2019

Documentacion de la Aplicación MyAPPs, progamada y diseñada por Emilio Rubiales Gutierrez

Desarrollada para Android 5.1 o superior

Android

MyAPPs

Emilio Rubiales

WWW.MYCLASS.ES

Obviando todo el contenido necesario para crear un Recycler View como los adaptadores, se explicara a continuación los últimos añadidos de la aplicación

Contenido

1. Reproductor de Musica

a. Servicio:

Para poder reproducir a través de una APP, un audio en segundo plano, debemos crear un servicio de sistema que se asignara a la propia APP, mientras esta esté en ejecución, aunque sea en segundo plano, podrá seguir reproduciendo audio.

Se ha creado una clase que gestionara dicho servicio:

```
/**
 * Clase MediaPlayerService, servicio que controla la ejecucion del media player
 */
public class MediaPlayerService extends Service implements
MediaPlayer.OnCompletionListener,
        MediaPlayer.OnPreparedListener, MediaPlayer.OnErrorListener,
MediaPlayer.OnSeekCompleteListener,
        MediaPlayer.OnInfoListener, MediaPlayer.OnBufferingUpdateListener,
        AudioManager.OnAudioFocusChangeListener {
```

Para poder gestionar su ciclo de vida, creamos los métodos necesarios:

Creación del servicio

```
/**
 * Metodos del ciclo de vida del servicio
 */
@Override
public IBinder onBind(Intent intent) {
    return iBinder;
}
@Override
public void onCreate() {
    super.onCreate();
    callStateListener();
    //Cambio del dispositivo de audio
    registerBecomingNoisyReceiver();
    //Listener a la espera de un nuevo audio
    register_playNewAudio();
}
```

Inicio del servicio

```
//Peticion para el inicio del servicio
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    try {

        //Cargamos la lista desde SharedPrefs
        StorageUtil storage = new StorageUtil(getApplicationContext());
        audioList = storage.loadAudio();
        audioIndex = storage.loadAudioIndex();


        if (audioIndex != -1 && audioIndex < audioList.size()) {
            //Comprobar que el indice esta dentro del rango establecido
            activeAudio = audioList.get(audioIndex);
        } else {
```

```
            stopSelf();
        }


    } catch (NullPointerException e) {
        stopSelf();
    }



    //Peticion del audioFocus
    if (requestAudioFocus() == false) {
        //No puede ganar el foco
        stopSelf();
    }



    if (mediaSessionManager == null) {
        try {
            initMediaSession();
            initMediaPlayer();
            buildNotification(PlaybackStatus.PLAYING);
        } catch (Exception e) {
            e.printStackTrace();
            stopSelf();
        }

    }

    //Manejo del intent de MediaControls
    handleIncomingActions(intent);
    return super.onStartCommand(intent, flags, startId);
}
```

Vinculación del Servicio a un proceso

```
/**
 * Bindeo del servicio
 */


public class LocalBinder extends Binder {
    public MediaPlayerService getService() {


        // Devuelve la instancia actual del servicio
        return MediaPlayerService.this;
    }
}
```

Para controlar y poder realizar una correcta eliminación del mismo, debemos instanciar dos métodos en concreto:

Desvincular Servicio

```
/**
 * Desbindeamos el servicio, desvinculamos el mediaplayer y quitamos la notificacion
 * @param intent
 * @return
 */
@Override
public boolean onUnbind(Intent intent) {
    mediaSession.release();
```

```
    removeNotification();
    stopMedia();

    return super.onUnbind(intent);
}
```

Destrucción del Servicio

```
@Override
public void onDestroy() {
    super.onDestroy();
    if (mediaPlayer != null) {
        stopMedia();
        mediaPlayer.release();
    }
    removeAudioFocus();
    //Desactivamos el listener de llamadas entrantes
    if (phoneStateListener != null) {
        telephonyManager.listen(phoneStateListener, PhoneStateListener.LISTEN_NONE);
    }

    removeNotification();

    //Eliminar los Broadcast Receiver
    unregisterReceiver(becomingNoisyReceiver);
    unregisterReceiver(playNewAudio);

    //clear cached playlist
    new StorageUtil(getApplicationContext()).clearCachedAudioPlaylist();

}
```

A continuación se explica los métodos que implican el uso del MediaPlayer por parte del Servicio para poder reproducir audios en segundo plano

Para ello empezaremos explicando los métodos para instanciar, reproducir, parar, pausar y resumir la reproducción actual.

Instancia del MediaPlayer

```
/**
 * MediaPlayer acciones
 */
private void initMediaPlayer() {
    if (mediaPlayer == null)
        mediaPlayer = new MediaPlayer();//nueva instancia del media player

    //Seteamos los listener del media player
    mediaPlayer.setOnCompletionListener(this);
    mediaPlayer.setOnErrorListener(this);
    mediaPlayer.setOnPreparedListener(this);
    mediaPlayer.setOnBufferingUpdateListener(this);
    mediaPlayer.setOnSeekCompleteListener(this);
    mediaPlayer.setOnInfoListener(this);
```

```
    //Reseteamos el media player
    mediaPlayer.reset();


    mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
    try {
        // Seteamos el datasource al audio actual
        mediaPlayer.setDataSource(activeAudio.getData());
    } catch (Exception e) {
        e.printStackTrace();
        stopSelf();
    }
    mediaPlayer.prepareAsync();
}
```

Reproducir y Parar

```
/**
 * Reproducimos una cancion
 */
private void playMedia() {
    try {
        if (!mediaPlayer.isPlaying()) {
            mediaPlayer.start();

        }
        buildNotification(PlaybackStatus.PLAYING);
    }catch(Exception ex){

        initMediaPlayer();
    this.playMedia();
}
}

/**
 * Paramos la reproduccion
 */
public void stopMedia() {
    if (mediaPlayer == null) return;
    if (mediaPlayer.isPlaying()) {
        mediaPlayer.stop();
    }
    removeNotification();
}
```

Para cuando la reproducción está en marcha, podremos pausar esta misma o resumirla en vez de reiniciar completamente el mediaPlayer[1]

Resumir y Pausar

```
/**
 * Pausamos la reproduccion
 */
public void pauseMedia() {
    try {
        if (mediaPlayer.isPlaying()) {
            mediaPlayer.pause();
            resumePosition = mediaPlayer.getCurrentPosition();
        }
        buildNotification(PlaybackStatus.PAUSED);
    }catch(Exception ex){
        initMediaPlayer();
```

[1] MediaPlayer: Item que ejecuta-reproduce archivos que contengan audio, video y algunas extensiones mas

```
        this.playMedia();
    }
}
/**
 * Continuamos la reproduccion
 */
public void resumeMedia() {
    try {
        if (!mediaPlayer.isPlaying()) {
            mediaPlayer.seekTo(resumePosition);
            mediaPlayer.start();
            buildNotification(PlaybackStatus.PLAYING);
        }
    }catch(Exception ex){
        initMediaPlayer();
        this.playMedia();
    }
}
```

También se añaden controles para poder pasar a la siguiente o a la canción anterior

Controles para cambiar de canción

```
/**
 * Pasamos a la siguiente cancion
 */
public void skipToNext() {
    if (audioIndex == audioList.size() - 1) {
        //Si es el ultimo de la playlist
        audioIndex = 0;
        activeAudio = audioList.get(audioIndex);
    } else {
        //cogemos la siguiente
        activeAudio = audioList.get(++audioIndex);
    }

    //Actualizamos el index en el XML
    new StorageUtil(getApplicationContext()).storeAudioIndex(audioIndex);
    stopMedia();
    //reseteamos el mp
    mediaPlayer.reset();
    initMediaPlayer();
    buildNotification(PlaybackStatus.PLAYING);
}
/**
 * Pasar a la cancion anterior
 */
public void skipToPrevious() {
    if (audioIndex == 0) {
        //Si es el primero
        //Seteamos el index al ultimo
        audioIndex = audioList.size() - 1;
        activeAudio = audioList.get(audioIndex);
    } else {
        //Cogemos el ultimo de la lista
        activeAudio = audioList.get(--audioIndex);
    }
    //Actualizamos el index en el XML
    new StorageUtil(getApplicationContext()).storeAudioIndex(audioIndex);

    stopMedia();
    //reseteamos el MP
    mediaPlayer.reset();
    initMediaPlayer();
    buildNotification(PlaybackStatus.PLAYING);
}
```

Para manejar el tipo de OutPut de Audio(Auriculares, altavoz, etc) usamos un método que maneja el tipo de Broadcast generando una acción

Manejo del Broadcast

```
/**
 * ACTION_AUDIO_BECOMING_NOISY -- cambiamos el output del audio
 */
private BroadcastReceiver becomingNoisyReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //pausamos on ACTION_AUDIO_BECOMING_NOISY
        pauseMedia();
        buildNotification(PlaybackStatus.PAUSED);
    }
};


private void registerBecomingNoisyReceiver() {
    //registramos despues de ganar el foco
    IntentFilter intentFilter = new
IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
    registerReceiver(becomingNoisyReceiver, intentFilter);
}
```

Al igual que con el dispositivo de audio, cuando algo interfiere en el foco del servicio (llamada entrante u otra aplicación que utilice una salida de audio)

Manejo de llamadas entrantes

```
private void callStateListener() {
    // Get manager
    telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
    //Instanciamos el listener
    phoneStateListener = new PhoneStateListener() {
        @Override
        public void onCallStateChanged(int state, String incomingNumber) {
            switch (state) {
                //Si una llamada existe el mp
                case TelephonyManager.CALL_STATE_OFFHOOK:
                case TelephonyManager.CALL_STATE_RINGING:
                    if (mediaPlayer != null) {
                        pauseMedia();
                        ongoingCall = true;
                    }
                    break;
                case TelephonyManager.CALL_STATE_IDLE:
                    //Se va la llamada, continuamos
                    if (mediaPlayer != null) {
                        if (ongoingCall) {
                            ongoingCall = false;
                            resumeMedia();
                        }
                    }
                    break;
            }
        }
    };
    // Registramos el telefono con el manager a la espera de cambios
    telephonyManager.listen(phoneStateListener,
            PhoneStateListener.LISTEN_CALL_STATE);
}
```

Para gestionar el servicio y las acciones junto a la notificacion, debemos instanciar un MediaSession el cual llamara a los métodos explicados previamente para manejar la reproducción

Instanciar MediaSession

```
private void initMediaSession() throws RemoteException {
    if (mediaSessionManager != null) return; //mediaSessionManager existe

    mediaSessionManager = (MediaSessionManager)
getSystemService(Context.MEDIA_SESSION_SERVICE);
    // instanciamos uno nuevo
    mediaSession = new MediaSession(getApplicationContext(), "AudioPlayer");
    //cogemos los controles
    transportControls = mediaSession.getController().getTransportControls();

    mediaSession.setActive(true);
    //Indicamos que tenga los controles de transport
    mediaSession.setFlags(MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);

    //Seteamos el metadata
    updateMetaData();

    // Asignamos el callback al mediasession
    mediaSession.setCallback(new MediaSession.Callback() {
        // Implementamos el  callbacks a los metodos del mediaPlayer
        @Override
        public void onPlay();
        @Override
        public void onPause();
        @Override
        public void onSkipToNext();

        @Override
        public void onSkipToPrevious();

        @Override
        public void onStop();

         @Override
        public void onSeekTo();
    });
}
```

Para el metadata del MediaSession[2], le actualizamos los datos a la canción actual

Actualizar Metadatos del MediaSession

```
/**
 * Actualizamos el metadata
 */
private void updateMetaData() {
    try {
        mediaSession.setMetadata(new MediaMetadata.Builder()
                .putBitmap(MediaMetadata.METADATA_KEY_ALBUM_ART,
getAlbumart(Long.parseLong(activeAudio.getCaratula())))
                .putString(MediaMetadata.METADATA_KEY_ARTIST, activeAudio.getArtist())
                .putString(MediaMetadata.METADATA_KEY_ALBUM, activeAudio.getAlbum())
                .putString(MediaMetadata.METADATA_KEY_TITLE, activeAudio.getTitle())
                .build());
    }catch(Exception ignored){}
}
```

Para poder crear una notificación que pueda gestionar los controles y la canción actual del reproductor, iniciaremos primero los canales por el cual va a visualizar dicha notificación

Instancia de los canales de notificación

```
/**
 * Iniciamos los canales de notificacion
 * @param context
```

[2] MediaSession: Item que maneja y controla el mediaPlayer

```
 */
public void initChannels(Context context) {
    if (Build.VERSION.SDK_INT < 26) {
        return;
    }
    NotificationManager notificationManager =
            (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
    NotificationChannel channel = new NotificationChannel("default",
            "Channel name",
            NotificationManager.IMPORTANCE_DEFAULT);
    channel.setDescription("Channel description");
    notificationManager.createNotificationChannel(channel);
}
```

Y a continuacion construimos la notificación, para ello instanciamos una **NotificationCompat.Builder**[3] en la cual le pasamos todos los parámetros de texto, imagen y botones de acción necesarios

Instancia del NotificationCompat.Builder

```
private void buildNotification(PlaybackStatus playbackStatus) {

    int notificationAction = android.R.drawable.ic_media_pause;//Necesita ser
instanciado
    PendingIntent play_pauseAction = null;
    initChannels(getApplicationContext());

    //Construimos una nueva notificacion
    if (playbackStatus == PlaybackStatus.PLAYING) {
        notificationAction = android.R.drawable.ic_media_pause;
        //Accion de pausa
        play_pauseAction = playbackAction(1);
    } else if (playbackStatus == PlaybackStatus.PAUSED) {
        notificationAction = android.R.drawable.ic_media_play;
        //Accion de reproducir
        play_pauseAction = playbackAction(0);
    }

    Bitmap largeIcon = getAlbumart(Long.parseLong(activeAudio.getCaratula()));
    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
    intent.putExtra("musica", "gotoMusica");
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_SINGLE_TOP);
    PendingIntent pendingIntent = PendingIntent.getActivity(getApplicationContext(), 1,
intent,PendingIntent.FLAG_UPDATE_CURRENT);


    Bundle bundle = new Bundle();
    bundle.putString("titulo", activeAudio.getTitle());
    bundle.putString("artista", activeAudio.getArtist());
    bundle.putString("album", activeAudio.getAlbum());
    bundle.putString("data", activeAudio.getData());
    bundle.putBoolean("shuffle",MainActivity.mode);


    // Creamos la notificacion
    NotificationCompat.Builder notificationBuilder = (NotificationCompat.Builder) new
NotificationCompat.Builder(getApplicationContext(),"default").setOngoing(true)
            // Escondemos el tiempo de vida
            .setShowWhen(false)
            // Seteamos el estilo
            .setStyle(new androidx.media.app.NotificationCompat.MediaStyle()
                    // Asignamos nuestro token de la sesion

.setMediaSession(MediaSessionCompat.Token.fromToken(mediaSession.getSessionToken()))
```

[3] NotificationCompat.Builder: Constructor del NotificationCompat que permite controlar fácilmente la “banderas” usadas asi como el layout de la notificacion

```
                    // Mostramos los controles
                    .setShowActionsInCompactView(0, 1, 2))
            .setColor(getResources().getColor(R.color.eurogamer))
            .setLargeIcon(largeIcon)
            .setSmallIcon(android.R.drawable.stat_sys_headset)
            .setContentText(String.format("%s | %s", activeAudio.getArtist(),
activeAudio.getAlbum()))
            .setContentTitle(activeAudio.getTitle())
            .setContentInfo(activeAudio.getData())
            .setContentIntent(pendingIntent)
            // Agregamos las acciones de reproduccion
            .addAction(android.R.drawable.ic_media_previous, "previous",
playbackAction(3))
            .addAction(notificationAction, "pause", play_pauseAction)
            .addAction(android.R.drawable.ic_media_next, "next", playbackAction(2))
            .addExtras(bundle);


    ((NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE)).notify(NOTIFICATION_ID,
notificationBuilder.build());
}
```

Para poder manejar las acciones desde la notificacion, creamos un método que, dependiendo del código de acción, ejecutará esta misma

Acciones del PlayBack

```
private PendingIntent playbackAction(int actionNumber) {
    Intent playbackAction = new Intent(this, MediaPlayerService.class);
    switch (actionNumber) {
        case 0:
            playbackAction.setAction(ACTION_PLAY);
            return PendingIntent.getService(this, actionNumber, playbackAction, 0);
        case 1:
            playbackAction.setAction(ACTION_PAUSE);
            return PendingIntent.getService(this, actionNumber, playbackAction, 0);
        case 2:
            playbackAction.setAction(ACTION_NEXT);
            return PendingIntent.getService(this, actionNumber, playbackAction, 0);
        case 3:
            playbackAction.setAction(ACTION_PREVIOUS);
            return PendingIntent.getService(this, actionNumber, playbackAction, 0);
        default:
            break;
    }
    return null;
}
```

En caso de que el servicio se haya detenido desde la App o desde el administrador, este removerá la notificación

Remover Notificacion

```
private void removeNotification() {
    NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.cancel(NOTIFICATION_ID);
    stopSelf();
}
```

Como hemos visto antes, se pueden enviar acciones desde la notificacion, esas acciones tambien las enviamos desde los Fragment pero el método que las interpreta es el siguiente

Manejo de Acciones

```
private void handleIncomingActions(Intent playbackAction) {
    if (playbackAction == null || playbackAction.getAction() == null) return;
```

```
    String actionString = playbackAction.getAction();
    if (actionString.equalsIgnoreCase(ACTION_PLAY)) {
        transportControls.play();
    } else if (actionString.equalsIgnoreCase(ACTION_PAUSE)) {
        transportControls.pause();
    } else if (actionString.equalsIgnoreCase(ACTION_NEXT)) {
        transportControls.skipToNext();
    } else if (actionString.equalsIgnoreCase(ACTION_PREVIOUS)) {
        transportControls.skipToPrevious();
    } else if (actionString.equalsIgnoreCase(ACTION_STOP)) {
        transportControls.stop();
    }
}
```

Por último, si al propio servicio se le indica que reproduzca otro audio cuando ya está otro en progreso, se llama a través de un BroadcastReceiver[4] el cual reinicia todo el reproductor para usarlo con otro audio pulsado, no con skipTo…

BroadCastReceiver para reproducir otro audio

```
private BroadcastReceiver playNewAudio = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        //Cogemos el index de nuestro XML
        StorageUtil storage = new StorageUtil(getApplicationContext());
        audioList = storage.loadAudio();

        audioIndex = new StorageUtil(getApplicationContext()).loadAudioIndex();
        if (audioIndex != -1 && audioIndex < audioList.size()) {
            //Comprobamos que este dentro del rango
            activeAudio = audioList.get(audioIndex);
        } else {
            stopSelf();
        }
        //Accion PLAY_NEW_AUDIO recibida
        //Reseteamos el mediaplayer
        stopMedia();
        mediaPlayer.reset();
        initMediaPlayer();
        updateMetaData();
        buildNotification(PlaybackStatus.PLAYING);
    }
};
```

Para terminar, cuando tenemos que reproducir otro audio, debemos registrarlo mediante un Intent[5] con un código que hemos generado nosotros, siendo este una constante

Registrar nuevo audio

```
private void register_playNewAudio() {
    //Registramos recibidor de PlayNewAudio
    IntentFilter filter = new
IntentFilter(MusicFragment.Broadcast_PLAY_NEW_AUDIO);//Codigo Constante
    registerReceiver(playNewAudio, filter);
}
```

[4] BroadcastReceiver: Componente que permite registrar acciones de aplicación o sistema

[5] Intent: Elemento que permite inicializar y transmitir datos entre aplicaciones o componentes de la misma

b. *Utilidad de Almacenamiento*

Como sabemos, cada aplicación tiene un "SandBox[6]" propio, esto limita las funciones de interactuar directamente con un servicio(Al estar en otro SandBox), como en nuestro caso. Pero si queremos enviar datos, podemos crear un fichero PUBLICO el cual podrá leer/escribir nuestro servicio.

Con esta funcionalidad, hemos creado esta clase, la cual manejará un archivo XML creado en el paquete de la APP en el dispositivo donde se almacenará la lista de reproducción en JSON y el index de la canción actual para poder manejarlo desde el servicio

```
/**
 * Clase que maneja el XML+JSON de almacenamiento de la lista de reproduccion y el index
actual
 */
public class StorageUtil {

    //Nombre del storage
    private final String STORAGE = " com.emiliorg.myrss.STORAGE";
    private SharedPreferences preferences;
    private Context context;

    public StorageUtil(Context context) {
        this.context = context;
    }
```

Como hemos indicado arriba, lo que realizaremos con esta clase al instanciarla, es poder guardar la lista de reproducción para que el servicio pueda recuperarla y asi ejecutar las canciones.

Gracias a la librería GSON[7] podemos realizar todas estas acciones, convirtiendo la lista de reproducción en ArrayList a un JSON almacenado en un XML.

A continuación explicaremos sus métodos.

Almacenamiento de la PlayList convertida a JSON dentro del XML

```
public void storeAudio(ArrayList<Audio> arrayList) {
    preferences = context.getSharedPreferences(STORAGE, Context.MODE_PRIVATE);

    SharedPreferences.Editor editor = preferences.edit();
    Gson gson = new Gson();
    String json = gson.toJson(arrayList);
    editor.putString("audioArrayList", json);
    editor.apply();
}
```

Almacenamiento del index de la cancion actual dentro del XML

```
public void storeAudioIndex(int index) {
    preferences = context.getSharedPreferences(STORAGE, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putInt("audioIndex", index);
    editor.apply();
}
```

Recoge la PlayList del XML transformándola a ArrayList

[6] SandBox: Es el aislamiento de procesos. Se utiliza para definir los limites de una APP en un espacio cerrado el cual no podrá exceder los limites del mismo

[7] GSON: Libreria para trabajar en formato JSON proporcionada por Google con la que transformamos los arrayList a J

```
public ArrayList<Audio> loadAudio() {
    preferences = context.getSharedPreferences(STORAGE, Context.MODE_PRIVATE);
    Gson gson = new Gson();
    String json = preferences.getString("audioArrayList", null);
    Type type = new TypeToken<ArrayList<Audio>>() {
    }.getType();
    return gson.fromJson(json, type);
}
```

Recoge el index actual del XML

```
public int loadAudioIndex() {
    preferences = context.getSharedPreferences(STORAGE, Context.MODE_PRIVATE);
    return preferences.getInt("audioIndex", -1);//devolvemos -1 si no encuentra nada
}
```

Por ultimo, agregamos un método que nos sirve para limpiar el fichero al salir de la aplicación para que no queden datos “residuales” dentro

Limpieza del fichero “Cache”

```
public void clearCachedAudioPlaylist() {
    preferences = context.getSharedPreferences(STORAGE, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();
    editor.clear().apply();
}
```

c. *Estado de PlayBack*

Variable creada para mandar a nuestra “manera” un tipo valor propio, simplemente tiene dos valores: “PLAYING” y “PAUSED”

```
public enum PlaybackStatus {
    PLAYING,
    PAUSED
}
```

d. *Objeto Audio*

Objeto creado con la finalidad de almacenar todos los archivos de extension de audio en nuestro programa y que implementa Serializable[8]

Objeto Audio

```
public class Audio implements Serializable {

    private String data;
    private String title;
    private String album;
    private String artist;
    private String caratula;

    public Audio(String data, String title, String album, String artist, String
caratula) {
        this.data = data;
        this.title = title;
        this.album = album;
        this.artist = artist;
```

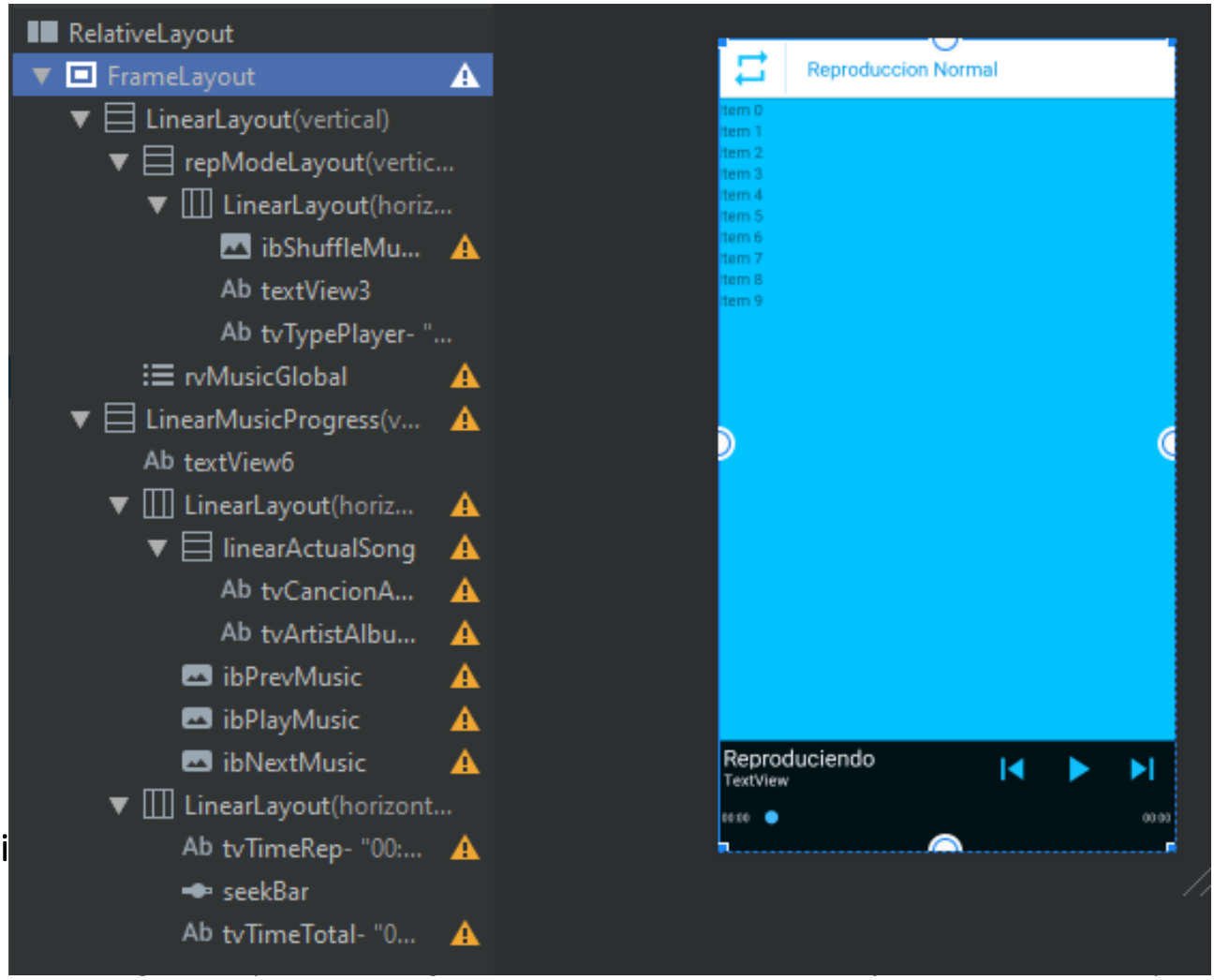
[8] Serializable: Interfaz la cual nos permite transformar los objetos a bytes

```
        this.caratula = caratula;
    }
```

e. *Fragment: Lista de Canciones y Reproductor*

i. Layout

Layout bastante compuesto por un padre RelativeLayout con un FrameLayout dentro donde, con LinearLayouts, distribuimos todo el contenido: RecyclerView, botones de aleatorio, seekBar y textviews de reproducción actual.



Ademas se ha optimizado la carga de datos permitiendo ejecutar esta recolecta de archivos en segundo plano tanto del XML como del storage del dispositivo.

Empezaremos explicando las AsyncTask[9] que gestionan la recolecta de datos.

[9] AsyncTask: Tarea asincrona que trabaja en Segundo plano para luego publicar el resultado en el Hilo de la interfaz

AsyncTask que recoge los datos del XML

```
public class musicLoaderXML extends AsyncTask<Void, Void, Void>{
    @Override
    protected Void doInBackground(Void... voids) {
        //Llamando a nuestra clase StorageUtil, recoge los datos del XML con los metodos
        StorageUtil storage = new StorageUtil(getContext());
        audioList = new ArrayList<>();
        audioList = storage.loadAudio();
        return null;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        //Si la lista ha salido null significa que el xml esta vacio, procedemos
        //a llamar la siguiente tarea que carga del sistema de archivos
        if (audioList == null) {
            new musicLoader().execute();
        } else {
            prelista = audioList;
            initRecyclerView();
        }
    }
}
```

AsyncTask que recoge los archivos del sistema de archivos

```
        public class musicLoader extends AsyncTask<Void, Void, Void> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        playModeLayout.setClickable(false);
    }

    @Override
    protected Void doInBackground(Void... voids) {
        loadAudio();//Carga mediante un metodo los archivos en un arraylist<Audio>
        return null;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        initRecyclerView();//Instancia el recycler
        playModeLayout.setClickable(true);
        if (!titulo.getText().equals(getResources().getString(R.string.reproduciendo)))
{
            playAudio(0);//Inicia el primer audio
        }
    }
}
```

Para coger los archivos del dispositivos, tenemos que explicar el metodo que los extrae y los guarda en un arraylist tipo Audio. Conseguimos esto instanciando un ContentResolver[10] que, con una Uri y un cursor, recoge los archivos

Metodo para recoger todos los archivos tipo audio

```
public void loadAudio() {
    ContentResolver contentResolver = getActivity().getContentResolver();
```

[10] ContentResolver: Componente de Android que trabaja para compartir datos

```
    Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;//Donde vamos a buscar
    String selection = MediaStore.Audio.Media.IS_MUSIC + " != 0";//Tipo de archivo
    String sortOrder = MediaStore.Audio.Media.TITLE + " ASC";//Orden
    //Creamos el cursor
    Cursor cursor = contentResolver.query(uri, null, selection, null, sortOrder);

    if (cursor != null && cursor.getCount() > 0) {//Si tiene contenido
        audioList = new ArrayList<>();
        //Recorremos el cursor
        while (cursor.moveToNext()) {
            String data =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.DATA));
            String title =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE));
            String album =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM));
            String artist =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST));
            String photo =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID));
            //Guardamos en la lista de audios
            audioList.add(new Audio(data, title, album, artist, photo));
        }
    }
    if (cursor != null) {
        cursor.close();//Cerramos el cursor}

        prelista = audioList;//Guardamos una lista previa
        StorageUtil storage = new StorageUtil(getContext());//Instanciamos nuestro
StorageUtil
        storage.clearCachedAudioPlaylist();//Limpiamos el XML
        storage.storeAudioIndex(-1);//Almacenamos el audio
        storage.storeAudio(prelista);//Almacenamos la lista
    }
}
```

Tambien hemos agregado una nueva funcionalidad a la APP que permite ordenar aleatoriamente la lista de reproducción, ese orden lo realizamos mediante una AsyncTask mas

AsyncTask para ordenar aleatoriamente

```
public class shuffleLoader extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... voids) {
        preshuffle = prelista;//Recoge la lista
        Collections.shuffle(preshuffle);//Reordena aleatoriamente la lista
        audioList = new ArrayList<>();
        audioList = preshuffle;
        StorageUtil storage = new StorageUtil(getContext());
        storage.clearCachedAudioPlaylist();//limpiamos el xml
        storage.storeAudioIndex(-1);//almacenamos el index
        storage.storeAudio(audioList);//almacenamos la lista
        return null;
    }
    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        initRecyclerView();//Reinstanciamos nuestro recycler
        playModeLayout.setClickable(true);
        playAudio(0);//Reproducimos el primer audio
    }

}
```

En dicha AsyncTask, ordenamos la lista con el método shuffle de la clase Collections de Java.

Pasemos a explicar los métodos que controlan el ciclo de vida del Fragment.

En el propio onCreate simplemente instanciamos los ítems del layout en el código(obviamos adjuntarlo ya que se supone que el lector tiene conocimientos sobre Android) para poder trabajar con ellos, al igual que sus listeners

Listeners

SeekBar

```
private void handleSeekbar() {
    seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {

            if (((MainActivity)getActivity()).player.getMediaPlayer() != null &&
fromUser) {
                ((MainActivity)getActivity()).player.getMediaPlayer().seekTo(progress *
1000);

current.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMediaP
layer().getCurrentPosition()));
            }
        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar;
        @Override
        public void onStopTrackingTouch(SeekBar seekBar);
    });

seekBar.getProgressDrawable().setColorFilter(getResources().getColor(R.color.whitebox),
PorterDuff.Mode.SRC_ATOP);
}
```

Botones de la interfaz

Boton de Play, Next y Previous

```
next.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Pasamos de cancion
        ((MainActivity)getActivity()).player.skipToNext();
    }
});

prev.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //Volvemos a la cancion anterior
        ((MainActivity)getActivity()).player.skipToPrevious();

    }
});
play.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            //Comprobamos si esta reproduciendo
            if (((MainActivity)getActivity()).player.getMediaPlayer().isPlaying()) {
                play.setImageResource(R.drawable.bt_play);//Cambiamos la imagen
                //Pausamos la cancion
                ((MainActivity)getActivity()).player.pauseMedia();
            } else {
                //Continuamos reproduciendo
```

```
                ((MainActivity)getActivity()).player.resumeMedia();
                play.setImageResource(R.drawable.bt_stop);//Cambiamos la imagen
            }
        } catch (Exception ex) {
            //Volvemos a iniciar el servicio
            playerIntent = new Intent(getContext(), MediaPlayerService.class);
            getActivity().startService(playerIntent);
            getActivity().bindService(playerIntent,
((MainActivity)getActivity()).serviceConnection, Context.BIND_AUTO_CREATE);
        }

    }
});
```

Boton de reproducir aleatoriamente o ordenada

```
playModeLayout.setOnClickListener(new View.OnClickListener() {
    @Override

    public void onClick(View v) {
        if (!MainActivity.mode) {
            //Reproducimos aleatoriamente las canciones
            playMode.setText(R.string.rep_shuffler);
            shuffle.setImageResource(R.drawable.bt_shuffle);
            new shuffleLoader().execute();
            MainActivity.mode = true;


        } else {
            //Reproducimos las canciones ordenadas por nombre
            playMode.setText(R.string.rep_normal);
            shuffle.setImageResource(R.drawable.bt_repeat);
            new musicLoader().execute();
            MainActivity.mode = false;


        }
    }
});
```

Boton para ir al fragment de la canción actual

```
playModeLayout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (!MainActivity.mode) {
            //Reproducimos aleatoriamente las canciones
            playMode.setText(R.string.rep_shuffler);
            shuffle.setImageResource(R.drawable.bt_shuffle);
            new shuffleLoader().execute();
            MainActivity.mode = true;

        } else {
            //Reproducimos las canciones ordenadas por nombre
            playMode.setText(R.string.rep_normal);
            shuffle.setImageResource(R.drawable.bt_repeat);
            new musicLoader().execute();
            MainActivity.mode = false;
        }
    }
});
```

Metodos para devolver estados o el propio Fragment

```
public boolean isPlaying() {
    return playing;
}
public void setPlaying(boolean playing) {
    this.playing = playing;
}
```

```
public MusicFragment returnThis() {

    return this;
}
```

Para poder reproducir una canción usamos un método con una sobrecarga

```
public void playAudio(int audioIndex) {
    //Check is service is active
    if (!((MainActivity)getActivity()).serviceBound) {
        //Almacena la playlist serializada en el XML
        StorageUtil storage = new StorageUtil(getContext());
        storage.storeAudioIndex(audioIndex);
        audioPos = audioIndex;
        playerIntent = new Intent(getContext(), MediaPlayerService.class);
        getActivity().startService(playerIntent);
        getActivity().bindService(playerIntent,
((MainActivity)getActivity()).serviceConnection, Context.BIND_AUTO_CREATE);
    } else {
        //Almacena el nuevo audio index
        StorageUtil storage = new StorageUtil(getContext());
        storage.storeAudioIndex(audioIndex);
        audioPos = audioIndex;
        Intent broadcastIntent = new Intent(Broadcast_PLAY_NEW_AUDIO);
        getActivity().sendBroadcast(broadcastIntent);
    }
    seekbarLayout.setVisibility(View.VISIBLE);
    play.setImageResource(R.drawable.bt_stop);

    titulo.setText(String.format("%s | %s",
getResources().getString(R.string.reproduciendo),
audioList.get(audioIndex).getTitle()));
    artistalbum.setText(String.format("%s | %s", audioList.get(audioPos).getArtist(),
audioList.get(audioIndex).getAlbum()));
    titulo.setSelected(true);
    artistalbum.setSelected(true);

    handler = new Handler();
    audioActual();

    if (!playing) {
        setPlaying(true);
        PlayingFragment playingFragment;
        StorageUtil storage = new StorageUtil(getContext());
        playingFragment = new PlayingFragment(storage.loadAudio(), audioIndex, this);

getFragmentManager().beginTransaction().setCustomAnimations(R.anim.fragment_effect,
R.anim.fragment_effect_exit, R.anim.fragment_effect, R.anim.fragment_effect_exit)
                .add(R.id.nav_host_fragment,
playingFragment).addToBackStack(null).commit();
        playing = true;
    }
}
```

*Sobrecarga del metodo **playAudio***

```
public void playAudio(int audioIndex, boolean back) {
    //Check is service is active
    if (!((MainActivity)getActivity()).serviceBound) {
        //Store Serializable audioList to SharedPreferences
        StorageUtil storage = new StorageUtil(getContext());
        storage.storeAudio(audioList);
        storage.storeAudioIndex(audioIndex);

        new Thread() {
            public void run() {

                playerIntent = new Intent(getContext(), MediaPlayerService.class);
```

```
                getActivity().startService(playerIntent);
                getActivity().bindService(playerIntent,
((MainActivity)getActivity()).serviceConnection, Context.BIND_AUTO_CREATE);
            }
        }.start();


    } else {
        //Store the new audioIndex to SharedPreferences
        StorageUtil storage = new StorageUtil(getContext());
        storage.storeAudio(audioList);
        storage.storeAudioIndex(audioIndex);
        new Thread() {
            public void run() {
                //Service is active
                //Send a broadcast to the service -> PLAY_NEW_AUDIO
                Intent broadcastIntent = new Intent(Broadcast_PLAY_NEW_AUDIO);
                getActivity().sendBroadcast(broadcastIntent);

            }
        }.start();
    }

    seekbarLayout.setVisibility(View.VISIBLE);
    play.setImageResource(R.drawable.bt_stop);

    artistalbum.setText(audioList.get(audioIndex).getTitle());


    handler = new Handler();
    audioActual();


}
```

Como he comentado en el breve resumen, el diseño de la interfaz lo manejamos a través de código gracias a multiples hilos que trabajan en el Hilo principal de la UI. Gracias a ello conseguimos interactuar con los elementos en tiempo real para que el usuario visualice sin ningún problema todo lo que ocurre al momento.

Metodo que contiene los hilos: ***audioActual()***

Hilo que maneja el progreso de la seekBar – Cada 1 segundo

```
new Thread() {
    @Override
    public void run() {
        try {
            getActivity().runOnUiThread(runnable = new Runnable() {
                @Override
                public void run() {
                    seekbarLayout.setVisibility(View.VISIBLE);
                    if (((MainActivity)getActivity()).player.getMediaPlayer() != null) {
                        try {

seekBar.setMax(((MainActivity)getActivity()).player.getMediaPlayer().getDuration() /
```

```
1000);
                            int mCurrentPosition =
((MainActivity)getActivity()).player.getMediaPlayer().getCurrentPosition() / 1000;
                            seekBar.setProgress(mCurrentPosition);

current.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMediaP
layer().getCurrentPosition()));

total.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMediaPla
yer().getDuration()));
                        } catch (Exception ex) {
                        }
                    }
                }
            });
            handler.postDelayed(this, 1000);
        } catch (Exception ex) {
        }
    }


}.start();
```

Hilo que maneja los textos del titulo y álbum/artista de la canción actual – Cada 300 ms

```
new Thread() {
    @Override
    public void run() {
        try {
            getActivity().runOnUiThread(actual = new Runnable() {
                @Override
                public void run() {
                    if (((MainActivity)getActivity()).player.getMediaPlayer() != null) {
                        if
(((MainActivity)getActivity()).player.getMediaPlayer().isPlaying()) {

                            play.setImageResource(R.drawable.bt_stop);
                            StorageUtil storageUtil = new StorageUtil(getContext());
                            try {
                                if (audioPos != storageUtil.loadAudioIndex()) {
                                    audioPos = storageUtil.loadAudioIndex();
                                    titulo.setText(String.format("%s | %s",
getResources().getString(R.string.reproduciendo), audioList.get(audioPos).getTitle()));
                                    artistalbum.setText(String.format("%s | %s",
audioList.get(audioPos).getArtist(), audioList.get(audioPos).getAlbum()));
                                    titulo.setSelected(true);
                                    artistalbum.setSelected(true);
                                }
                            } catch (Exception ex) {}
                        } else {
                            play.setImageResource(R.drawable.bt_play);
                        }}}});handler.postDelayed(this, 300); } catch (Exception ex) {}}
}.start();
```

Agregar que para los ítems del Recycler view, le generamos una imagen a las canciones la cual extraemos del ID que podemos obtener desde sus metadatos

Metodo para generar un bitmap a partir de una ID

```
public Bitmap getAlbumart(Long album_id) {
    Bitmap bm = null;
    try {
        final Uri sArtworkUri = Uri
                .parse("content://media/external/audio/albumart");

        Uri uri = ContentUris.withAppendedId(sArtworkUri, album_id);

        ParcelFileDescriptor pfd = getContext().getContentResolver()
```

```
                .openFileDescriptor(uri, "r");

        if (pfd != null) {
            FileDescriptor fd = pfd.getFileDescriptor();
            bm = BitmapFactory.decodeFileDescriptor(fd);
        }
    } catch (Exception e) {
    }
    if (bm == null) {
        bm = BitmapFactory.decodeResource(getResources(), R.drawable.musicdef);
    }
    return bm;
}
```

Por ultimo, la única modificación realizada en el recyclerView, los ítems tienen un onClickListener en un imageButton el cual despliega un menú que, o inicia un sharing intent para compartir el audio o para borrar la canción

Listener

```
holder.more.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        PopupMenu popup = new PopupMenu(musicFragment.getContext(), holder.more);
        //Inflating the Popup using xml file
        popup.getMenuInflater()
                .inflate(R.menu.song_menu, popup.getMenu());
        //registering popup with OnMenuItemClickListener
        popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
            public boolean onMenuItemClick(MenuItem item) {
                int id = item.getItemId();
                switch (id) {
                    case R.id.compartir:
                        //Realizamos un Intent pasado extras relacionados al asunto y al
cuerpo del mensaje
                        Intent sharingIntent = new
Intent(android.content.Intent.ACTION_SEND);
                        sharingIntent.setType("audio/*");
                        sharingIntent.putExtra(Intent.EXTRA_STREAM,
Uri.parse(musica.getData()));
                        musicFragment.startActivity(Intent.createChooser(sharingIntent,
"Compartido desde MyAPPs de Emilio Rubiales"));
                        break;
                    case R.id.borrar:
                            alertDialog(musica.getData());
                        break;
                }
                return true;
            }
        });
        popup.show();}
});
```

Dicha accion de borrar despliega un dialogo el cual pregunta al usuario si de verdad quiere borrar el archivo

Dialogo

```
private void alertDialog(final String pathBorrado) {
    AlertDialog.Builder dialog = new AlertDialog.Builder(musicFragment.getContext());
    dialog.setMessage("Accion irreversible: ¿Quiere borrar la cancion?");
    dialog.setTitle("Alerta");

    System.out.println(pathBorrado);
    dialog.setPositiveButton("Confirmar",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,
                                    int which) {

                    Toast.makeText(musicFragment.getContext(), "Elemento borrado",
Toast.LENGTH_LONG).show();
```

```
                    MyFiles.delete(musicFragment.getContext(), new File(pathBorrado));
                    musicFragment.loadList();
                }
            });
    dialog.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {


            Toast.makeText(musicFragment.getContext(), "Borrado Cancelado",
Toast.LENGTH_SHORT);
        }
    });
    AlertDialog alertDialog = dialog.create();
    alertDialog.show();
}
```

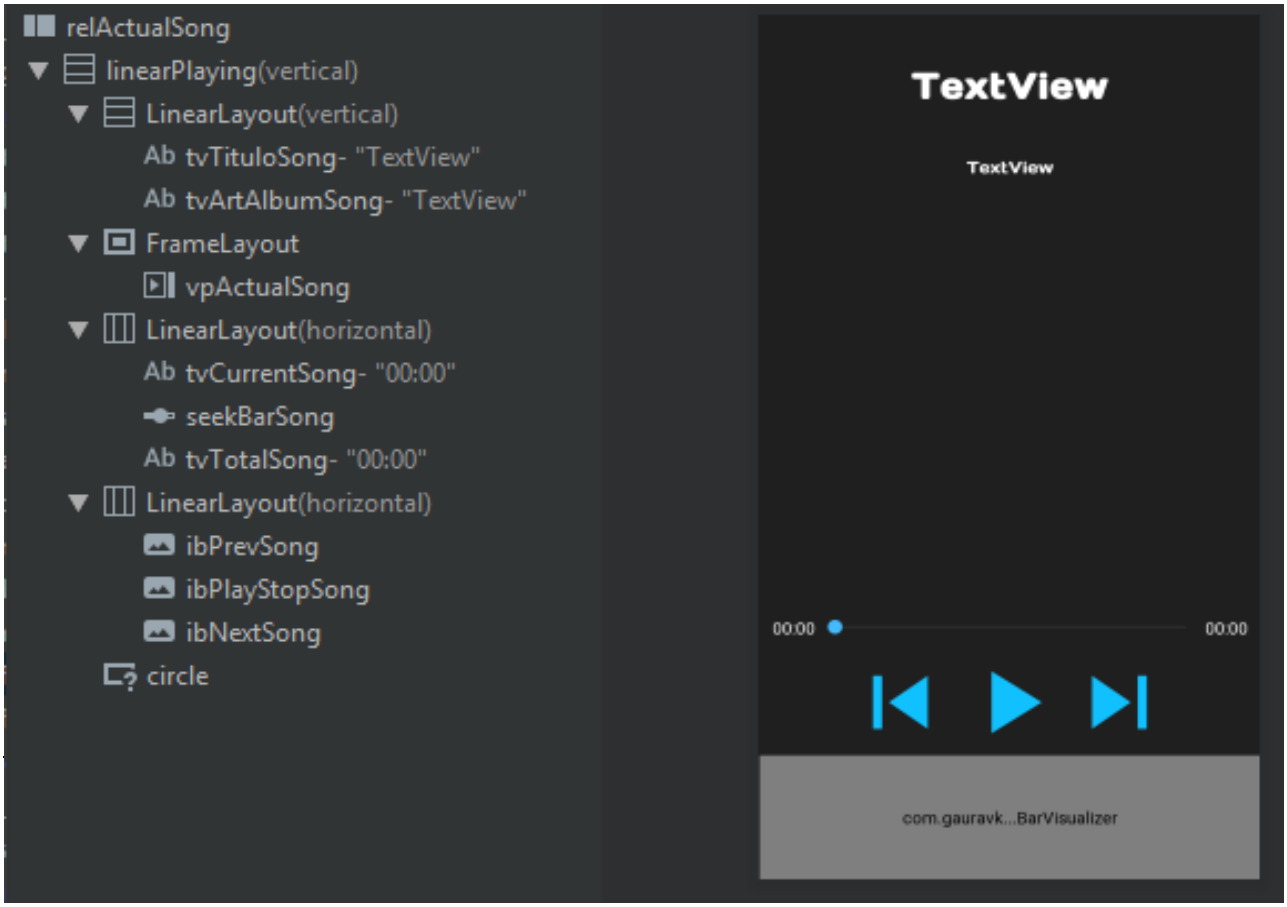
Concluimos el Fragment de la lista de Canciones

f. Fragment: Reproduciendo Cancion

En este Fragment, como ampliación para poder hacer un Slider de canciones, hemos creado una vista para cuando reproduces una canción, mostrar su imagen en el centro siendo este un viewPager que, al deslizar para cambiar de imagen, cambie de canción. Ademas se han agregado mas ítems visuales como una seekBar, titulo(autoscrollable) y álbum/artista de la canción, sus botones de gestión del mediaPlayer y por ultimo, un Audio Visualizer[11] para mostrar mas dinámica y esteticamente la reproducción de la cancion

i. Layout

Al igual que el Fragment anterior, en este tenemos un diseño introducido en un RelativeLayout y distribuido gracias a los LinearLayouts junto con un ViewPager



ii. Codigo

Nada mas instanciar el Fragment, este mismo recibe por parámetro en el constructor la lista de reproducción, el indice de la canción y el Fragment con la lista completa de canciones

Constructor

```
public PlayingFragment(ArrayList<Audio> audioList, int audioIndex, MusicFragment
musicFragment) {
    this.audioList = audioList;
    this.audioIndex = audioIndex;
    this.musicFragment = musicFragment;
}
```

Obviando la parte en la cual inicializamos todos los ítems del layout en el onCreate y el listener de la Seek Bar[12](Al ser exactamente lo mismo que en el Fragment anterior) vamos a proceder a explicar el onViewCreated, donde si que instanciamos e iniciamos varios hilos que manejan la interfaz.
Exactamente igual que en la lista de canciones, procedemos a hacer un Hilo que maneja únicamente la seekBar para que, en tiempo real, avance.

Instancia e inicio del Hilo de Actualizacion de la Seek Bar

```
public void seekBarUpdate() {
    new Thread() {
        @Override
        public void run() {
            try {
                getActivity().runOnUiThread(runnable = new Runnable() {
                    @Override
                    public void run() {

                        if (((MainActivity)getActivity()).player.getMediaPlayer() !=
null) {
                            try {

barraProg.setMax(((MainActivity)getActivity()).player.getMediaPlayer().getDuration() /
1000);
                                int mCurrentPosition =
((MainActivity)getActivity()).player.getMediaPlayer().getCurrentPosition() / 1000;
                                barraProg.setProgress(mCurrentPosition);

currentS.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMedia
Player().getCurrentPosition()));

totalS.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMediaPl
ayer().getDuration()));
                            } catch (Exception ex) {
                            }
                        }
                    }
                });
                //Timeamos la accion con un handler que publica cada 1 segundo el hilo
                handler.postDelayed(this, 1000);
            } catch (Exception ex) {
            }
        }
    }.start();
}
```

[12] Seek Bar: Item que realiza una función parecida a la Barra de Progreso tradicional pero que permite ser manipulada mediante Drag & Move del usuario

Instancia del Listener de la Seek Bar

```
private void handleSeekbar() {
    barraProg.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {

            if (((MainActivity)getActivity()).player.getMediaPlayer() != null &&
fromUser) {
                ((MainActivity)getActivity()).player.getMediaPlayer().seekTo(progress *
1000);

currentS.setText(Times.milliSecondsToTimer(((MainActivity)getActivity()).player.getMedia
Player().getCurrentPosition()));
            }
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
        }
    });
}
```

Igual que la seekBar, el ViewPager que instanciamos, le asignamos un único hilo para que maneje su instancia

Para el slider, como el Recycler View, necesita un adaptador, no lo explicaremos ya que se asume que se conoce dicha clase

```
public class ViewSongAdapter extends PagerAdapter {


    private Context context;
    private LayoutInflater layoutInflater;
    private ArrayList<Audio> audiolist;
    private MusicFragment musicFragment;

    public ViewSongAdapter(Context context, ArrayList<Audio> audiolist, MusicFragment
musicFragment) {
        this.context = context;
        this.audiolist = audiolist;
        this.musicFragment = musicFragment;
    }

    @Override
    public int getCount() {
        return audiolist.size();
    }

    @Override
    public boolean isViewFromObject(@NonNull View view, @NonNull Object object) {
        return view == object;
    }


    @NonNull
    @Override
    public Object instantiateItem(@NonNull ViewGroup container, int position) {

        layoutInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        View view = layoutInflater.inflate(R.layout.song_image_item, null);
        ImageView imageView = (ImageView)view.findViewById(R.id.image);
```

```
imageView.setImageBitmap(musicFragment.getAlbumart(Long.parseLong(audiolist.get(position
).getCaratula())));
        ViewPager vp = (ViewPager) container;
        vp.addView(view, 0);


        return view;


    }

    @Override
    public void destroyItem(@NonNull ViewGroup container, int position, @NonNull Object
object) {

        ViewPager vp = (ViewPager) container;
        View view = (View) object;

        vp.removeView(view);

    }

}
```

Instancia del Hilo que genera el Slider y su Listener

```
public void loadSlider() {
    new Thread() {
        @Override
        public void run() {
            try {
                getActivity().runOnUiThread(runnable = new Runnable() {
                    @Override
                    public void run() {
                        ViewSongAdapter viewSongAdapter = new
ViewSongAdapter(getContext(), audioList, musicFragment);
                        viewPager.setAdapter(viewSongAdapter);
                        viewPager.setCurrentItem(audioIndex, false);
                        viewPager.setScrollBarFadeDuration(500);
                        viewPager.setOffscreenPageLimit(3);
                        viewPager.setPageTransformer(false, new FadePageTransformer());
                        if (audioIndex < 0) {
                            audioIndex = 0;
                        }
                        artalbum.setText(String.format("%s | %s",
audioList.get(audioIndex).getArtist(), audioList.get(audioIndex).getAlbum()));
                        //Listener del viewPager para el cambio de cancion al hacer un
drag
                        viewPager.addOnPageChangeListener(new
ViewPager.OnPageChangeListener() {
                            @Override
                            public void onPageScrolled(int position, float
positionOffset, int positionOffsetPixels) {

                            }

                            @Override
                            public void onPageSelected(int position) {
                                musicFragment.playAudio(position, false);
                                titulo.setText(audioList.get(position).getTitle());
                                artalbum.setText(String.format("%s | %s",
audioList.get(position).getArtist(), audioList.get(position).getAlbum()));
                            }

                            @Override
                            public void onPageScrollStateChanged(int state) {
```

```
                            }
                        });
                    try {
                        int audioSessionId =
((MainActivity)getActivity()).player.getMediaPlayer().getAudioSessionId();
                        if (audioSessionId != -1)
                            visualizer.setAudioSessionId(audioSessionId);
                    }catch(Exception ex){}
                    }
                });

barraProg.getProgressDrawable().setColorFilter(getResources().getColor(R.color.whitebox)
, PorterDuff.Mode.SRC_ATOP);

            } catch (Exception ex) {
            }
        }
    }.start();
}
```

Como anteriormente hemos hecho con la Seek Bar, instanciamos un nuevo Hilo que trabaja en el Hilo de la Interfaz para gestionar que, mientras este en proceso nuestro servicio, muestre en pantalla la informacion correcta

Hilo que maneja la información correcta en pantalla, incluyendo los botones

```
public void isPlayingSong() {
    new Thread() {public void run() {
            try {
                getActivity().runOnUiThread(runnable = new Runnable() {
                    @Override
                    public void run() {
                        if (((MainActivity)getActivity()).player.getMediaPlayer() !=
null) {
                            if
(((MainActivity)getActivity()).player.getMediaPlayer().isPlaying()) {
                                play.setImageResource(R.drawable.bt_stop_large);
                            } else {
                                play.setImageResource(R.drawable.bt_play_large);
                            }
                        }
                        StorageUtil storageUtil = new StorageUtil(getContext());
                        audioIndex = storageUtil.loadAudioIndex();
                        if (viewPager.getCurrentItem() != audioIndex) {
                            viewPager.setCurrentItem(audioIndex, false);
                        }
                        titulo.setSelected(true);
                    }
                });handler.postDelayed(this, 200); } catch (Exception ex) {}}}.start();}
```

Para mostrar dicha información hemos creado una animación personalizada que hacia un Fade In-Fade Out

```
public class FadePageTransformer implements ViewPager.PageTransformer {
    public void transformPage(View view, float position) {
        view.setTranslationX(view.getWidth() * -position);
        if (position <= -1.0F || position >= 1.0F) {
            view.setAlpha(0.0F);
        } else if (position == 0.0F) {
            view.setAlpha(1.0F);
        } else {
            // La posicion esta dentro de -1.0F & 0.0F OR 0.0F & 1.0F
            view.setAlpha(1.0F - Math.abs(position));
        }
    }

}
```

Una vez explicado el Slider, pasamos a explicar brevemente los botones, los cuales tienen la misma acción que en el Fragment anterior, exceptuando que agregamos que simplemente cambia la imagen del slider

Listener de los botones de control del Media Player

```
public void changeSong() {
    next.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                ((MainActivity)getActivity()).player.skipToNext();
                StorageUtil storageUtil = new StorageUtil(getContext());
                audioIndex = storageUtil.loadAudioIndex();
                viewPager.setCurrentItem(audioIndex, false);

            } catch (Exception ex) {
                Toast.makeText(getContext(), "Ultima Cancion", Toast.LENGTH_SHORT);
            }
        }});
    previous.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                ((MainActivity)getActivity()).player.skipToPrevious();
                StorageUtil storageUtil = new StorageUtil(getContext());
                audioIndex = storageUtil.loadAudioIndex();
                viewPager.setCurrentItem(audioIndex, false);
            } catch (Exception ex) {
                Toast.makeText(getContext(), "Primera Cancion", Toast.LENGTH_SHORT); }
        }});
    play.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                if (((MainActivity)getActivity()).player.getMediaPlayer().isPlaying()) {
                    ((MainActivity)getActivity()).player.pauseMedia();
                } else {
                    ((MainActivity)getActivity()).player.resumeMedia();
                }
            }catch(Exception ex){
                ((MainActivity)getActivity()).player.getMediaPlayer().reset();}
        }});}
```

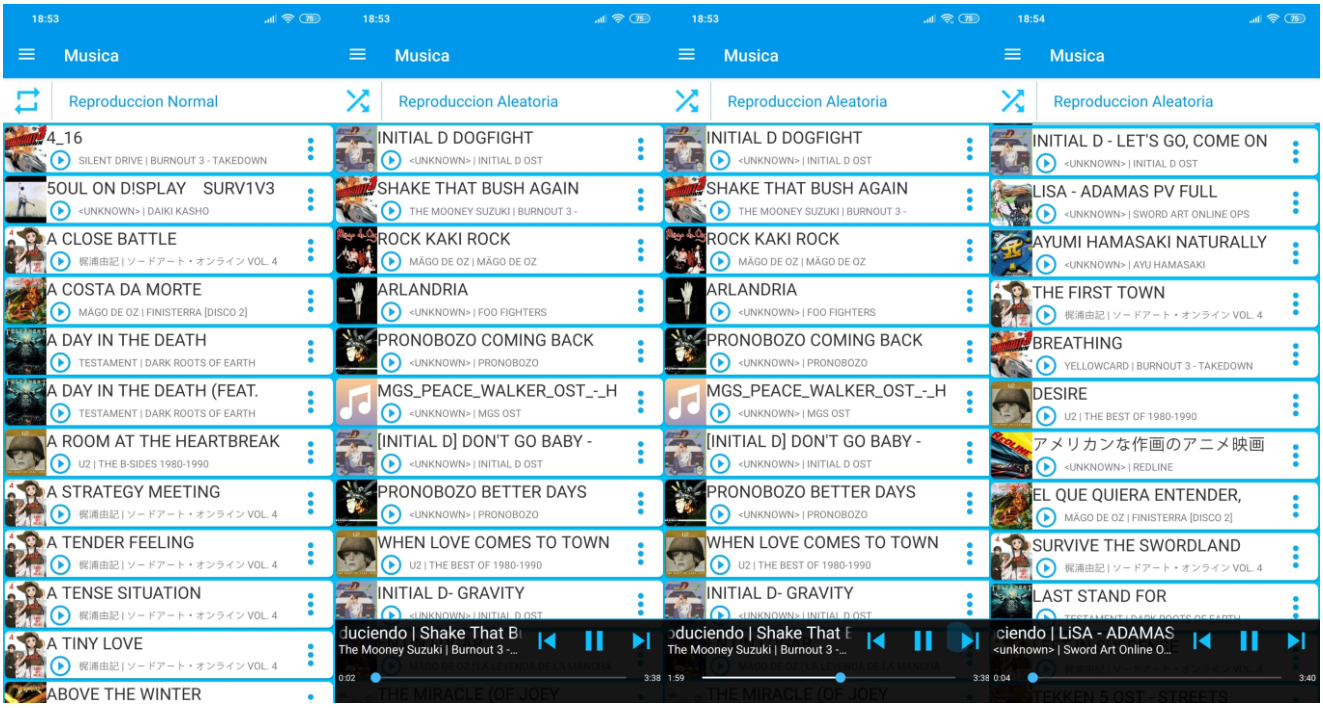
Por ultimo en el mismo onDestroy del Fragment, desvinculamos los Hilos del handler y le hacemos un release al Audio Visualizer

onDestroy[13]

```
public void onDestroy() {
    super.onDestroy();
    handler = new Handler();
    try {
        if (visualizer != null) {
            visualizer.release();
        }
    }catch(Exception ex){}
    handler.removeCallbacksAndMessages(runnable);
    musicFragment.setPlaying(false);

}
```

[13] onDestroy(): Metodo que se ejecuta al final del ciclo de vida del Fragment

g. Capturas "In Action"



2. Reproductor de Video

a. Objeto Video

Creacion de un Objeto Video donde almacenaremos en memoria los videos que encontremos en el sistema de archivos

Parametros y Constructor

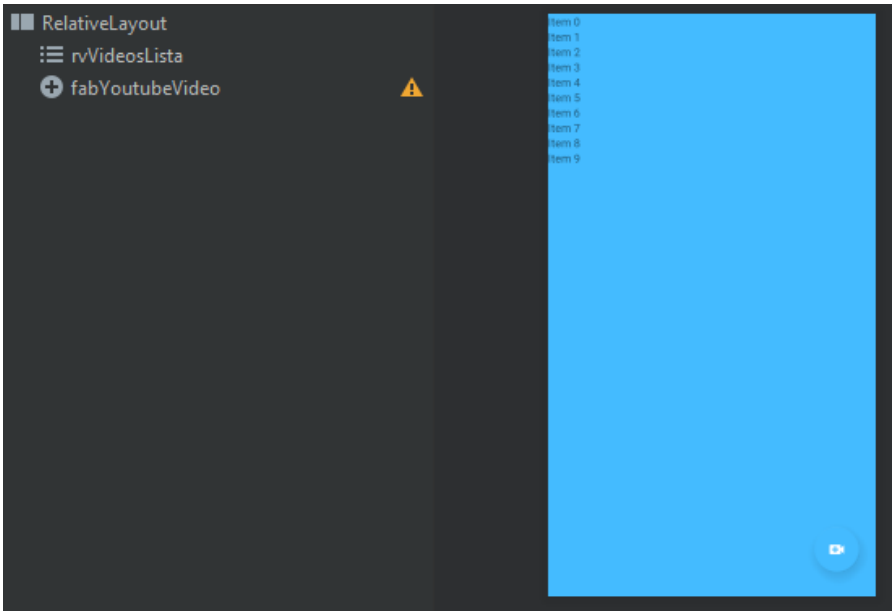
```
public class Video {
    String path, titulo, duracion;
    Bitmap thumb;
    String width, height, orientation;
    boolean selected;

    public Video(String path, Bitmap thumb, String titulo, String duracion, String
width, String height, String orientation, boolean selected) {
        this.path = path;
        this.thumb = thumb;
        this.titulo = titulo;
        this.duracion = duracion;
        this.width = width;
        this.height = height;
        this.orientation = orientation;
        this.selected = selected;
    }
```

b. Fragment: Lista de Videos

Como hemos explicado antes con la música, se nos presenta una lista de videos extraidos del sistema de archivos del dispositivo.

i. Layout

A diferencia del Reproductor de música, mostramos una Interfaz bastante mas descargada la cual se compone de un RelativeLayout incluyendo solamente un RecyclerView y un Floating Action Button[14]



ii. Codigo

Nada mas instanciar el Fragment, tenemos como parámetros iniciales que lo usaremos a lo largo del mismo para manejar los objetos y los datos a mostrar

Parametros de la Clase

```
public class VideoPlayerFragment extends Fragment {
    private ArrayList<Video> videosList;
    private RecyclerView recyclerView;
    private VideoAdapter adapter;
    private View root;
    private FloatingActionButton ytVideo;
    public boolean playing;
```

Nada mas empezar, en la instancia de la UI en onCreateView, manejamos la actividad principal a través de Flags[15] para modificar la visualización en pantalla

onCreate y modificación de Flags

```
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup
container,
                         @Nullable Bundle savedInstanceState) {
getActivity().setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);//Portra
it
    ((MainActivity)getActivity()).setDrawerLocked(false);
```

[14]Floating Action Button: Boton circular flotante que dispara una acción principal de la Interfaz, en nuestro caso, del Fragment y que se activa con la acción de Tap del usuario

[15] Flags: "Bandera" que hace referencia a uno o mas bits en los que se almacena una variable de configuración del programa

```
    getActivity().
getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);//Limpiamos
FullScreen
    ((MainActivity)getActivity()).getSupportActionBar().show();
getActivity().getWindow().addFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN;
//Modo Ventana
    root = inflater.inflate(R.layout.video_player_fragment, container, false);
    playing = false;
    return root;
}
```

Al llegar al onActivityCreated, simplemente instanciamos los ítems de la UI y con una AsyncTask, cargamos la lista de Videos.

Dicha AsyncTask realiza en background la llamada al método que recoge la lista de videos del dispositivo

AsyncTask videoLoader

```
public class videoLoader extends AsyncTask<Void, Void, Void>{


    @Override
    protected Void doInBackground(Void... voids) {
        fetchVideos();

        return null;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        initRecycler();
    }
}
```

El metodo que recoge todos los videos, con similar cuerpo que en la música, utilizando un cursor, almacenándolo en un ArrayList

Fetch de todos los videos

```
public void fetchVideos(){
    videosList = new ArrayList<>();
    Uri uri;
    Cursor cursor;
    int column_index_data, thumb;
    String absolutePathImage = null;
    uri = MediaStore.Video.Media.EXTERNAL_CONTENT_URI;
    String[] projection = {MediaStore.MediaColumns.DATA,
            MediaStore.Video.Media.BUCKET_DISPLAY_NAME,
            MediaStore.Video.Media._ID,
            MediaStore.Video.Thumbnails.DATA,
            MediaStore.Video.Media.DURATION,
            MediaStore.Video.Media.TITLE,
            MediaStore.Video.Media.WIDTH,
            MediaStore.Video.Media.HEIGHT};
    String orderBy = MediaStore.Images.Media.DATE_TAKEN;
    cursor = getContext().getContentResolver().query(uri, projection, null, null,
orderBy + " DESC");
    column_index_data = cursor.getColumnIndexOrThrow(MediaStore.MediaColumns.DATA);
    cursor.moveToFirst();

    while(cursor.moveToNext()){
        absolutePathImage = cursor.getString(column_index_data);
        String title =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.TITLE));
        String height =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.HEIGHT));
```

```
        String width =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.WIDTH));
        String duracion =
cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Video.Media.DURATION));
        int id =
cursor.getInt(cursor.getColumnIndexOrThrow(MediaStore.Video.Media._ID));
        BitmapFactory.Options options=new BitmapFactory.Options();
        options.inSampleSize = 1;
        ContentResolver crThumb = getContext().getContentResolver();
        Bitmap curThumb = MediaStore.Video.Thumbnails.getThumbnail(crThumb, id,
MediaStore.Video.Thumbnails.MICRO_KIND, options);

        Video video = new Video();
        video.setPath(absolutePathImage);
        video.setTitulo(title);
        video.setHeight(height);
        video.setWidth(width);
        video.setThumb(curThumb);
        video.setDuracion(Times.milliSecondsToTimer(Long.parseLong(duracion)));
        videosList.add(video);

    }
}
```

En cambio la nueva funcionalidad, respecto a la música, es el poder reproducir videos de YouTube ingresando su URL compartida comenzando por https://youtu.be/idVideo o https://youtube.x/watch?v

Listener del FAB

```
ytVideo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        alertDialog();
    }
});
```

La acción que despliega el dialogo que pregunta al usuario, aportando un editText como input, que URL quiere reproducir, la realiza el Floating Action Button que hemos explicado antes en el layout

Metodo que despliega el dialogo

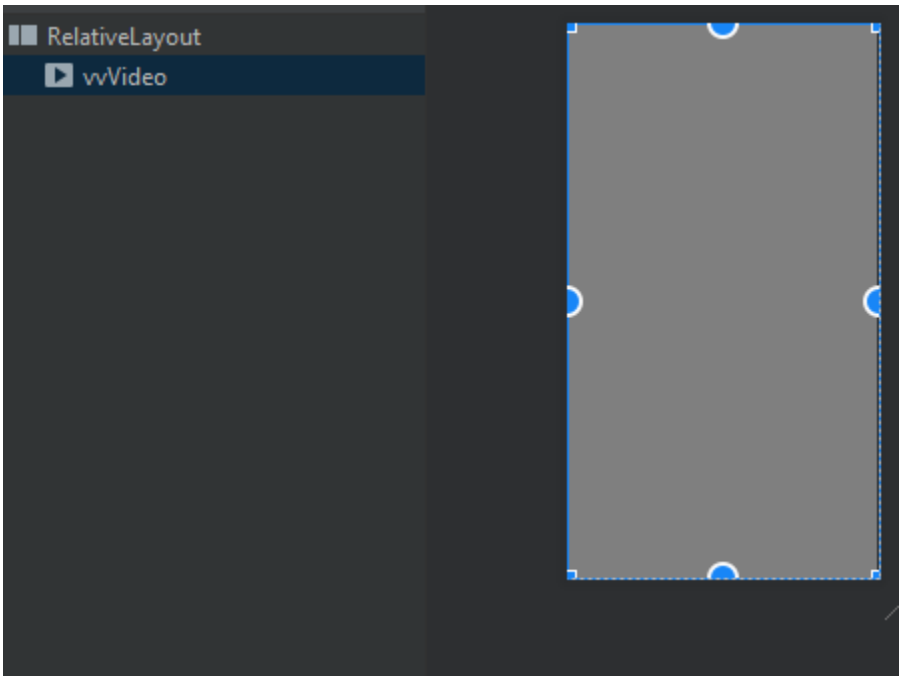
```
private void alertDialog() {
    AlertDialog.Builder dialog = new AlertDialog.Builder(getContext());
    dialog.setMessage("Introduce la URL del Video a Reproducir");
    dialog.setTitle("Alerta");
    // Seteamos el input
    final EditText input = new EditText(getContext());
    // Especificamos el tipo de input
    input.setInputType(InputType.TYPE_CLASS_TEXT);
    dialog.setView(input);
    dialog.setPositiveButton("Confirmar",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,
                                    int which) {
                    String path = input.getText().toString().trim();
                    if(path.isEmpty()){
                        Toast.makeText(getContext(),"URL incorrecta",
Toast.LENGTH_SHORT).show();
                    }else {
                        getActivity().
getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
getActivity().getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREE
N);
getActivity().setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                        ((MainActivity)getActivity()).setDrawerLocked(true);
                        ((MainActivity)getActivity()).getSupportActionBar().hide();
                        YoutubeFragment youtubeFragment = new
YoutubeFragment(input.getText().toString());
```

```
getFragmentManager().beginTransaction().addToBackStack(null).replace(R.id.nav_host_fragment, youtubeFragment).commit();
                    }
                }
            });

    dialog.setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(getContext(),"Reproduccion Cancelada",
Toast.LENGTH_SHORT).show();
        }
    });
    AlertDialog alertDialog = dialog.create();
    alertDialog.show();
}
```

c. *Fragment: Video Player*

i. Layout

Disposicion lo mas sencilla posible compuesta de un simple Relative Layout con un Video View



ii. Codigo

Abreviando la explicación, simplemente vamos a mostrar el constructor y el listener que reproduce el video, asi como el onDestroy, donde tratamos las Flags de la App

Parametros y Constructor

```
public class PlayingFragment extends Fragment {
    private VideoView videoView;
    private String path;
    private VideoPlayerFragment videoPlayerFragment;
    private View root;
    /**
     * Constructor
     * @param path ruta del video
     * @param videoPlayerFragment fragment de la playlist
     */
    public PlayingFragment(String path, VideoPlayerFragment videoPlayerFragment) {
        this.path = path;
        this.videoPlayerFragment = videoPlayerFragment;
    }
```

Como hemos dicho tratamos las Flags

onCreate(Cambiamos la ventana a FullScreen)

```
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    ((MainActivity)
getActivity()).getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

onDestroy(Volvemos a poner la ventana en Portrait y modo Ventana)

```
public void onDestroy() {
    super.onDestroy();
    videoPlayerFragment.playing = false;
    ((MainActivity)getActivity()).getSupportActionBar().show();
    getActivity(). getWindow().clearFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

getActivity().getWindow().addFlags(WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN)
;
}
```

Pasando a los listeners, mostramos para cuando complete el video haga un onBackPressed en el Main y un onPrepared para que inicie el video cuando cargue y si, el video es en panorámico, con las Flags modificamos la orientación de la pantalla

Listener del Video View

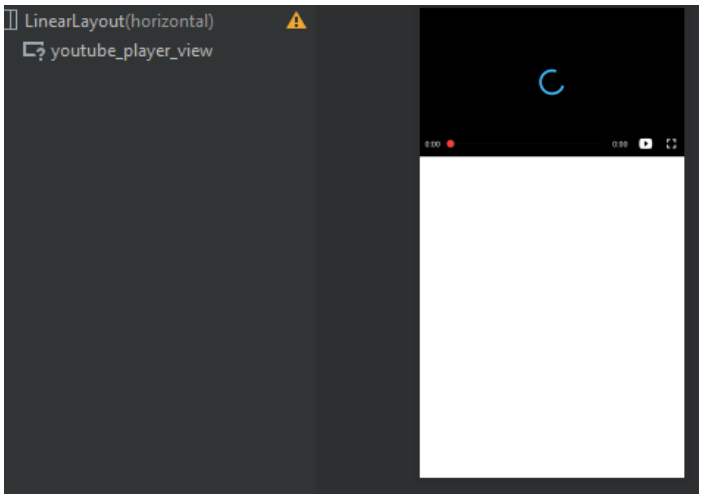
```
videoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mediaPlayer) {
        ((MainActivity)getActivity()).onBackPressed();
    }
});
videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
    @Override
    public void onPrepared(MediaPlayer mediaPlayer) {
        videoView.start();
        mediaPlayer.setOnVideoSizeChangedListener(new
MediaPlayer.OnVideoSizeChangedListener() {
            @Override
            public void onVideoSizeChanged(MediaPlayer mp, int width, int height) {
```

```
                MediaController mediaController = new MediaController(getContext());
                videoView.setMediaController(mediaController);
                mediaController.setAnchorView(videoView);
                if(mp.getVideoWidth() > mp.getVideoHeight()){
getActivity().setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                }
            }
        });
}});
```

d. Fragment: Youtube Player

i. Layout

Aun mas simple que el Fragment anterior, simplemente se resume en un linear Layout con una Vista "Youtube Player"



ii. Codigo

En este Fragment, nos limitaremos a explicar el Constructor, los parámetros y como iniciamos la reproducción del video

Constructor y parámetros iniciales

```
public class YoutubeFragment extends Fragment {
    private View root;
    private YouTubePlayerView youTubePlayerView;
    private String videourl;
    /**
     * Constructor en el que tratamos las posibles URLs que se pueden obtener al
compartir un enlace
     * @param videourl url del video
     */
    public YoutubeFragment(String videourl) {
        if(videourl.contains("youtu.be/")){
            try {
                this.videourl = videourl.substring(videourl.indexOf("be/")+3,
videourl.indexOf("?"));
            }catch(Exception ex){
                this.videourl = videourl.substring(videourl.indexOf("be/")+3);
            }
        }else if(videourl.contains("?v=")){
            try {
                this.videourl = videourl.substring(videourl.indexOf("?v=")+3,
videourl.indexOf("?"));
            }catch(Exception ex){
                this.videourl = videourl.substring(videourl.indexOf("?v=")+3);
            }
```

```
        }
    }
```

El parámetro YouTubePlayerView[16] lo creamos gracias a la implementación de dependencias de terceros ya que no podemos usar la propia de Google® ya que está desactualizada y totalmente obsoleta. Dicho esto, el video se inicializa con el siguiente método

onActivityCreated

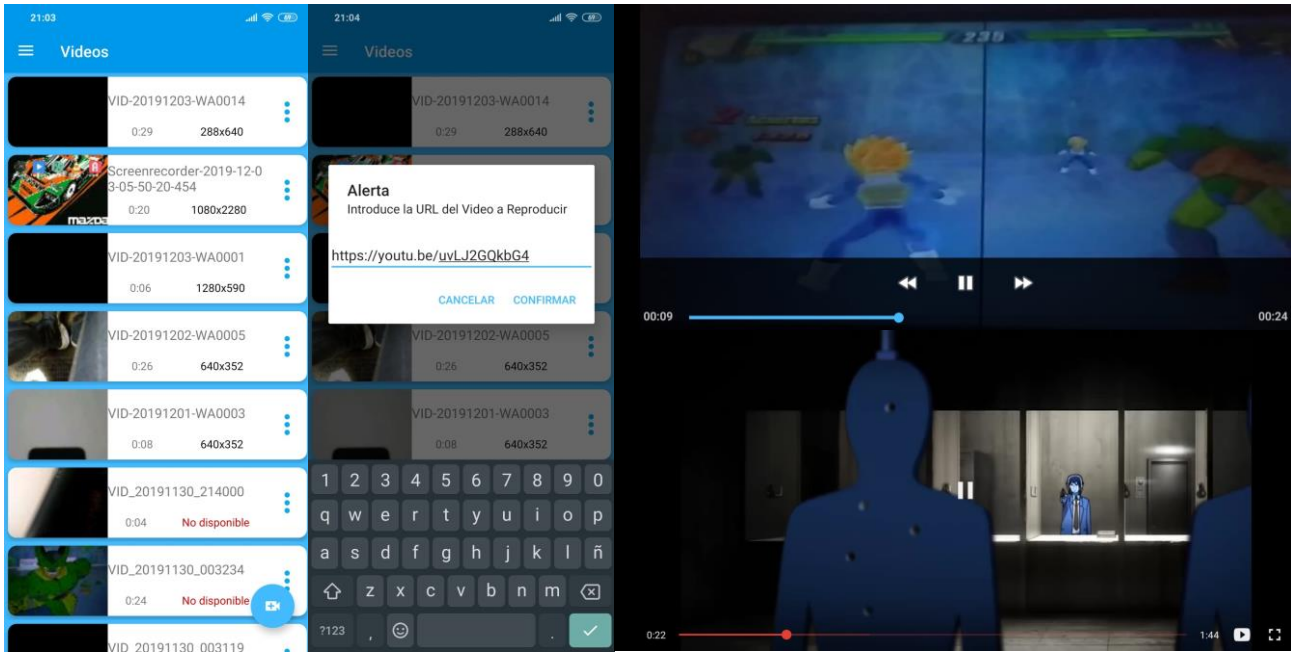
```
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    youTubePlayerView = root.findViewById(R.id.youtube_player_view);
    getLifecycle().addObserver(youTubePlayerView);
    youTubePlayerView.addYouTubePlayerListener(new AbstractYouTubePlayerListener() {
        @Override
        public void onReady(@NonNull YouTubePlayer youTubePlayer) {
            youTubePlayer.loadVideo(videourl, 0);
        }
    });
}
```

Y al hacer el onDestroy del Fragment, debemos hacerle un release al player

onDestroy

```
public void onDestroy() {
    super.onDestroy();
    youTubePlayerView.release();
}
```

e. Capturas "In Action"



[16] Android-Youtube Player: https://github.com/PierfrancescoSoffritti/android-youtube-player

3. Sensores

En este punto, donde vamos a explicar el funcionamiento y detección de los sensores en nuestra APP, se han diseñado varias animaciones para poder interactuar y ganar usabilidad para el propio usuario

a. Animacion: Movimiento de Objeto por Acelerometro

La primera animación consta de un objeto que podemos mover gracias al Acelerometro en un espacio definido gracias a la implementación de View en la clase. Lo primero que hacemos al instanciar un objeto de este tipo, es crear las variables de la clase Paint y setearle el color(Para el objeto)

Constructor y parametros de inicio

```
public class AnimacionBall extends View {

    private static final int CIRCLE_RADIUS = 10; //Pixeles

    private Paint mPaint;
    private int x;
    private int y;
    private int viewWidth;
    private int viewHeight;

    public AnimacionBall(Context context) {
        super(context);
        mPaint = new Paint();
        mPaint.setColor(getResources().getColor(android.R.color.holo_blue_light));
    }
```

Una vez instanciado el objeto, desde el propio Fragment que adquiere los datos del Acelerometro, vamos llamando al metodo que plasma cada movimiento en la propia interfaz

Metodo onSensorEvent que cambia la posición el objeto

```
public void onSensorEvent (SensorEvent event) {
    x = x - (int) event.values[0];
    y = y + (int) event.values[1];
    //Nos aseguramos de que dibujamos dentro del espacio establecido
    //Entonces los maximos valores donde podremos dibujar son:
    if (x <= 0 + CIRCLE_RADIUS) {
        x = 0 + CIRCLE_RADIUS;
    }
    if (x >= viewWidth - CIRCLE_RADIUS) {
        x = viewWidth - CIRCLE_RADIUS;
    }
    if (y <= 0 + CIRCLE_RADIUS) {
        y = 0 + CIRCLE_RADIUS;
    }
    if (y >= viewHeight - CIRCLE_RADIUS) {
        y = viewHeight - CIRCLE_RADIUS;
    }
}
```

Y para dibujar el objeto, llamamos al método implementado onDraw

onDraw

```
protected void onDraw(Canvas canvas) {
    canvas.drawCircle(x, y, CIRCLE_RADIUS, mPaint);
    //Necesitamos llamar al invalidador para dibujar constantemente
    invalidate();
}
```

b. *Animacion: Vista de Avión*

Para jugar con el sensor de Rotacion Vectorial, hemos tratado de simular la visión de un avión, simulando con la rotación del móvil, los movimientos de Cabeceo, Balanceo y Guiñada propio de los aviones.

Dicha animación se compone de dos clases:
Clase IndicadorAltura, genera el dibujo ambiental, simulando un horizonte y un punto de mira para crear sensación de movimiento al interactuar

Esta misma clase, se inicializa con parámetros estáticos con código de colores para la visión y la cantidad de grados de visión que tendrá el usuario

Parametros de la clase IndicadorAltura

```
public class IndicadorAltura extends View {
    //Mostrar FPSs en el Log
    private static final boolean LOG_FPS = false;
    //Colores del escenario
    private static final int SKY_COLOR = Color.parseColor("#36B4DD");
    private static final int EARTH_COLOR = Color.parseColor("#865B4B");
    private static final int MIN_PLANE_COLOR = Color.parseColor("#E8D4BB");
    private static final float TOTAL_VISIBLE_PITCH_DEGREES = 45 * 2; // 45

    private final PorterDuffXfermode mXfermode;
    private final Paint mBitmapPaint;
    private final Paint mEarthPaint;
    private final Paint mPitchLadderPaint;
    private final Paint mMinPlanePaint;
    private final Paint mBottomPitchLadderPaint;

    // Bitmaps creados que llamaremos a continuacion
    private Bitmap mSrcBitmap;
    private Canvas mSrcCanvas;
    private Bitmap mDstBitmap;

    private int mWidth;
    private int mHeight;

    private float mPitch = 0; // Grados
    private float mRoll = 0; // Grados, rotacion a la izquierda es positivo
```

Nada mas instanciar el objeto, se instancian los Paints de cada elemento del dibujo

Constructor de la clase IndicadorAltura

```
public IndicadorAltura(Context context, AttributeSet attrs) {
    super(context, attrs);

    mXfermode = new PorterDuffXfermode(PorterDuff.Mode.SRC_IN);//Crea la esfera de
```

```
vision
    mBitmapPaint = new Paint();//Paint del cielo
    mBitmapPaint.setFilterBitmap(false);

    mEarthPaint = new Paint();//Paint de la tierra
    mEarthPaint.setAntiAlias(true);
    mEarthPaint.setColor(EARTH_COLOR);

    mPitchLadderPaint = new Paint();//Paint del trazado que indica el cabeceo por encima
del horizonte
    mPitchLadderPaint.setAntiAlias(true);
    mPitchLadderPaint.setColor(Color.WHITE);
    mPitchLadderPaint.setStrokeWidth(3);

    mBottomPitchLadderPaint = new Paint();//Paint del trazado que indica el cabeceo por
debajo del horizonte
    mBottomPitchLadderPaint.setAntiAlias(true);
    mBottomPitchLadderPaint.setColor(Color.WHITE);
    mBottomPitchLadderPaint.setStrokeWidth(3);
    mBottomPitchLadderPaint.setAlpha(128);

    mMinPlanePaint = new Paint();//Paint del trazado que indica la Guiñada
    mMinPlanePaint.setAntiAlias(true);
    mMinPlanePaint.setColor(MIN_PLANE_COLOR);
    mMinPlanePaint.setStrokeWidth(5);
    mMinPlanePaint.setStyle(Paint.Style.STROKE);
}
```

Para poder crear el bitmap, hemos creado un método que, según el tamaño, lo redimensiona al tamaño requerido

Generacion del Bitmap del Entorno

```
private Bitmap getSrc() {
    if (mSrcBitmap == null) {
        mSrcBitmap = Bitmap.createBitmap(mWidth, mHeight, Bitmap.Config.ARGB_8888);
        mSrcCanvas = new Canvas(mSrcBitmap);
    }
    Canvas canvas = mSrcCanvas;

    float centerX = mWidth / 2;
    float centerY = mHeight / 2;

    // Background
    canvas.drawColor(SKY_COLOR);

    // Guarda la instancia sin ninguna rotacion
    // por ende podemos revertirla y arreglar problemas
    canvas.save();

    // Orientamos el horizonte para alinear el cabeceo y balanceo
    canvas.rotate(mRoll, centerX, centerY);
    canvas.translate(0, (mPitch / TOTAL_VISIBLE_PITCH_DEGREES) * mHeight);

    // Dibujamos el horizonte para dar sensacion de no haber limite
    // para tener en cuenta el cabeceo del avion
    canvas.drawRect(-mWidth, centerY, mWidth * 2, mHeight * 2, mEarthPaint);

    // Dibujamos una linea blanca en el horizonte
    float ladderStepY = mHeight / 12;
    canvas.drawLine(-mWidth, centerY, mWidth * 2, centerY, mPitchLadderPaint);
    for (int i = 1; i <= 4; i++) {
        float y = centerY - ladderStepY * i;
        float width = mWidth / 8;
        canvas.drawLine(centerX - width / 2, y, centerX + width / 2, y,
mPitchLadderPaint);
```

```
    }

    // Dibujamos una serie de alineaciones para medir el cabeceo
    float bottomLadderStepX = mWidth / 12;
    float bottomLadderStepY = mWidth / 12;
    canvas.drawLine(centerX, centerY, centerX - bottomLadderStepX * 3.5f, centerY
            + bottomLadderStepY * 3.5f, mBottomPitchLadderPaint);
    canvas.drawLine(centerX, centerY, centerX + bottomLadderStepX * 3.5f, centerY
            + bottomLadderStepY * 3.5f, mBottomPitchLadderPaint);
    for (int i = 1; i <= 3; i++) {
        float y = centerY + bottomLadderStepY * i;
        canvas.drawLine(centerX - bottomLadderStepX * i, y, centerX + bottomLadderStepX
* i, y,
                mBottomPitchLadderPaint);
    }

    // Restauramos el dibujo inicial
    canvas.restore();

    // Dibujamos el punto central
    canvas.drawPoint(centerX, centerY, mMinPlanePaint);

    // Dibujamos media circunsferencia
    float minPlaneCircleRadiusX = mWidth / 6;
    float minPlaneCircleRadiusY = mHeight / 6;
    RectF wingsCircleBounds = new RectF(centerX - minPlaneCircleRadiusX, centerY
            - minPlaneCircleRadiusY, centerX + minPlaneCircleRadiusX, centerY +
minPlaneCircleRadiusY);
    canvas.drawArc(wingsCircleBounds, 0, 180, false, mMinPlanePaint);

    // Dibujamos las alas del avion
    float wingLength = mWidth / 6;
    canvas.drawLine(centerX - minPlaneCircleRadiusX - wingLength, centerY, centerX
            - minPlaneCircleRadiusX, centerY, mMinPlanePaint);
    canvas.drawLine(centerX + minPlaneCircleRadiusX, centerY, centerX +
minPlaneCircleRadiusX
            + wingLength, centerY, mMinPlanePaint);

    // Dibujamos una linea vertical
    canvas.drawLine(centerX, centerY + minPlaneCircleRadiusY, centerX, centerY
            + minPlaneCircleRadiusY + mHeight / 3, mMinPlanePaint);

    return mSrcBitmap;
}
```

Posteriormente generamos otro método que crea un pequeño ovalo, uniendo con ello los elementos que conforman el avión, tal como su línea vertical y sus dos trazas (Alas)horizontales

Metodo con el que generamos la pieza restante del avión

```
private Bitmap getDst() {
    if (mDstBitmap == null) {
        mDstBitmap = Bitmap.createBitmap(mWidth, mHeight, Bitmap.Config.ARGB_8888);
        Canvas c = new Canvas(mDstBitmap);
        c.drawColor(Color.TRANSPARENT);

        Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
        p.setColor(Color.RED);
        c.drawOval(new RectF(0, 0, mWidth, mHeight), p);
    }
    return mDstBitmap;
}
```

Y, al igual que la animación anterior, implementamos el método onDraw, que muestra en pantalla el resultado

Metodo que muestra en pantalla el resultado

```
protected void onDraw(Canvas canvas) {
    if (LOG_FPS) {
        countFps();
    }

    Bitmap src = getSrc();
    Bitmap dst = getDst();

    int sc = saveLayer(canvas);
    canvas.drawBitmap(dst, 0, 0, mBitmapPaint);
    mBitmapPaint.setXfermode(mXfermode);
    canvas.drawBitmap(src, 0, 0, mBitmapPaint);
    mBitmapPaint.setXfermode(null);

    canvas.restoreToCount(sc);
}
```

Por ultimo pero no menos importante, agregamos un método para salvar el dibujo actual del Frame

Metodo que guarda el frame actual

```
private int saveLayer(Canvas canvas) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        return canvas.saveLayer(0, 0, mWidth, mHeight, null);
    } else {

        return canvas.saveLayer(0, 0, mWidth, mHeight, null, Canvas.ALL_SAVE_FLAG);
    }
}
```

Clase **Orientation**, implementa SensorEventListener. Controla el estado del sensor.
Nada mas crear el objeto, se nos presenta una interfaz con el método personal onOrientationChanged, que aplica la orientación actual

```
public class Orientation implements SensorEventListener {

    public interface Listener {
        void onOrientationChanged(float pitch, float roll);
    }
```

Con la clase instanciada se crea, a través del parámetro activity que le hemos pasado, el sensor Manager, el propio sensor y obtenemos la ventana

Constructor de la clase

```
public Orientation(Activity activity) {
    mWindowManager = activity.getWindow().getWindowManager();
    mSensorManager = (SensorManager) activity.getSystemService(Activity.SENSOR_SERVICE);

    //Puede ser null si el sensor no esta disponible
    mRotationSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
}
```

Dentro de nuestra clase personal que hará la función de Listener, recogemos todos los métodos implementados por la interfaz SensorEventListener y agregamos los nuestros propios como startListening y StopListening

Metodos necesarios para el Listener

```
public void startListening(Listener listener) {
    if (mListener == listener) {
        return;
    }
    mListener = listener;
    if (mRotationSensor == null) {
        System.out.println("Sensor de rotacion vectorial no disponible");
        return;
    }
    mSensorManager.registerListener(this, mRotationSensor, SENSOR_DELAY_MICROS);
}

public void stopListening() {
    mSensorManager.unregisterListener(this);
    mListener = null;
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    if (mLastAccuracy != accuracy) {
        mLastAccuracy = accuracy;
    }
}

@Override
public void onSensorChanged(SensorEvent event) {
    if (mListener == null) {
        return;
    }
    if (mLastAccuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {
        return;
    }
    if (event.sensor == mRotationSensor) {
        updateOrientation(event.values);
    }
}
```

Para acabar con la clase, una vez recogido todos los datos del sensor, aplicamos al avión la rotación necesaria para dar la sensación de movimiento al usuario, con ello conseguimos una simulación fluida y vistosa

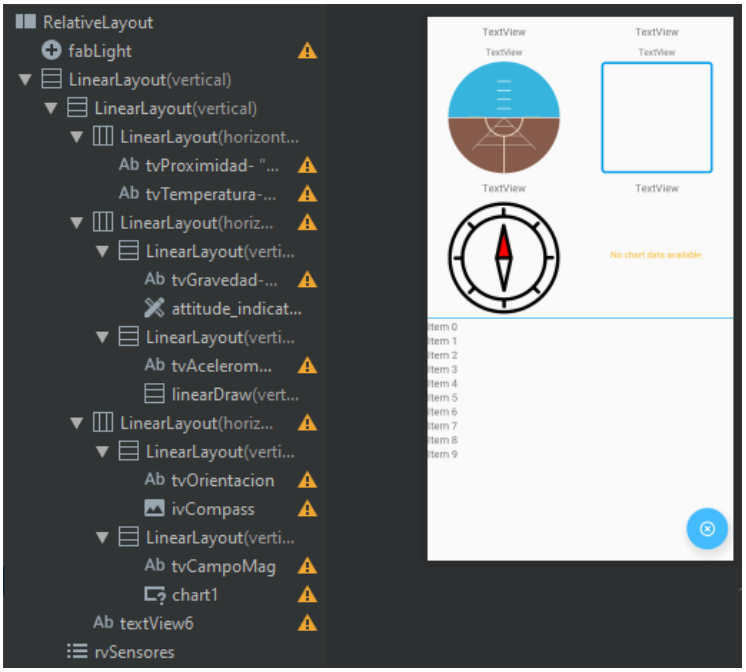
Metodo que actualiza la rotación del avión

```
private void updateOrientation(float[] rotationVector) {
    float[] rotationMatrix = new float[9];
    SensorManager.getRotationMatrixFromVector(rotationMatrix, rotationVector);

    final int worldAxisForDeviceAxisX;
    final int worldAxisForDeviceAxisY;

    // Remapeamos los ejes si el dispositivo tiene un panel de control
    // y ajustamos la matriz de rotacion al dispositivo
    switch (mWindowManager.getDefaultDisplay().getRotation()) {
        case Surface.ROTATION_0:
        default:
            worldAxisForDeviceAxisX = SensorManager.AXIS_X;
            worldAxisForDeviceAxisY = SensorManager.AXIS_Z;
            break;
        case Surface.ROTATION_90:
```

```
            worldAxisForDeviceAxisX = SensorManager.AXIS_Z;
            worldAxisForDeviceAxisY = SensorManager.AXIS_MINUS_X;
            break;
        case Surface.ROTATION_180:
            worldAxisForDeviceAxisX = SensorManager.AXIS_MINUS_X;
            worldAxisForDeviceAxisY = SensorManager.AXIS_MINUS_Z;
            break;
        case Surface.ROTATION_270:
            worldAxisForDeviceAxisX = SensorManager.AXIS_MINUS_Z;
            worldAxisForDeviceAxisY = SensorManager.AXIS_X;
            break;
    }

    float[] adjustedRotationMatrix = new float[9];
    SensorManager.remapCoordinateSystem(rotationMatrix, worldAxisForDeviceAxisX,
            worldAxisForDeviceAxisY, adjustedRotationMatrix);

    // Transformamos la rotacion en generar un cabeceo, guiñada y rotacion(Basandonos de
que estamos
    // en un avion)
    float[] orientation = new float[3];
    SensorManager.getOrientation(adjustedRotationMatrix, orientation);

    // Convertimos los radianes a grados
    float pitch = orientation[1] * -57;
    float roll = orientation[2] * -57;

    mListener.onOrientationChanged(pitch, roll);
}
```

c. *Fragment: Sensores*

i. *Layout*



En este caso presentamos otro layout bastante compuesto, comenzando por un Relative Layout en la que anidamos numerosos LinearLayouts para generar dicha interfaz. En ella se integran multiples elementos como la View que hemos creado de la Vista de Avion, la del movimiento con el Acelerometro, un ImageView que rota como una Brujula gracias al sensor de orientación o un grafico que muestra, actualizado en tiempo real, el Flujo Magnetico donde esta posicionado el dispositivo. Por ultimo, los TextView mostraran sus valores y además los del sensor de Proximidad y Temperatura.

A parte, se ha agregado otro RecyclerView el que indica todos los sensores que tiene el teléfono y un FAB que gestionara el propio Flash del móvil para poder usarlo cual Linterna.

ii. Codigo

Nada mas mostrar el inicio del Fragment, vemos una multitud de parámetros con los que jugaremos a lo largo de la vista.

Parametros Iniciales

```
public class SensoresFragment extends Fragment implements SensorEventListener,
Orientation.Listener {

    private RecyclerView recyclerView;
    private SensorManager sensorManager;
    private List<Sensor> listsensor;
    private LineChart mChart;
    private Thread thread;
    private boolean encendida;
    private TextView acelerometro, orientacion, temperatura, proximidad, campomag,
gravedad;
    private ArrayList<String> liststring;
    private SensorAdapter adaptador;
    private AnimacionBall mAnimacionBall = null;
    private LinearLayout accelerometer;
    private Orientation mOrientation;
    private boolean plotData = true;
    private ImageView compass;
    private View root;
    private FloatingActionButton fabLight;

    private IndicadorAltura mIndicadorAltura;
    private float currentDegree = 0f;
```

Justo al crear la vista con el método onCreateView, instanciamos todos los ítems y creamos el Listener del FAB el cual llamada a dos métodos

FAB Listener

```
fabLight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if (!encendida) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                flashLightOn();
            }
            encendida = true;
        } else {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                flashLightOff();
            }
            encendida = false;
        }
    }
});
```

Estos métodos son quienes gestionaran el Flash del dispositivo

Metodos para encender y apagar el Flash

```
private void flashLightOn() {
    CameraManager cameraManager = (CameraManager)
getActivity().getSystemService(Context.CAMERA_SERVICE);

    try {
        String cameraId = cameraManager.getCameraIdList()[0];
        cameraManager.setTorchMode(cameraId, true);
        encendida = true;
        fabLight.setImageResource(R.drawable.ic_highlight_black_24dp);
```

```
    } catch (CameraAccessException e) {
    }
}

/**
 * Metodo para apagar la linterna
 */
@RequiresApi(api = Build.VERSION_CODES.M)
private void flashLightOff() {
    CameraManager cameraManager = (CameraManager)
getActivity().getSystemService(Context.CAMERA_SERVICE);

    try {
        String cameraId = cameraManager.getCameraIdList()[0];
        cameraManager.setTorchMode(cameraId, false);
        encendida = false;
        fabLight.setImageResource(R.drawable.ic_highlight_off_black_24dp);
    } catch (CameraAccessException e) {
    }
}
```

Tambien, en el mismo onCreate, instanciamos el MPAndroidChart[17] (Grafico) que usamos para el flujo magnético.

Instancia del Grafico

```
public void instanciarChart() {
    mChart = (LineChart) root.findViewById(R.id.chart1);
    mChart.getDescription().setEnabled(true);
    mChart.getDescription().setText(getString(R.string.flujo_magnetico));

    // Habilitamos los gestos
    mChart.setTouchEnabled(true);

    // Habilitamos el reescalado y arrastrar
    mChart.setDragEnabled(true);
    mChart.setScaleEnabled(true);
    mChart.setDrawGridBackground(false);

    //Zoom de X e Y
    mChart.setPinchZoom(true);

    //Background del grafico
    mChart.setBackgroundColor(Color.WHITE);

    LineData data = new LineData();
    data.setValueTextColor(Color.WHITE);

    //Datos vacios
    mChart.setData(data);

    //Leyenda del graphico
    Legend l = mChart.getLegend();

    //Modificamos la leyenda
    l.setForm(Legend.LegendForm.LINE);
    l.setTextColor(Color.WHITE);

    XAxis xl = mChart.getXAxis();
    xl.setTextColor(Color.WHITE);
    xl.setDrawGridLines(true);
```

[17] MPAndroidChart: https://github.com/PhilJay/MPAndroidChart

```
    xl.setAvoidFirstLastClipping(true);
    xl.setEnabled(true);

    YAxis leftAxis = mChart.getAxisLeft();
    leftAxis.setTextColor(Color.WHITE);
    leftAxis.setDrawGridLines(false);
    leftAxis.setAxisMaximum(100f);
    leftAxis.setAxisMinimum(-100f);
    leftAxis.setDrawGridLines(true);

    YAxis rightAxis = mChart.getAxisRight();
    rightAxis.setEnabled(false);

    mChart.getAxisLeft().setDrawGridLines(false);
    mChart.getXAxis().setDrawGridLines(false);
    mChart.setDrawBorders(false);


    agregarDatos();
}
```

El manejo de toda la información que recibirá el propio Grafico la manejaremos a través de un Hilo que estará a la espera de nuevas entradas hasta que cerremos el Fragment. Simplemente hace, infinitamente hasta el fin de la vista, que nuestro listener de datos recoja datos cada 10 milisegundos

Hilo a la escucha de entradas

```
private void agregarDatos() {
    //Agregamos los datos
    if (thread != null) {
        thread.interrupt();
    }

    thread = new Thread(new Runnable() {

        @Override
        public void run() {
            while (true) {
                plotData = true;
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    });

    thread.start();
}
```

Y a traves del metodo addEntry que recibe el evento del sensor, agrega el valor actual del mismo

AddEntry

```
private void addEntry(SensorEvent event) {

    LineData data = mChart.getData();

    if (data != null) {
```

```
        ILineDataSet set = data.getDataSetByIndex(0);


        if (set == null) {
            set = createSet();
            data.addDataSet(set);
        }


        data.addEntry(new Entry(set.getEntryCount(), event.values[0] + 5), 0);
        data.notifyDataChanged();

        // Notificamos el cambio de datos
        mChart.notifyDataSetChanged();

        // Limite de visibilidad
        mChart.setVisibleXRangeMaximum(150);


        // Movemos siempre el grafico al ultimo dato recogido
        mChart.moveViewToX(data.getEntryCount());

    }
}
```

Por ultimo, explicamos como recopilamos datos de los sensores. Primero los instanciamos y registramos en nuestro SensorManager que será quien gestione los eventos de manera sincronizada para enviar datos a la UI

Registramos la lista

```
public void registerList() {
    //Registramos en nuestro manager los sensores
    //Orientacion
    listsensor = sensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
    if (!listsensor.isEmpty()) {
        Sensor orientationSensor = listsensor.get(0);
        sensorManager.registerListener(this, orientationSensor,
                SensorManager.SENSOR_DELAY_UI);
    }
    //Acelerometro
    listsensor = sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
    if (!listsensor.isEmpty()) {
        Sensor acelerometerSensor = listsensor.get(0);
        sensorManager.registerListener(this, acelerometerSensor,
                SensorManager.SENSOR_DELAY_UI);
    }
    //Proximidad
    listsensor = sensorManager.getSensorList(Sensor.TYPE_PROXIMITY);
    if (!listsensor.isEmpty()) {
        Sensor magneticSensor = listsensor.get(0);
        sensorManager.registerListener(this, magneticSensor,
                SensorManager.SENSOR_DELAY_UI);
    }
    //Temperatura ambiental
    listsensor = sensorManager.getSensorList(Sensor.TYPE_AMBIENT_TEMPERATURE);
    if (!listsensor.isEmpty()) {
        Sensor temperatureSensor = listsensor.get(0);
        sensorManager.registerListener(this, temperatureSensor,
                SensorManager.SENSOR_DELAY_FASTEST);
    } else {
        temperatura.setText(R.string.temp_no_disponible);
```

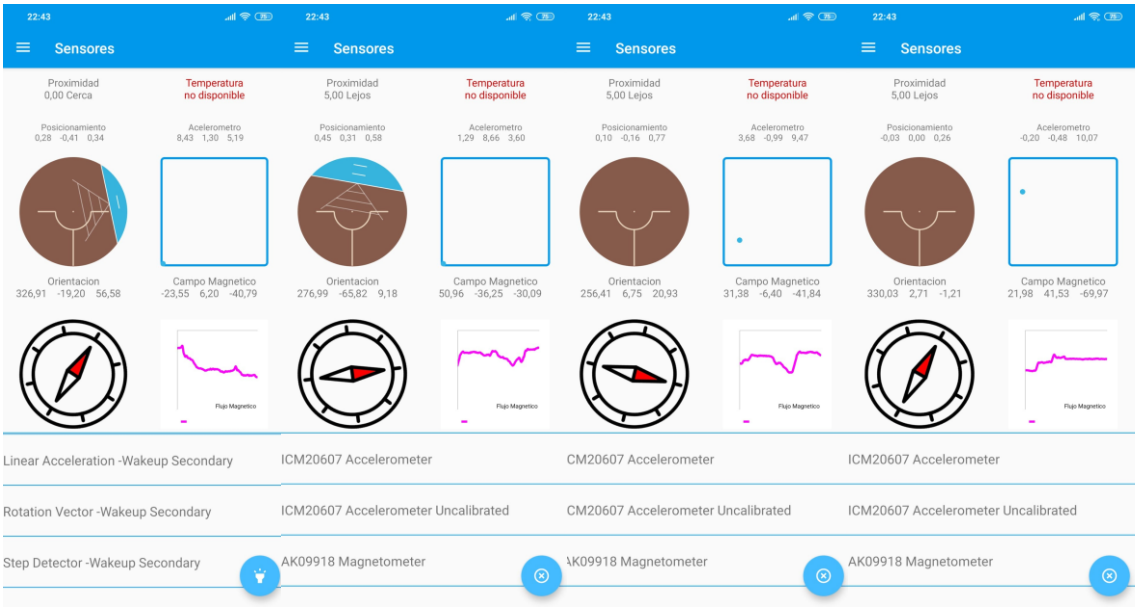
```
        temperatura.setTextColor(getResources().getColor(android.R.color.holo_red_dark));
    }
    //Rotacion Vectorial
    listsensor = sensorManager.getSensorList(Sensor.TYPE_ROTATION_VECTOR);
    if (!listsensor.isEmpty()) {
        Sensor temperatureSensor = listsensor.get(0);
        sensorManager.registerListener(this, temperatureSensor,
                SensorManager.SENSOR_DELAY_UI);
    } else {
        gravedad.setText(R.string.no_disponible);
        gravedad.setTextColor(getResources().getColor(android.R.color.holo_red_dark));
    }

    listsensor = sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);

    if (!listsensor.isEmpty()) {
        Sensor temperatureSensor = listsensor.get(0);
        sensorManager.registerListener(this, temperatureSensor,
                SensorManager.SENSOR_DELAY_UI);
    } else {
        campomag.setText(R.string.no_disponible);
        campomag.setTextColor(getResources().getColor(android.R.color.holo_red_dark));
    }
    listsensor = sensorManager.getSensorList(Sensor.TYPE_LIGHT);

    if (!listsensor.isEmpty()) {
        Sensor temperatureSensor = listsensor.get(0);
        sensorManager.registerListener(this, temperatureSensor,
                SensorManager.SENSOR_DELAY_UI);
    }
}
```

Y recogemos todos esos datos con el onSensorChanged que escribira o llamara a los métodos necesarios para mostrar la información en la UI

```
public void onSensorChanged(SensorEvent event) {
    synchronized (this) {
        StringBuilder builder;
        switch (event.sensor.getType()) {
            case Sensor.TYPE_ORIENTATION:
                builder = new StringBuilder();
                builder.append("Orientacion\n");
                for (int i = 0; i < 3; i++) {
                    float degree = Math.round(event.values[0]);
                    builder.append(String.format("%s    ", String.format("%.2f",
event.values[i])));
                    //Creamos una animacion de rotacion
                    RotateAnimation ra = new RotateAnimation(
                            currentDegree,
                            -degree,
                            Animation.RELATIVE_TO_SELF, 0.5f,
                            Animation.RELATIVE_TO_SELF,
                            0.5f);
                    //Duracion de la animacion
                    ra.setDuration(210);
                    //Seteamos la animacion despues del estado de reservado
                    ra.setFillAfter(true);
                    //Iniciamos la animacion
                    compass.startAnimation(ra);
                    currentDegree = -degree;
                }
                builder.append("\n");
                orientacion.setText(builder.toString());
                break;
            case Sensor.TYPE_ACCELEROMETER:
                //Acelerometro
                mAnimacionBall.onSensorEvent(event);
                builder = new StringBuilder();
                builder.append("Acelerometro\n");
```

```
                for (int i = 0; i < 3; i++) {
                    builder.append(String.format("%s    ", String.format("%.2f",
event.values[i])));
                }
                builder.append("\n");
                acelerometro.setText(builder.toString());
                break;
            case Sensor.TYPE_ROTATION_VECTOR:
                //Rotacion vectorial
                builder = new StringBuilder();
                builder.append("Posicionamiento\n");
                for (int i = 0; i < 3; i++) {
                    builder.append(String.format("%s    ", String.format("%.2f",
event.values[i])));
                }
                builder.append("\n");
                gravedad.setText(builder.toString());
                break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                //Campo Magnetico
                builder = new StringBuilder();
                builder.append("Campo Magnetico\n");
                for (int i = 0; i < 3; i++) {
                    builder.append(String.format("%s    ", String.format("%.2f",
event.values[i])));
                }
                builder.append("\n");
                campomag.setText(builder.toString());
                if (plotData) {
                    addEntry(event);
                    plotData = false;
                }
                break;
            case Sensor.TYPE_AMBIENT_TEMPERATURE:
                //Temperatura ambiental
                builder = new StringBuilder();
                builder.append("Temperatura\n");
                builder.append(String.format("%s    ", String.format("%.2f",
event.values[0])));
                builder.append("\n");
                temperatura.setText(builder.toString());
                break;
            case Sensor.TYPE_PROXIMITY:
                //Proximidad
                builder = new StringBuilder();
                builder.append("Proximidad\n");
                if (event.values[0] > 0) {
                    builder.append(String.format("%s    ", String.format("%.2f Lejos",
event.values[0])));
                } else {
                    builder.append(String.format("%s    ", String.format("%.2f Cerca",
event.values[0])));
                }
                builder.append("\n");
                proximidad.setText(builder.toString());
                break;
        }
    }
}
```

d. Capturas "In Action"



Concluye la explicación de la última Ampliacion de MyAPPs desarrollada y documentada en su totalidad por Emilio Rubiales Gutierrez.

¡Un placer programar dicha APP y espero que el documento sea de ayuda!

4. Anexo y Conclusion

Agradecimientos a Valdio Veliu por proveer del tutorial usado para tomar referencias en el Reproductor Multimedia
https://www.sitepoint.com/a-step-by-step-guide-to-building-an-android-audio-player-app/

¡Como conclusión, un placer diseñar estas APPs, sobre todo cuando amas la programación y le pones tanto sudor y esfuerzo al trabajo! Ansioso por empezar con la siguiente
Espero que se valore todas las horas que se le han echado sin parar durante mas de 2 semanas y muchas gracias