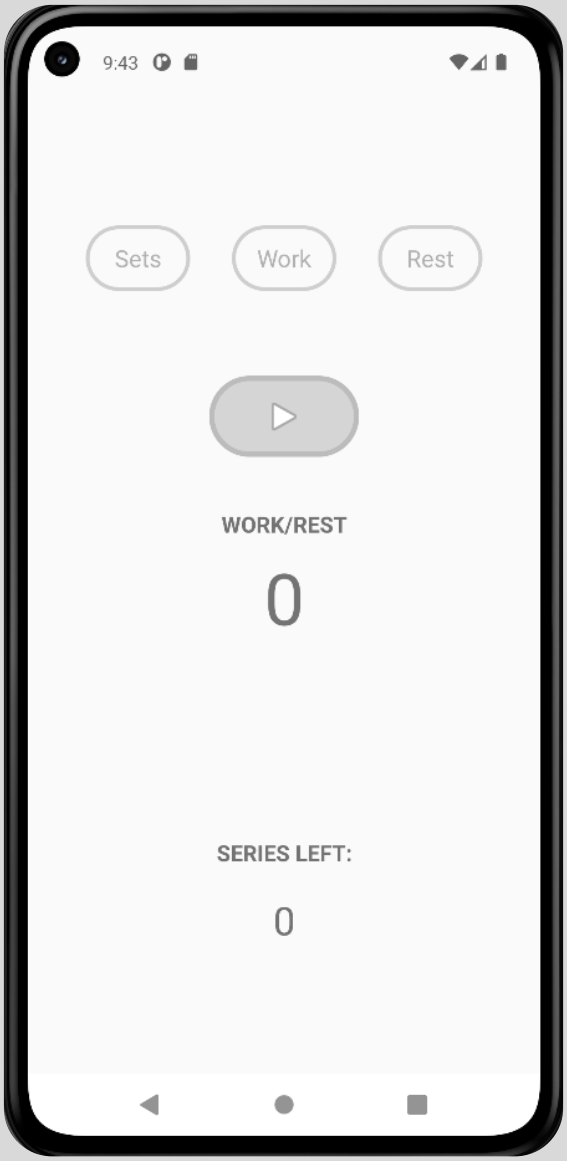


TABATA

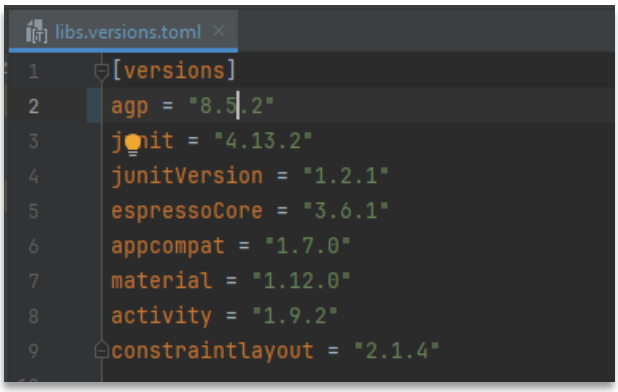
Alfonso Rincón Cuerva
2º DAM
Programación Multimedia y de Dispositivos Móviles

ÍNDICE

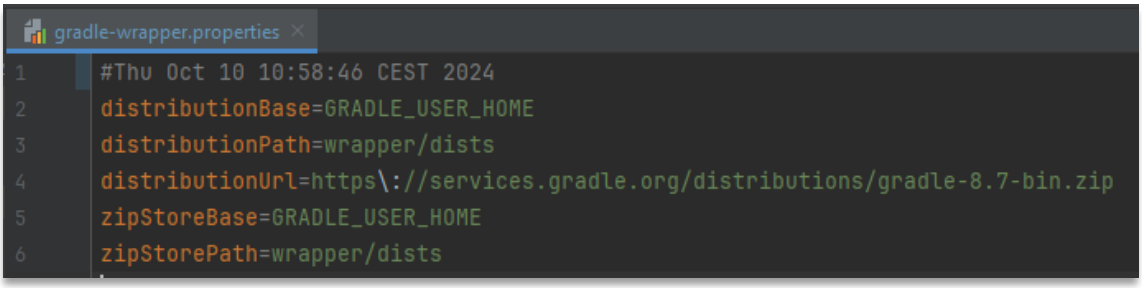
HERRAMIENTAS Y TECNOLOGÍAS

VERSIONES

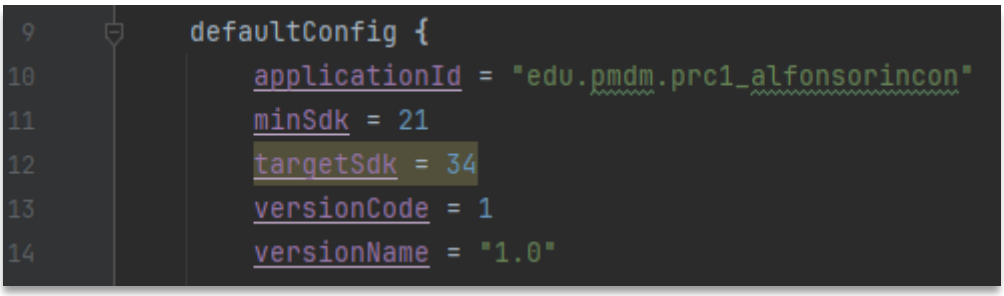
En el archivo **libs.versions.toml** encontramos versiones de diferentes bibliotecas y plugins del proyecto.



La versión de Gradle es de 8.7



- **minSdk**: en esta aplicación es de 21. Esto significa que la versión de API mínima que soporta la app es la 21.
- **targfetSdk**: es la versión de SDK para la cual está optimizada la app. En esta app, es de 34.



EMULADORES

- **Pixel API 30**: 5,0'' 1080x1920 420dpi. ***System image***: R (API 30)
- **Pixel 5**: 6,0'' 1080x2340 440dpi. ***System image***: R (API 30)

PLATAFORMA

El programa fue creado en la plataforma **Android Studio**, que es un entorno de desarrollo integrado oficial para crear aplicaciones Android. Incluye características como

editor de código, diseñador de interfaces, emulador de Android y sistema de compilación Gradle.

LENGUAJES

Los lenguajes utilizados para la creación de esta aplicación son los siguientes:

- **Java**: para la lógica de la aplicación e interactuar con los usuarios.
- **XML**: para agregar características y estilos a la aplicación.
- **Gradle**: es el sistema de compilación que utiliza Android.

PROBLEMA PLANTEADO

Se nos pide crear un programa que simule un cronometro Tabata para el entrenamiento. En él, el usuario deberá insertar el número de series a realizar, el tiempo de trabajo y el tiempo de descanso, y pulsar en un botón para que el entrenamiento comience.

Dependiendo del estado del programa, la interfaz tendrá un estado u otro.

- Si el tiempo de trabajo está corriendo, el fondo de pantalla deberá ser verde y tendrá un texto de **WORK**.
- Si el tiempo de descanso es el que está corriendo, el fondo será rojo, y el texto dirá **REST**.
- En el caso de que haya acabado el entrenamiento, el color volverá a su estado inicial, y el texto dirá **FINISHED**.

Además de esto, debe de indicar el número de series restantes y emitir sonidos.

SOLUCIÓN AL PROBLEMA

Para resolver este programa, he creado un método para llevar el tiempo de ejercicio. Una vez que el usuario haya indicado el entrenamiento que quiere realizar y pulse el botón, se ejecutará este método. Este método también pondrá el fondo de color verde.

El método de ejercicio estará formado por un contador, y una vez que este llegue a su fin, irá al método de descanso, y se le pasarán el tiempo, series y descanso como parámetros. Este método es similar al anterior, y tendrá un condicional para hacer una cosa según cada caso:

- Si hay series restantes, se reproducirá el tiempo de descanso, y una vez terminado, se volverá a ejecutar el método de ejercicio. El color de fondo se pondrá en rojo. Esto será así hasta que no haya más series restantes.
- El otro caso, es en el cual ya no hay series restantes. Si esto sucede, se restaurará el color inicial, y se pondrá a 0 todo. Con esto, habrá acabado el bucle, ya que no se volverá a llamar al método de contador.

FUNCIONES UTILIZADAS

ICONO: en el archivo AndroidManifest.xml, asignaré un nuevo icono para la aplicación, el cual es un diseño de un cronometro.

```
android:icon="@mipmap/icon_crono"
android:label="PRC1_AlfonsoRincon"
android:roundIcon="@mipmap/icon_crono"
```

VARIABLES PARA COLORES: en la carpeta ***values*** encontramos el archivo ***colors.xml*.** En este creo variables para todos los colores que usaré para cambiar el color de fondo y de botón, en cada uno de los estados.

```
<color name="black">#FF000000</color>
<color name="white">#FFFFFFFF</color>
<color name="background_green">#90FF6E</color>
<color name="text_green">#2D910E</color>
<color name="btnBackground_green">#2D910E</color>
<color name="button_green">#1D6507</color>
<color name="background_red">#FF816E</color>
<color name="text_red">#B12D19</color>
<color name="btnBackground_red">#B12D19</color>
<color name="button_red">#9E1F0C</color>
<color name="background_grey">#FAFAFA</color>
<color name="text_grey">#737373</color>
<color name="button_grey">#BCBCBC</color>
<color name="btnBackground_grey">#D5D5D5</color>
```

VARIABLES DE STRINGS: en la carpeta **values** también encontramos el archivo **strings.xml.** Aquí creo diferentes variables para cada String que usaré posteriormente en diálogos.

```
<string name="dialog_text_highValues">Los valores introducidos son muy grandes. El número máximo de Sets es 100, y
<string name="dialog_title_invalidValues">NÚMEROS INVÁLIDOS</string>
<string name="dialog_text_invalidValues">Uno o más números introducidos no son válidos. Deben ser mayores de 0</st
<string name="dialog_title_wait">ESPERE</string>
<string name="dialog_text_wait">ESPERE A QUE EL PROGRAMA TERMINE</string>
<string name="dialog_title_empty">HAY CAMPOS SIN RELLENAR</string>
<string name="dialog_text_empty">Debe rellenar todos los campos pra comenzar el programa</string>
```

ALERTAS: utilizando **AlertDialog**, se mostrarán alertas en cada caso de prueba. Esto se hará en un método, para poder usarlo varias veces, y se le pasa como parámetros los textos que tendrá de título y descripción.

.setTitle: lo usamos para ponerle un título al dialog.

.setMessage: lo usamos para ponerle una descripción al dialog.

.setPositiveButton: botón de Aceptar.

Usamos **.show()** para mostrar el diálogo, y **dismiss()** para cerrarlo al pulsar en Aceptar.

```
public void mostrarAlerta(int title, int text) {  4 usages  alfonsaco
    // Se mostrará un alerta, en caso de que haya campos sin rellenar
    AlertDialog alerta=new AlertDialog.Builder( context: MainActivity.this)
            .setTitle(title)
            .setMessage(text)
            .setPositiveButton( text: "Aceptar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {  alfonsaco
                    // Acción al presionar el botón "Aceptar"
                    dialog.dismiss(); // Cerrar el diálogo
                }
            })
            .show();
}
```

```
if(Integer.parseInt(etxtRest.getText().toString())>1000 || Integer.parseInt(etxtSets.getText().toString())>100 || Integer.parseInt(etxtWork.getText(
    mostrarAlerta("VALORES DEMASIADO GRANDES", "Los valores introducidos son muy grandes. El número máx...");
```

MÉTODO DE EJERCICIO: este método servirá para medir el tiempo de entrenamiento. Recibirá como parámetros el número de series, tiempo de descanso y tiempo de entrenamiento.

Utiliza el método cambiarColor con variables del XML de colors.xml para cambiar el color del fondo, fuentes y del botón.

Tras esto, se ejecuta un CountDownTimer para medir el tiempo del entrenamiento, y mostrar por pantalla cada 1 segundo, el tiempo que queda. Una vez terminado, llama al método de descanso, pasando los mismos parámetros.

```
private void exerciseCounter(int work, int rest, int sets) {  2 usages  alfonsaco
    cambiarColor(R.color.background_green, R.color.text_green,  value: "green");

    // Tiempo en milisegundos
    int totalFinal=work*1000;
    timer=new CountDownTimer(totalFinal,  countDownInterval: 1000) {  alfonsaco
        @Override  no usages  alfonsaco
        public void onTick(long l) {
            txtSecondsLeft.setText(String.valueOf( l: l/1000));
            txtSeriesLeft.setText(String.valueOf( i: sets-1));
            txtState.setText("WORK");
        }

        @Override  no usages  alfonsaco
        public void onFinish() {
            // Reproducir el sonido beep, siempre que no se esté en la última serie
            if((sets-1) != 0) {
                playSoundBeep();
            }
            restCounter(work, rest, sets);
        }
    }.start();
}
```

```
exerciseCounter(work, rest, sets);
```

CAMBIAR COLOR: con este método, cambiaremos el color de diferentes elementos de la aplicación.

```
int btnStroke=getResources().getIdentifier( name: "button_"+value, defType: "color",getPackageName());
int btnBackground=getResources().getIdentifier( name: "btnBackground_"+value, defType: "color", getPackageName());
```

A pesar de que no se recomienda utilizar ***.getIdentifier()*** para el nombre del ID del elemento, en este caso lo utilizaremos ya que es la única forma de poder hacer esto, sin tener que escribir un método para cada color. Esto es porque Android Studio no nos deja poner, por ejemplo, ***'R.id.btnBackground_'+color***, utilizando una variable para completar el texto, ya que R.id es un int no un String.

```
private void cambiarColor(int bakgroundColor, int textColor, String value) {  3 usages  alfonsaco
    constraintLayout.setBackgroundColor(ContextCompat.getColor( context: this, bakgroundColor));

    // Obtenemos el valor del color para usarlo en los textos
    int color=ContextCompat.getColor( context: this, textColor);
    txtSecondsLeft.setTextColor(color);
    txtState.setTextColor(color);
    txtSeriesLeft.setTextColor(color);
    txtSeriesLeftMessage.setTextColor(color);
    etxtRest.setTextColor(color);
    etxtSets.setTextColor(color);
    etxtWork.setTextColor(color);

    // Variables para el botón. Aunque la función no sea muy recomendada, es la única manera de poder utilizar las variables del método
    // y cambiar los colores del .xml
    int btnStroke=getResources().getIdentifier( name: "button_"+value, defType: "color",getPackageName());
    int btnBackground=getResources().getIdentifier( name: "btnBackground_"+value, defType: "color", getPackageName());

    GradientDrawable btnColours=(GradientDrawable)btnPlay.getBackground();
    btnColours.setColor(ContextCompat.getColor( context: this, btnBackground));
    // El 10 es equivalente al grosor del borde del botón
    btnColours.setStroke( width: 10,ContextCompat.getColor( context: this, btnStroke));
}
```

```
cambiarColor(R.color.background_green, R.color.text_green, value: "green");
```

REPRODUCIR SONIDO: se crean 2 métodos para reproducir sonido. Utilizamos la clase ***MediaPlayer***, con la cual podemos obtener un sonido y reproducirlo.

Para ello, se crea la carpeta **raw,** dentro de la carpeta de recursos.

```
public void playSoundBeep() {  2 usages  alfonsaco
    MediaPlayer sound=MediaPlayer.create(
            getApplicationContext(), R.raw.beep);
    sound.start();
}
```

MÉTODO DE DESCANSO: este método se ejecuta tras el método de ejercicio, siempre y cuando, haya series restantes. También utiliza el método de cambiar color, y un ***CountDownTimer***.

Al principio del método, modifica sets para que se reste 1 serie cada vez que se complete el ejercicio. Como hemos visto anteriormente, tendrá 2 posibles casos. Una vez se haya completado el programa, sonará un Gong, y se habrá terminado el bucle.

```
// Función para el contador de descanso
private void restCounter(int work, int rest, int sets) {  1 usage  alfonsaco
    // Contador para determinar las series restantes
    int cont=sets;
    cont--;
    int finalCont = cont;

    if(finalCont > 0) {
        cambiarColor(R.color.background_red, R.color.text_red,  value: "red");

        timer=new CountDownTimer( millisInFuture: rest*1000,  countDownInterval: 1000) {  alfonsaco
            @Override  no usages  alfonsaco
            public void onTick(long l) {
                txtSecondsLeft.setText(String.valueOf( l: l/1000));
                txtState.setText("REST");
            }

            @Override  no usages  alfonsaco
            public void onFinish() {
                playSoundBeep();
                exerciseCounter(work, rest, finalCont);
            }
        }.start();
    } else {
        // Restaurar colores iniciales
        cambiarColor(R.color.background_grey, R.color.text_grey,  value: "grey");
        // Cambiar el texto de los TextView
        txtSeriesLeft.setText(String.valueOf(finalCont));
        txtSecondsLeft.setText(String.valueOf(finalCont));
        txtState.setText("FINISHED");
        // Ponemos el boolean a true, para que podamos volver a ejecutar la función
        state=true;
        playSoundGong();
    }
}
```

PRUEBAS REALIZADAS

Se realizan diversas pruebas de caja negra para verificar la funcionalidad del programa. Tras ello, se programan sus respectivas soluciones:

1. **EVITAR VALORES MENORES DE 0**
   Para evitar que el usuario introduzca valores o iguales a 0, para evitar series negativas o entrenamientos sin sentido, se agregan condicionales. En el caso de que el usuario introduzca estos valores, se lanzará un diálogo que nos informará de que esto no es posible.

2. **EVITAR VALORES MUY ELEVADOS**
   El programa no permitirá introducir valores demasiado elevados, como entrenamientos de más de 100 series, o series muy largas, con una duración elevada, ya que un entrenamiento así no tendría sentido.
   Para solucionar esto, se agrega una condicional que desplegará un diálogo en caso de que se ponga un valor de este tipo.

3. **CAMPOS VACÍOS**
   El programa no debería empezar si hay campos vacíos. Para evitar esto, se crean condicionales, que desplegarán un diálogo en dicho caso, hasta que el usuario introduzca todos los valores necesarios.

4. **EVITAR MÚLTIPLES CONTADORES**
   Si pulsases varias veces en el botón, se ejecutarían varios contadores a la vez, por tanto, el programa empezaría a actuar de forma extraña y de forma incoherente.
   Para evitar este problema, se crea un booleano, que solo permita ejecutar el contador cuando este esté en true. Al empezara el programa, se pondrá en false, hasta que se haya terminado, donde se volverá a poner en true.

5. **DISTINTAS RESOLUCIONES:**
   Se ha probado la aplicación en pantallas con diferentes resoluciones, además de en dispositivos móviles reales.

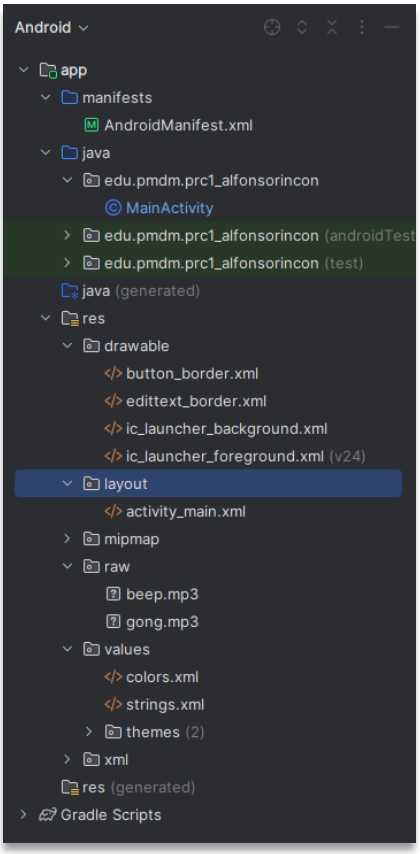
ESTRUCTURA DE LA APLICACIÓN

drawable: aquí están los dos XML que he creado para darles estilos a los elementos. Uno es para el borde del botón, otro es para los bordes de los tres EditText.

mipmap: dentro de esta carpeta están los archivos del icono de la aplicación.

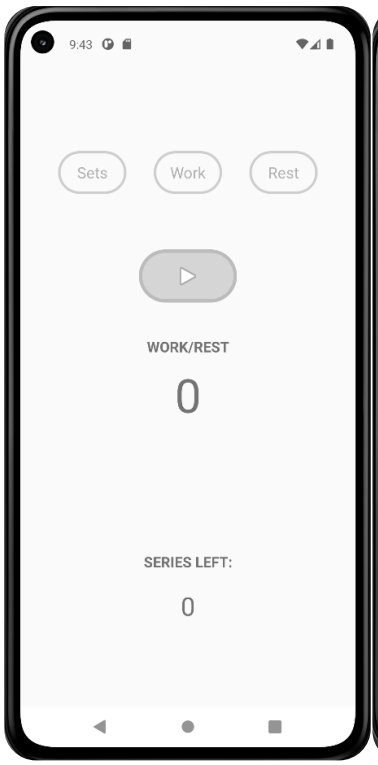
raw: esta carpeta contiene los recursos de audio que utilizaremos en la aplicación. Cuando se ejecutan los métodos de reproducir audio, es a esta carpeta a la que accede para reproducirlos.

values: en sus archivos ***colors.xml*** y ***string.xml*** se encontrarán todas las variables de colores y strings que hemos creado.

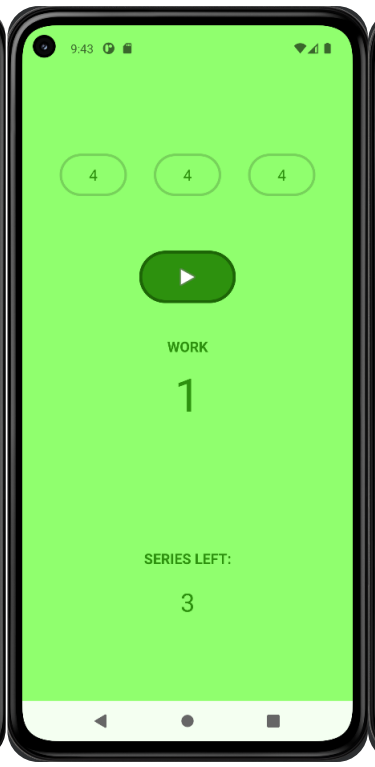


EJECUCIÓN DE LA APLICACIÓN

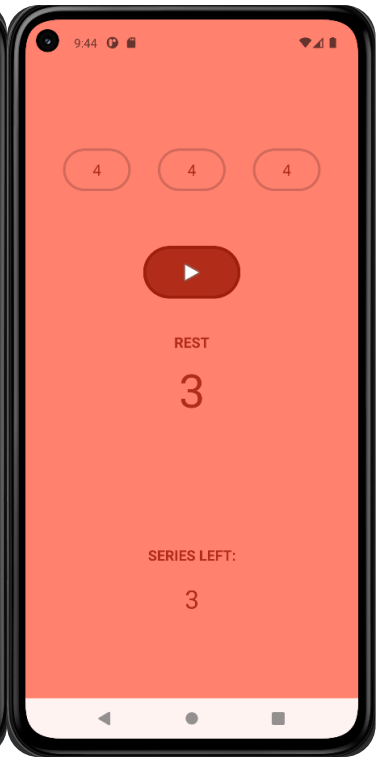
ESTADO INICIAL



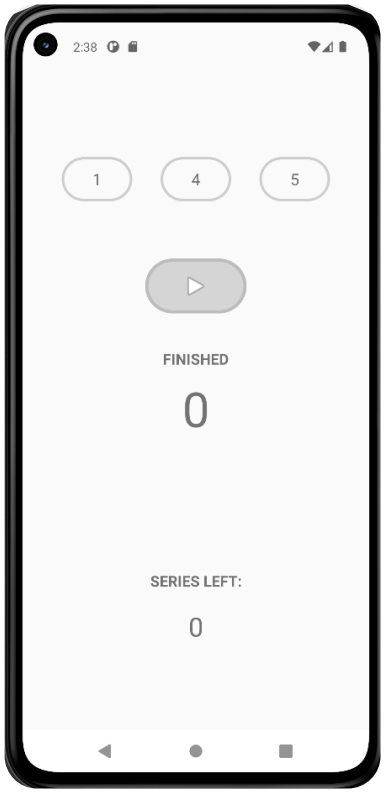
ENTRENAMIENTO



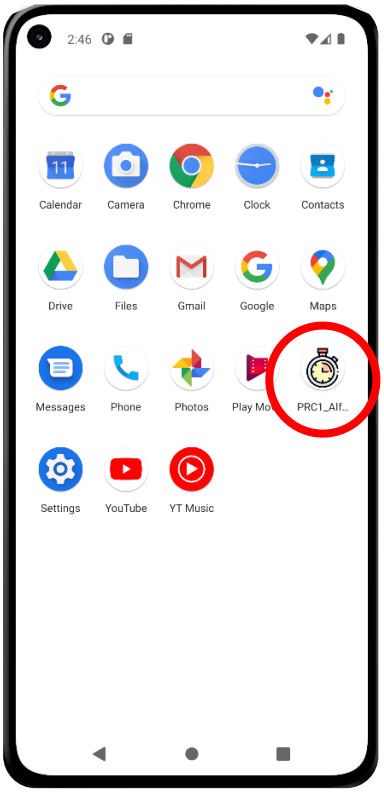
DESCANSO

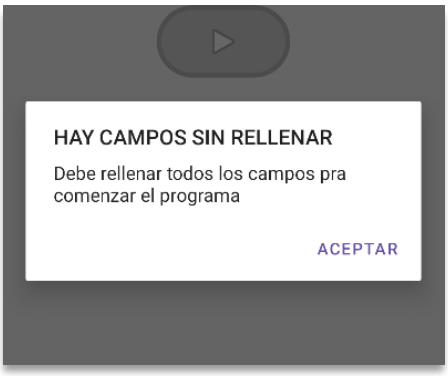


ESTADO FINAL



ICONO





DIÁLOGOS

COSAS QUE HE APRENDIDO

1. Utilizar el XML de color y de Strings.
2. Utilizar los drawables y crear estilos para elementos desde aquí.
3. Agregar un icono a la aplicación.
4. Usar diálogos.
5. Agregar sonidos.
6. Utilizar el método CountDownTimer.

BIBLIOGRAFÍA

Descripción general de MediaPlayer. (s. f.). Android Developers. https://developer.android.com/media/platform/mediaplayer?hl=es-419

How to make a countdown timer in Android? (s. f.). Stack Overflow. https://stackoverflow.com/questions/10032003/how-to-make-a-countdown-timer-in-android