepeticiones.get(i));
}
```

6.14. Dependencias Glide.

```
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'

implementation 'com.github.bumptech.glide:glide:4.12.0'
```

6.15. Ejemplo explicado de cómo usar Glide

El contexto es “this”: Glide.*with*(**this**)

El drawable lo genero con este código, y le doy como parámetro el nombre del gif que es un String y lo hago drawable.

```
            Drawable =
getResources().getDrawable(getResources().getIdentifier(nombresDeGifs.get(i),
"drawable",
            getPackageName()));
```

Y lo pongo en. load(drawable)

Y por último en donde se cargará es en el ImageView vacío:

. into(**idImagenes**.get(i));

ejemplo completo:

Glide.*with*(**this**). load(drawable). into(**idImagenes**.get(i));

6.16. Botón comenzar rutina

```
public void botonEmpezarRutinaParaLlegarMostrarRutina(View view){
   bolsa.putStringArrayList("nombresDeGifs",nombresDeGifs);
   bolsa.putStringArrayList("nombreDeEjercicios",nombreDeEjercicios);
   bolsa.putStringArrayList("numeroRepeticiones",numeroRepeticiones);
   bolsa.putIntegerArrayList("tiempos",tiempos);
   bolsa.putInt("kcal",kcaloriasRecuperadas);
   Intent ir1=new Intent(this,MostrarRutina.class);
   ir1.putExtras(bolsa);
   startActivity(ir1);
}
```

6.17. XML del círculo sin progreso

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
   android:shape="ring"
   android:thickness="10dp"
   android:innerRadius="100dp"
   android:useLevel="false"

   >
   <solid android:color="#ED1C24">  </solid>
</shape>
```

6.18. Obtención de un contexto.

```
private static Context;



MostrarRutina.context = getApplicationContext();

public static Context getAppContext() {
   return MostrarRutina.context;
}
```

6.19. Creación de una tabla dinámica.

```
private static Context;
```

```
MostrarRutina.context = getApplicationContext();

public static Context getAppContext() {
   return MostrarRutina.context;
}

TableDynamic tableDynamic=new TableDynamic(tabla,getApplicationContext());
tableDynamic.addHeader(header);
tableDynamic.addData(leerBdHistorial());

private String[]header={"Fecha","Num_Rutina","Kcal","Tiempo(min)"};

private ArrayList<String[]> leerBdHistorial()  {

   ArrayList<String[]> frases = new ArrayList<String[]>();
   ConexionSQLiteHelperHistorial connHistorial=new
ConexionSQLiteHelperHistorial(this,"bd_historial",null,1);
   SQLiteDatabase dbHistorial=connHistorial.getReadableDatabase();
   Cursor;
      cursor = dbHistorial.query(Utilidades.TABLA_HISTORIAL, null, null, null, null,
null, null);

   while(cursor.moveToNext()){
      frases.add(0,new
String[]{cursor.getString(0),cursor.getString(1),cursor.getString(2),cursor.getString(3)});

   }
   return frases;
}
```

6.20. Fijar orientación.

```
android:configChanges="orientation"

android:screenOrientation="portrait"
```

6.21. Mantener pantalla encendida.

```
android:keepScreenOn="true"

getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

6.22. Seleccionar color del StatusBar y NavegatioBar.

```
getWindow().setStatusBarColor(Color.parseColor("#ED1C24"));
getWindow().setNavigationBarColor(Color.parseColor("#ED1C24"));
```

7. Webgrafía

https://stackoverflow.com/questions/9948105/android-how-to-iterate-an-r-drawable-object //obtener un drawable

https://www.youtube.com/watch?v=nX9E5k0xhLQ // creación de un “Popup”, ventanas emergentes de ayuda

https://www.youtube.com/watch?v=khXOSsExgeA //Glide gifs

https://www.youtube.com/watch?v=Z_lUC3TVjBI //cuenta atrás

https://www.flipandroid.com/ajustar-imagen-en-imagebutton-en-android.html //para centrar los botones con imágenes

https://es.stackoverflow.com/questions/161742/rotar-una-imagen-en-android-studio //rotación animación

https://www.youtube.com/watch?v=TjJ3cHcPWw8 // y sus otros 5 o 6 videos de SQLite

https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es // para recordar el ciclo de vida

https://www.youtube.com/watch?v=GCrgW6ROYu0&t=343s // cronómetro redondo

https://www.youtube.com/watch?v=IAfGGzZoMwk //Usar objeto MediaPlayer

https://www.it-swarm-es.com/es/android/android-media-player-se-reproduce-en-segundo-plano-pero-no-se-detiene-cuando-se-apaga-la-aplicacion/1042003700/ // parar música desde otros activities

https://www.youtube.com/watch?v=g9W4f6YSGRM // cronómetro

https://www.youtube.com/watch?v=rSafkhA6TZA // tabla dinámica

https://es.stackoverflow.com/questions/66787/verificar-existencia-bd-y-tabla-en-androi //verificar base de datos

https://es.stackoverflow.com/questions/10267/c%C3%B3mo-obtener-context-dentro-de-un-m%C3%A9todo-static-en-android // obtener contexto

https://developer.android.com/training/scheduling/wakelock?hl=es-419 // hacer que no se apague la pantalla

https://stackoverflow.com/questions/1683185/how-do-i-create-a-listview-with-rounded-corners-in-android // borde redondo

https://es.stackoverflow.com/questions/46871/cambiar-color-statusbar // cambiar color *status* y *navegation*

Enlace github: https://github.com/franandujarc/TFG_Francisco_Andujar