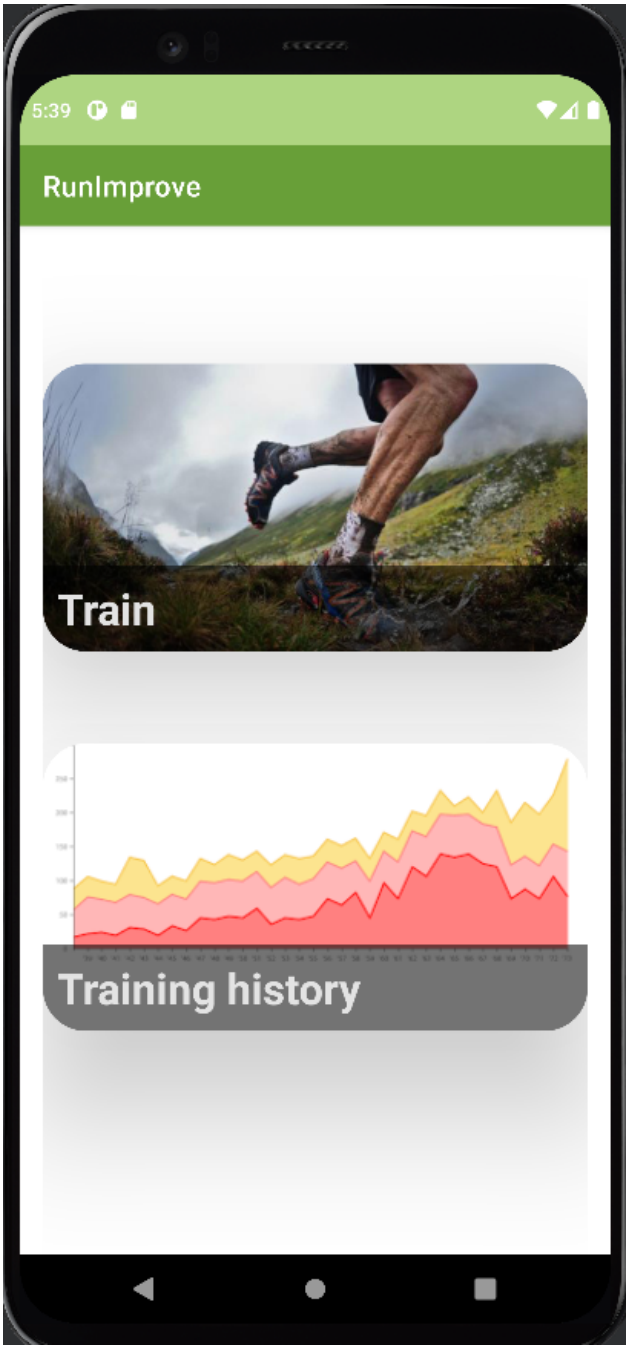
RunImprove

Alejandro Escalona García



Índice

Introducción

La app de RunImprove está pensada para gente amateur aficionada a correr y más en concreto a realizar trail running o correr por montaña. Para aquellos que no tienen un entrenamiento muy planificado o que no saben muy bien cómo orientar su entrenamiento entre las diferentes variedades del mismo que se puede realizar.

A continuación explicaré las diferentes funcionalidades que se pueden ejecutar.

Estructura del sistema

1. Requisitos funcionales

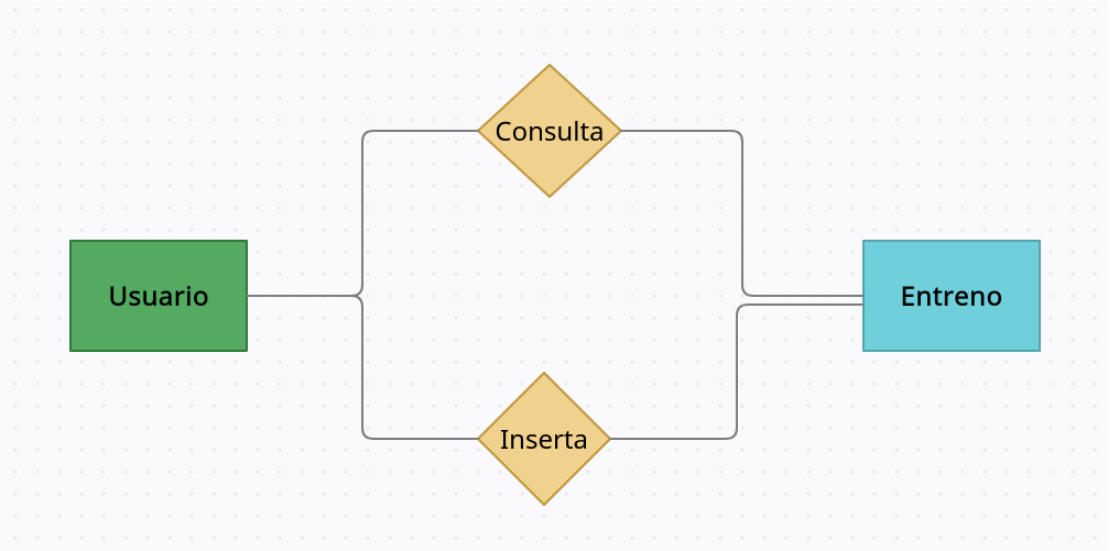
- RF1. Selección del tipo entrenamiento
- RF2. Inicio del ejercicio
- RF3. Realización del descanso con un cronómetro.
- RF4. Aviso al usuario de finalización del descanso mediante una alarma.
- RF5. Opción de guardar el entrenamiento y su progreso.
- RF6. Consulta del historial de entrenamientos.
- RF7. Borrado de uno o todos los entrenamientos.
- RF8. Visualización de porcentajes de entrenamientos por tipo mediante una gráfica.

2. Requisitos no funcionales

- Requisitos hardware y software: Dispositivo Android.
- Requisitos de interfaz:
    - Diseño y uso sencillo.
    - Tamaño adecuado de texto y botones.
    - Diseño responsive para diferentes dispositivos.

3. Diagramas

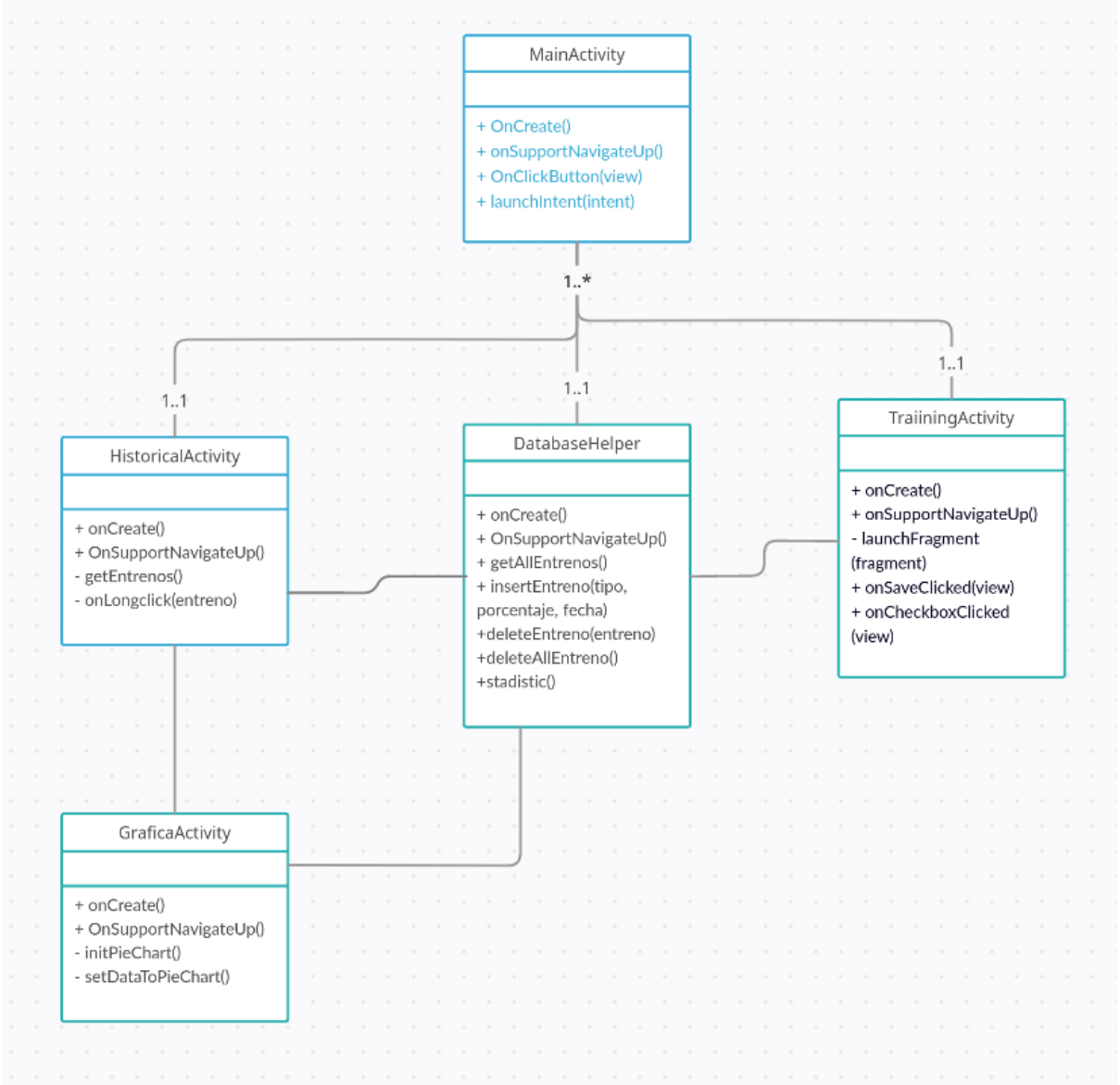
Modelo Entidad Relación (BD)



Modelo Relacional (BD)

Entrenamiento
Id : Long
Tipo : String
Porcentaje : Double
Fecha : String

Diagrama de Clases



Implementación

Herramientas

- **Android Studio:** Para la creación e implementación de la aplicación he utilizado el ide de android studio, tal y como recomiendan desde Google.
- **Lenguaje:** En cuanto al lenguaje de programación he utilizado Kotlin, ya que ofrece diferentes ventajas sobre Java, y es hacía se orienta el futuro de Android.
- **Base de datos:** He utilizado SQLite que es una base de datos de software libre y es una manera de almacenar información de forma rápida y eficaz.

Código

En cuanto al código fuente de la aplicación, he utilizado activities para la parte de la home, el historial y la selección del tipo de entrenamiento y la gráfica. Después para mostrar cada tipo de entrenamiento he utilizado Fragment.

Para desplazarnos entre todos ellos he utilizado los intent, que los pasaremos a la propia función de StartActivity().

Para la gráfica he utilizado una librería externa, la cual tuve que realizar el siguiente import:

```
import com.github.mikephil.charting.charts.PieChart
```

Para poder utilizarlo, además de incluirlo en las dependencias del gradle. Una vez importado, usamos la clase PieChart y todos los métodos que incluye para setearlo.

Lo que hacemos en la activity localizamos componente en la vista que sin setearlo está vacío, y ejecutamos la función de iniciar el gráfico, en la que configuramos tamaño de las etiquetas, la rotación y demás configuraciones.

```
<com.github.mikephil.charting.charts.PieChart
    android:id="@+id/PieChart"
```

Después ejecutamos la función la función de introducir los datos, que vamos obteniendo de la base de datos, con la función de stadistic que implementé en la clase de DataBaseHelper. Y le vamos insertando los colores y la forma del gráfico incluida el agujero de en medio de la gráfica.

```
private fun initPieChart() {
    pieChart.setUsePercentValues(true)
    pieChart.description.text = ""
    //hollow pie chart
    pieChart.isDrawHoleEnabled = false
    pieChart.setTouchEnabled(false)
    pieChart.setDrawEntryLabels(false)
    //adding padding
    pieChart.setExtraOffsets( left: 20f,  top: 0f,  right: 20f,  bottom: 0f)
    pieChart.setUsePercentValues(true)
    pieChart.isRotationEnabled = false
    pieChart.legend.orientation = Legend.LegendOrientation.HORIZONTAL
    pieChart.legend.isWordWrapEnabled = true
    pieChart.legend.textSize = 36f
    pieChart.legend.formSize = 30f
}
/**
 * @author Alejandro Escalona García
 * Rellena de datos el gráfico
 */
private fun setDataToPieChart() {
    val datos = database.stadistic()
    pieChart.setUsePercentValues(true)
    val dataEntries = ArrayList<PieEntry>()
    dataEntries.add(PieEntry(datos[0].toFloat(), "Sprints"))
    dataEntries.add(PieEntry(datos[1].toFloat(), "Hit"))
    dataEntries.add(PieEntry(datos[2].toFloat(), "Bajadas"))
```

En todas las vistas de la app para aumentar la accesibilidad en todos los dispositivos, aparece la flecha en la parte superior de la app para poder volver a la vista anterior, ya que aunque los usuarios están acostumbrados a la flecha de la parte inferior para volver atrás me parece que puede facilitar el flujo entre vistas.

A la hora de que suene la alarma al acabar el cronómetro de los descansos he utilizado un `CountDown Timer,` a este objeto se le pasa cuanto tiempo en total tiene que hacer la cuenta atrás y cada cuanto es el intervalo de tiempo que resta. Después pusé que se cambiará el número en el textView y que al acabar sonase la alarma con la clase

RingToneManager y que al darle al botón de parar la alarma se detuviera, que se muestra en un modal en el medio de la pantalla.

```
mBinding.btnPlay.setOnClickListener {  it: View!
    object : CountDownTimer( millisInFuture: 10000,  countDownInterval: 1000) {
        @SuppressLint( ...value: "SetTextI18n")
        override fun onTick(millisUntilFinished: Long) {
            mBinding.tvCuentaAtras.setText("" + millisUntilFinished / 1000)
            mBinding.btnPause.setOnClickListener {  it: View!
                mBinding.tvCuentaAtras.setText("")
                cancel()
            }
        }
        override fun onFinish() {
            val notificacion =
                RingtoneManager.getDefaultUri(RingtoneManager.TYPE_RINGTONE)
            val r = RingtoneManager.getRingtone(activity, notificacion)
            r.play()
            this.cancel()
            val ab = MaterialAlertDialogBuilder(requireContext())
                .setTitle(getString(R.string.quieres_parar_alarma))
                .setCancelable(true)
                .setPositiveButton(getString(R.string.parar_alarma)) { dialogInterface, i ->
                    r.stop()
                }
            ab.create().show()
            mBinding.btnStopMusic.setOnClickListener {  it: View!
                r.stop()
            }
        }
```

Para guardar todos los entrenamientos que los usuarios van realizando, he creado una clase llamada DataBaseHelper que es la encargada de la gestión de la base de datos, sobrescribiendo métodos como el onCreate y creando otros como inserEntreno, getEntreno o deleteAll. En ellos he usado Constantes para aumentar la facilidad de gestión y mantenimiento del código. De esta forma solo se pueden realizar acciones a través de esta clase y no se realizan cosas que no se deba.

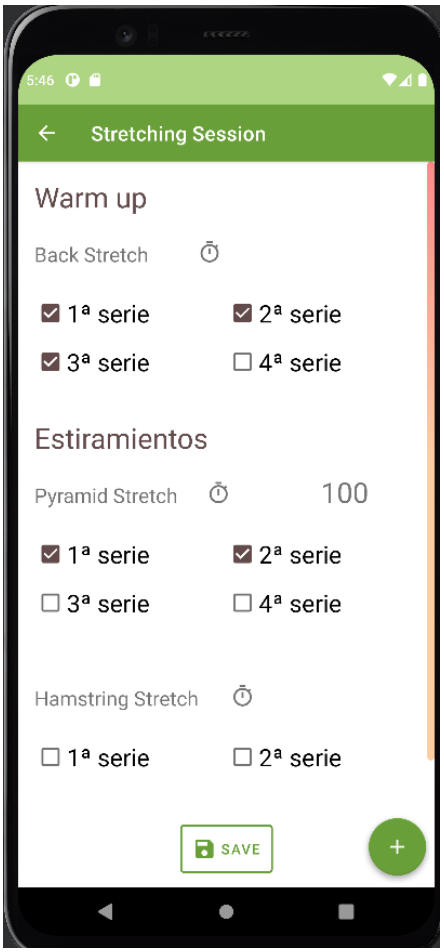
```
class DatabaseHelper(context: Context) : SQLiteOpenHelper
    (context, Constants.DATABASE_NAME,  factory: null, Constants.DATABASE_VERSION) {

    /**
     * @author Alejandro Escalona García
     * @constructor Crea una base de datos.
     */
    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE ${Constants.ENTRENOS} " +
                "(${Constants.PROPERTY_ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "${Constants.PROPERTY_TIPO} VARCHAR(45)," +
                "${Constants.PROPERTY_PORCENTAJE} REAL," +
                "${Constants.PROPERTY_DATE} VARCHAR(20))"
        db?.execSQL(createTable)
    }
```

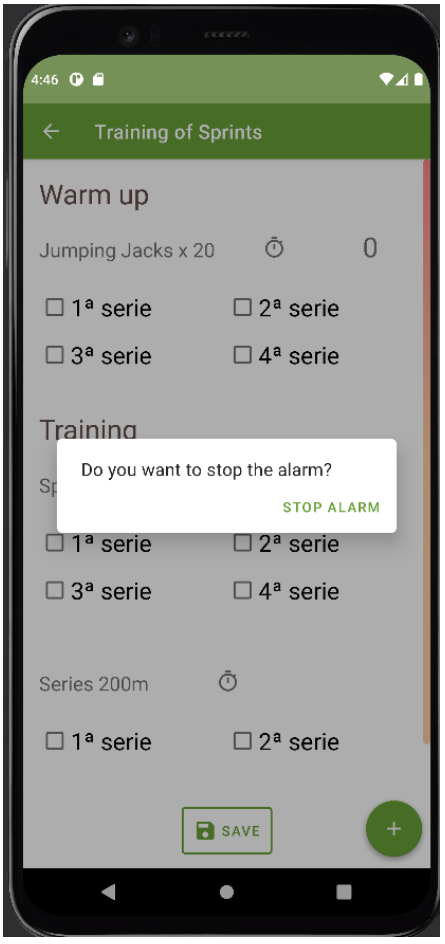
Funcionalidades

4. Marcador de series & Cronometro

En la app se va a poder marcar las series de cada tipo de entrenamiento conforme se va realizando, de forma que no te pierdes contando, ya que cuando estás concentrado en el ejercicio, no son pocas las ocasiones que contamos de más o de menos el número de series que hemos realizado. Además que según el porcentaje de series que hayamos completado de cada serie se quedará registrado en la base de datos que mostraremos más adelante.

También una vez que hemos acabado cada serie, podremos darle a un “cronómetro” que nos indicará cuántos segundos nos quedan de descanso y que si se nos olvida mirar luego sonará una alarma cuando acabe el descanso, que sonará aunque bloqueemos el dispositivo móvil.

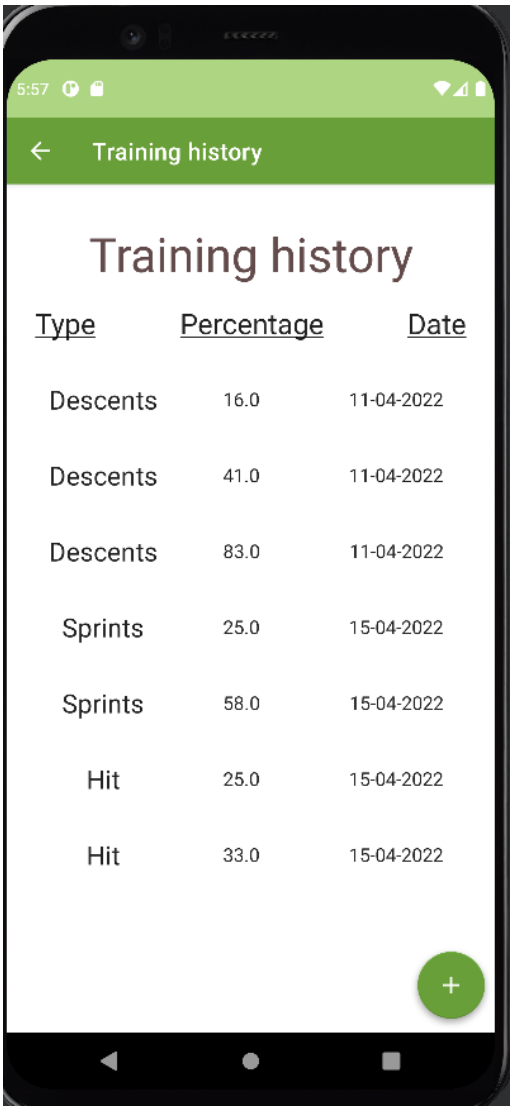


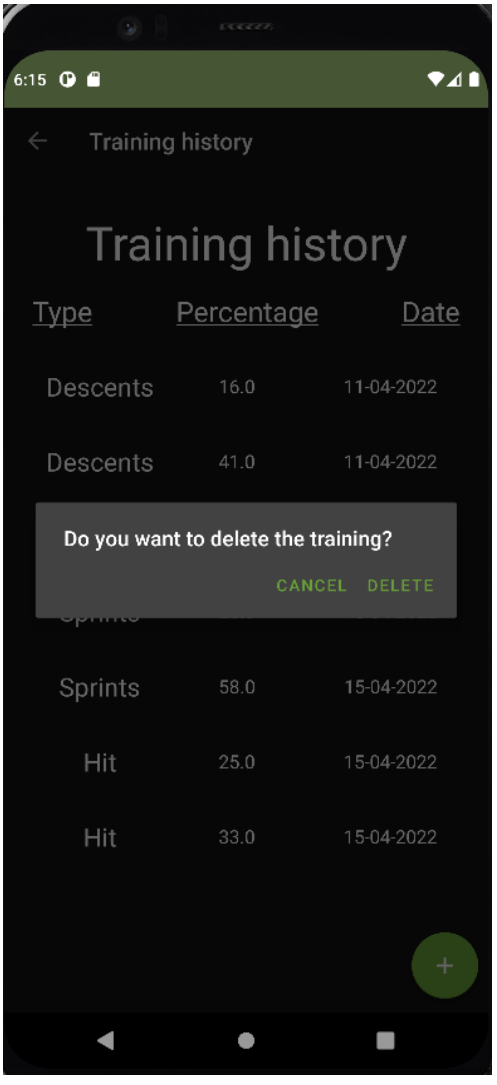


5. Registro de entrenamientos

En la app podremos ir guardando cada entrenamiento que realicemos dándole al botón de abajo que pondrá guardar o save, depende del idioma en el que tengamos el dispositivo móvil. Una vez que lo hayamos guardado, nos podremos dirigir al apartado de historial de entrenamiento, y en él nos saldrá la fecha del entrenamiento, el tipo de entrenamiento y en qué porcentaje lo hemos completado, que lo ideal sería el 100%, pero en muchas ocasiones no lo acabamos.

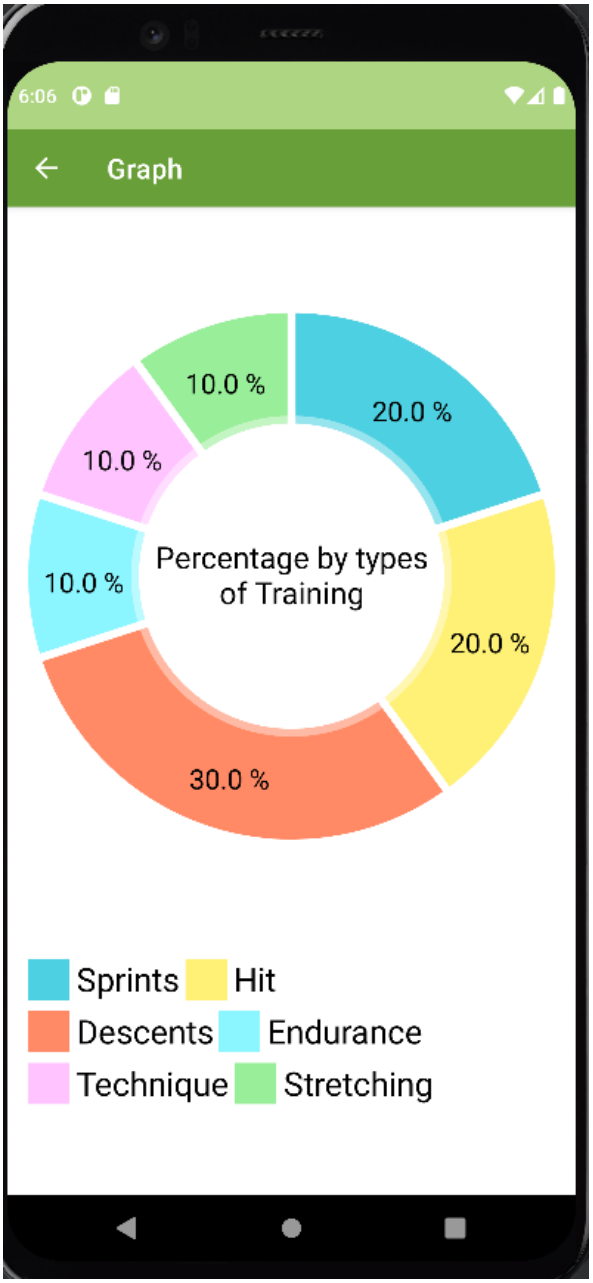
También se podrán borrar todos los entrenamientos, o si el usuario lo prefiere manteniendo pulsado sobre un entrenamiento le aparecerá un mensaje de confirmación para borrar un entrenamiento si lo desea.





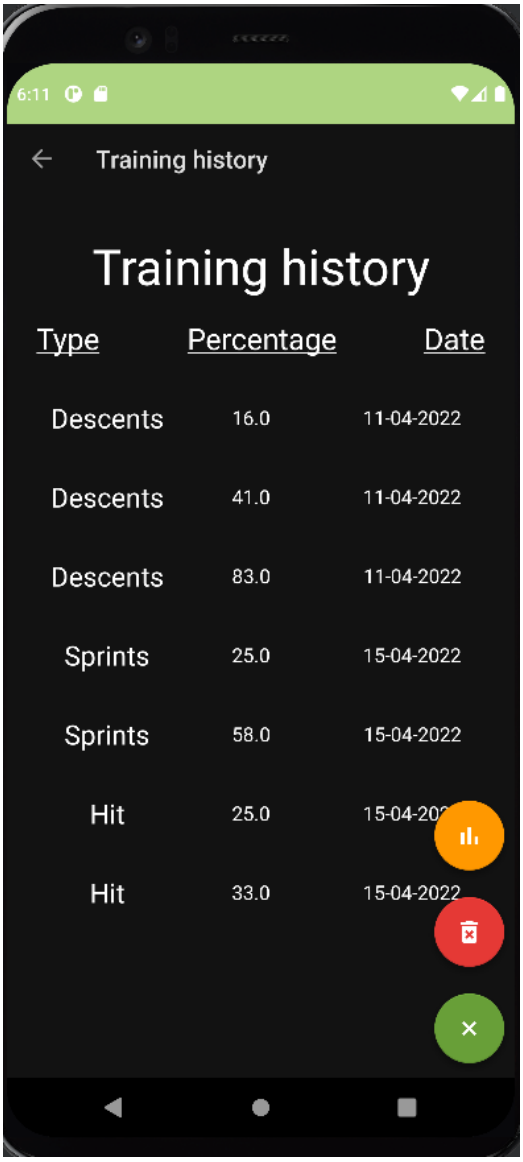
6. Gráfica por tipo de entrenamiento

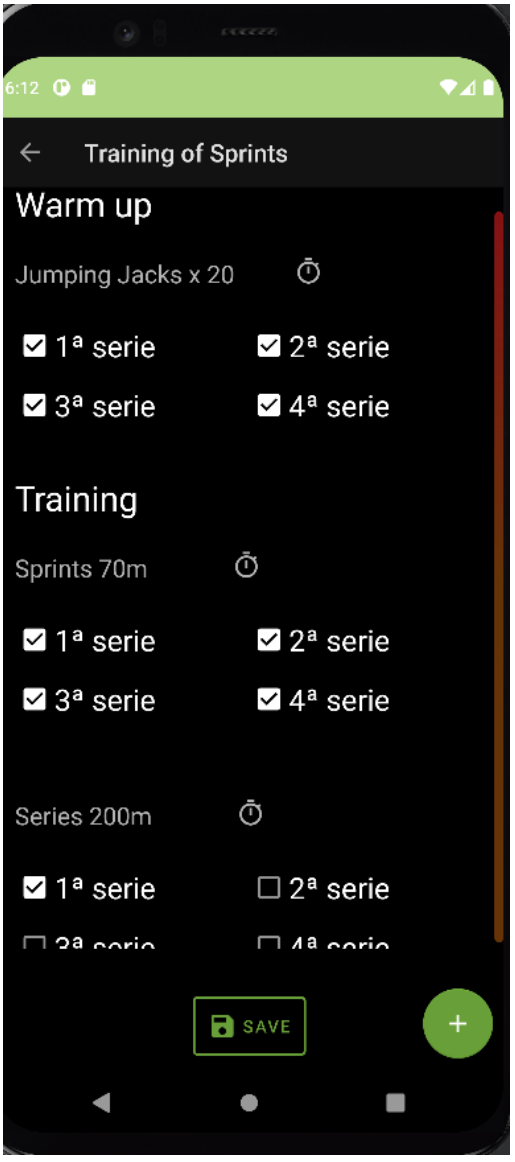
En la app vamos a poder visualizar fácilmente que tipo de entrenamiento es el que hacemos más por si alguno de los entrenamientos lo hacemos menos y se nos olvida o ver cuál nos gusta más. En definitiva podemos ver en qué porcentaje realizamos cada entrenamiento.



7. App adaptada al modo noche

Toda la app se podrá ver si el usuario lo prefiere en modo oscuro, por lo que cada pantalla se adaptará al modo que prefiera el usuario. Esto se hace a través de la elección de los colores según el modo que tiene el usuario.





8. App multilenguaje

Está app determina el idioma en el que el usuario tiene su dispositivo móvil y se pondrá en Español o inglés dependiendo de cual sea automáticamente, tiene todos los textos traducidos. Esto lo he realizado a través de tener dos ficheros strings.xml uno para el español y otro para el inglés.