

.mp3

Miguel Pardo Navarro

Programación Multimedia y Dispositivos Móviles

2º DAMv 2024

- Proyecto Final -





Índice

<i>Descripción del proyecto</i>	2
<i>Propuesta técnica</i>	2
<i>Flujo de navegación</i>	2
<i>Código a destacar</i>	14
<i>Diseño</i>	20
<i>Requisitos del proyecto</i>	23
<i>Manual de usuario</i>	36
<i>Fuentes utilizadas</i>	39

Descripción del proyecto

.mp<3 es una aplicación hecha por y para amantes de la música. En un primer vistazo, puede que pensemos que no se trata más que de un reproductor de audio como cualquier otro, que no ofrece multitud de novedades ni servicios que podamos encontrar en grandes plataformas de renombre. ¿Y esto no es, acaso, una virtud?

En estos días en los que nos vemos constantemente inundados con grandes cantidades de información y estímulos, resulta una bocanada de aire fresco encontrarse con una aplicación que te permitirá concentrarte plenamente en uno de los mayores placeres de la vida: **escuchar música**.

Con **.mp<3** únicamente necesitarás tener en tu móvil los archivos .mp3 de tus canciones favoritas, y nosotros nos encargamos del resto. Nuestra aplicación listará todos los archivos y te los servirá en bandeja para que simplemente escojas la canción que deseas escuchar en ese momento concreto.

Sin límite ni restricción alguna, podrás disfrutar de tu música como y cuando quieras. No hace falta ni mencionar que aquí no encontrarás ningún tipo de anuncio ni publicidad invasiva que tanto abunda en otros servicios gratuitos. Porque, evidentemente, **.mp<3** es totalmente **gratis**. Se trata de un proyecto hecho, como su nombre bien indica, por amor al arte. Y, en concreto, se trata de nuestra particular declaración de amor a la música.

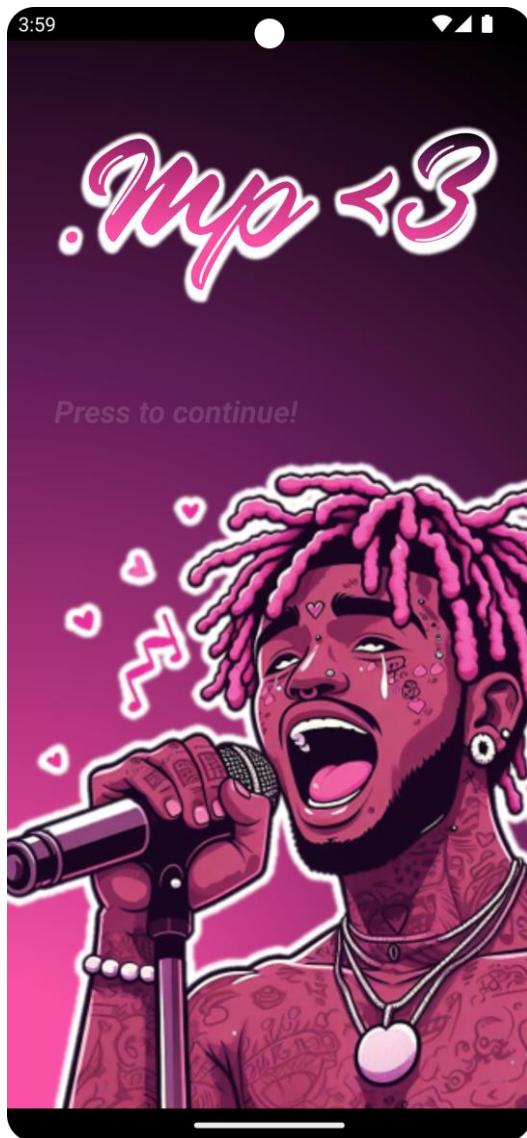
Propuesta técnica

Antes de nada, debemos mencionar que el proyecto ha sido realizado en la versión Hedgehog de Android Studio, con lenguaje Java y la API 33 de Android. Las pruebas del proyecto se han llevado a cabo con un emulador de Pixel 7 Pro API 33.

Flujo de navegación

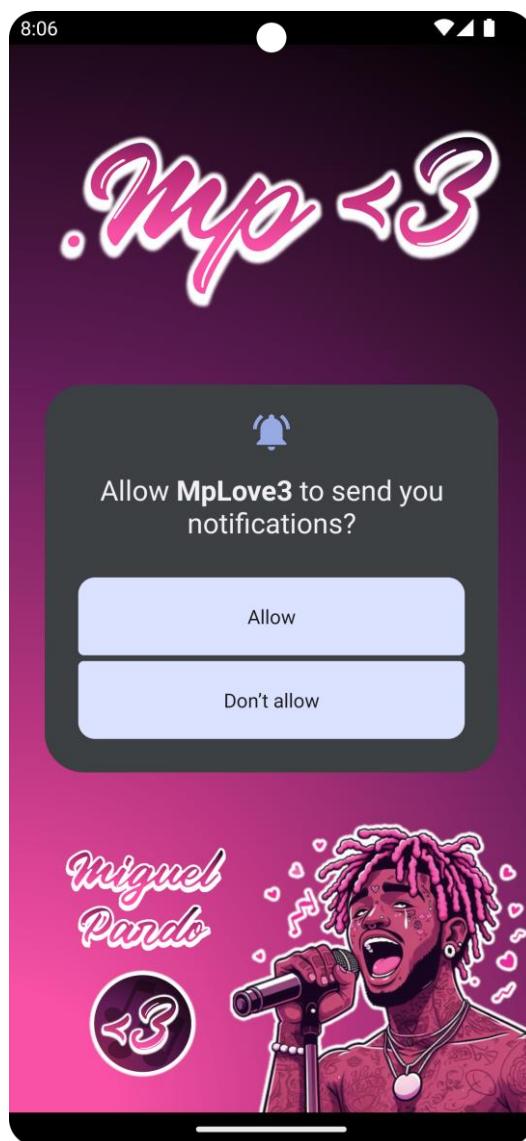
En este apartado se explicará con detalle todo el proceso de navegación entre las distintas pantallas de la aplicación y sus elementos.

Al abrir la aplicación por primera vez, se nos mostrará la activity ‘**Login**’. Aquí, lo primero que veremos será el logotipo del proyecto y la imagen del cantante en grande para decorar la primera pantalla de la app. Esta imagen, que en realidad se trata de un ImageButton, se deslizará automáticamente hacia arriba y se nos mostrará un mensaje animado que nos indica que debemos pulsar para continuar. De esta manera conseguimos que la primera impresión que tengan los usuarios sea la de un aplicación viva e interactiva.



Primera imagen que vemos en Login.

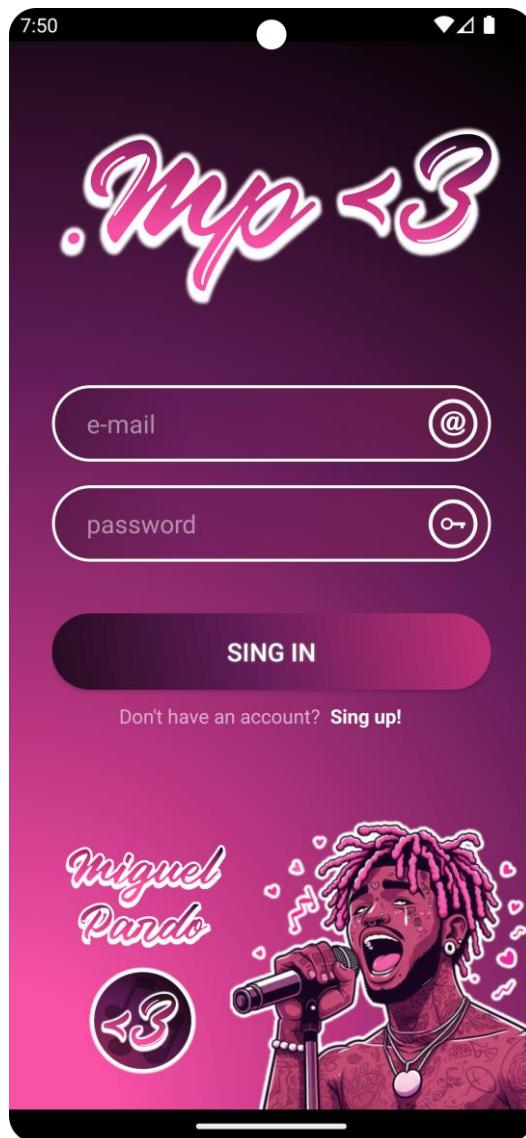
Una vez pulsemos la imagen, se mostrará el formulario de iniciar sesión, aparte de otros elementos decorativos y de información, pero no sin antes preguntar al usuario si desea otorgar a la app el permiso necesario para mandar notificaciones a su dispositivo. Si el usuario acepta se le enviará la notificación pertinente cuando se dé el caso, que explicaremos más adelante. Si no acepta, simplemente no recibirá notificaciones y se mostrará un mensaje Toast indicando esto mismo, pero podrá seguir utilizando la aplicación sin problema alguno.



Mensaje que nos pregunta si deseamos otorgar permisos para enviar notificaciones.

Volviendo al inicio de sesión, éste mostrará dos campos de texto especiales, uno para indicar el email y otro para la contraseña del usuario con el que queremos iniciar sesión. Este usuario es único y exclusivo para esta aplicación, siendo en realidad parte de la base de datos que utiliza la aplicación en Firebase Cloud Storage. Los usuarios serán documentos de la colección 'users', con los que guardaremos la información de los quienes se registran a la aplicación gracias a los campos de 'name', 'email' y 'password'.

La principal función del usuario será la de hacer que se recuerden tus credenciales y no obligarte a iniciar sesión cada vez que deseemos utilizar la aplicación. Una vez hayamos llenado los campos de texto correctamente, si pulsamos el botón 'SING IN', ya podremos entrar a la app como tal y disfrutar de sus funcionalidades.



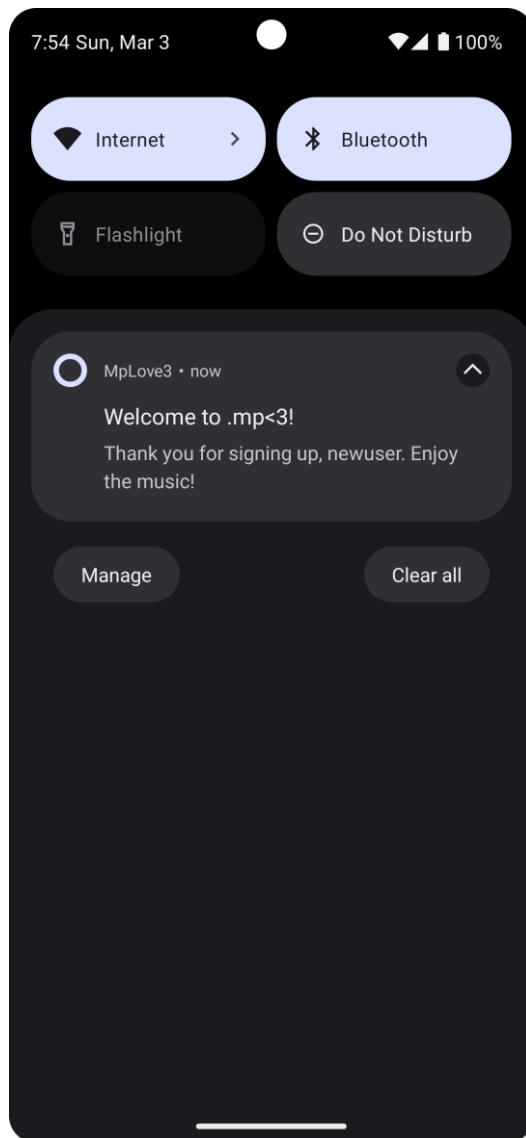
Login mostrando el formulario de inicio de sesión

Pero, si es la primera vez que utilizamos la aplicación y no tenemos ningún usuario, ¿cómo entramos? Bien, para eso está la indicación que nos dice que si no tenemos una cuenta con la que registrarnos podemos crear una con el botón de 'Sing up!'. Tras pulsarlo, se intercambiará el formulario de iniciar sesión por el de registrar un nuevo usuario. Aquí, podremos crearnos el usuario escogiendo un nombre, un email y una contraseña. Para poder proceder, será necesario marcar el checkbox de los términos y condiciones de uso, y, tras hacerlo, se coloreará el botón con la flechita hacia adelante que nos permitirá crear nuestro usuario.



Login mostrando el formulario de registro, sin marcar el checkbox de los términos y marcándolo.

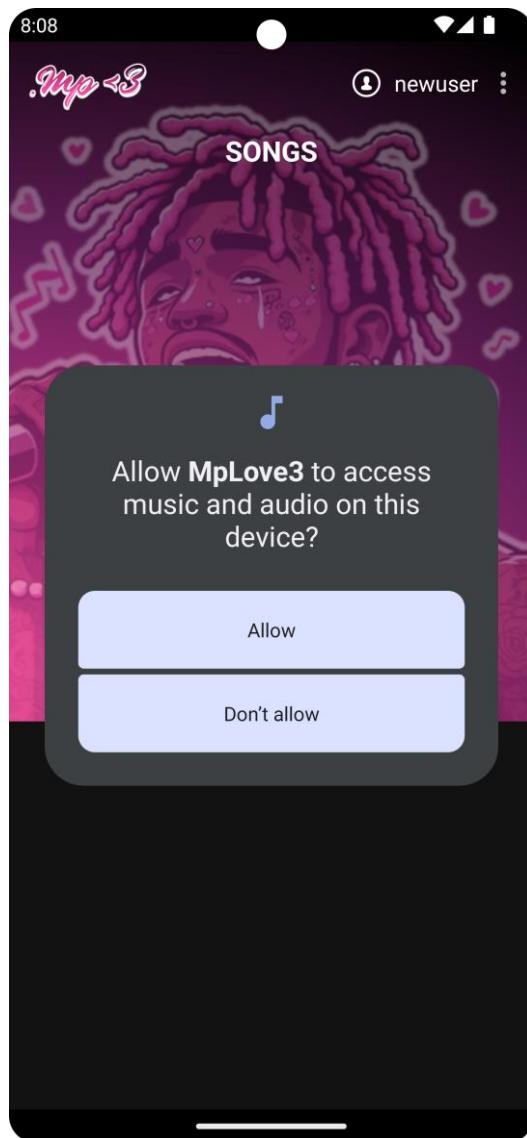
Bien, es en este momento cuando, si se indicó que deseamos otorgar los permisos de envío de notificaciones, se enviará una de estas. Tras el registro con éxito de un nuevo usuario, le llegará la notificación indicando nuestra gratitud por completar este movimiento y decidir disfrutar de la música de la mano de nuestra aplicación.



Notificación enviada cuando creamos un nuevo usuario (por algún motivo, en el emulador no se muestra la imagen del logo, en el móvil sí).

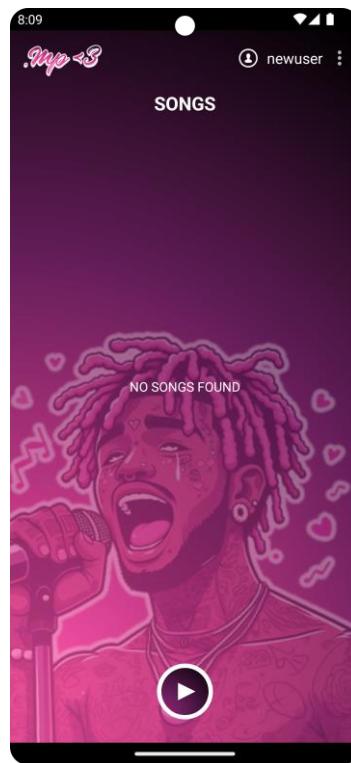
Con esto queda explicada la funcionalidad del primer activity ‘Login’. Tras iniciar sesión correctamente, ya sea con nuestro usuario o uno nuevo, se abrirá el segundo activity, ‘**MusicList**’. Esta actividad realmente funciona como la actividad principal del proyecto, ya que, si hemos iniciado sesión, la app guardará nuestros datos mediante el uso de preferencias y recordará que no debe mostrarnos de nuevo el login, sino este segundo activity. Esta actividad incluye también un Fragment, ‘MusicPlayerFragment’ que explicaremos más adelante.

La primera vez que accedamos a esta actividad, nos saltará un mensaje preguntándonos si queremos dar los permisos de acceso a los archivos de música y audio que haya en tu dispositivo. Estos permisos juegan un papel crucial para la aplicación, ya que son necesarios para poder listar todos los archivos .mp3 que haya en tu móvil y ofrecerlos para su escucha.

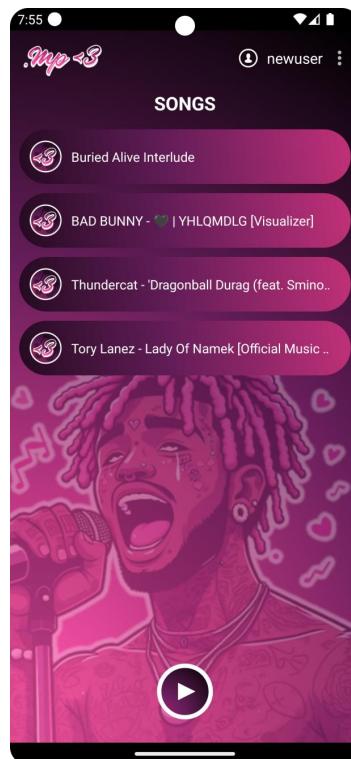


Mensaje que nos pregunta si deseamos otorgar permisos para acceder a los archivos de música y audio del dispositivo.

Si decidimos rechazar la solicitud, nos saldrá un mensaje que indique que no hay canciones disponibles. Si aceptamos, la aplicación procederá a listar las canciones que tengas en tu dispositivo en un RecyclerView, que usará una clase Adapter para gestionar su funcionalidad. Es importante mencionar, que será el usuario el que deba encontrar las canciones que desee escuchar, haciéndose cargo de cómo las consigue y las consecuencias que pueda tener. Los archivos deben estar en formato mp3. No es una práctica poco común la de descargar archivos de tus canciones favoritas, para así poder escucharlas de forma rápida y sin preocupaciones. Es aquí donde gana valor nuestra app, ya que facilita esta labor.



Si no otorgamos permisos o no tenemos ningún archivo .mp3 en nuestro dispositivo, se mostrará este mensaje en MusicList.

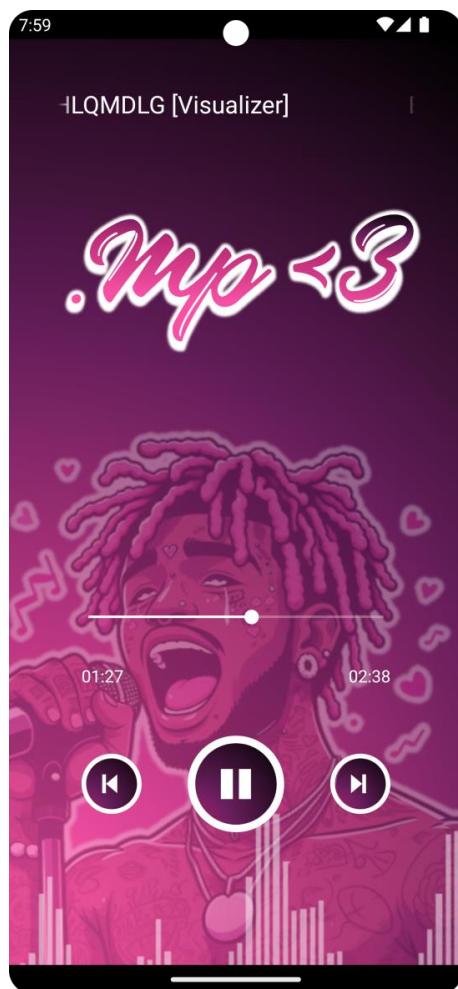


Si aceptamos y tenemos archivos .mp3, se listarán en MusicList.

Y es que, una vez tengamos la lista, solo tendremos que pulsar sobre una canción para reproducirla. Al hacerlo, se abrirá la una nueva actividad, ‘**MusicPlayer**’, que almacena toda la lógica del reproductor de música, automáticamente, iniciará a sonar la canción.

Aquí mostraremos el título de la canción que se está reproduciendo, aunque realmente se trata del nombre del archivo .mp3 que contenga la canción. También veremos el tiempo que dura la canción y el tiempo exacto en el que se encuentra su reproducción, llenando una Seekbar conforme avanza el tiempo.

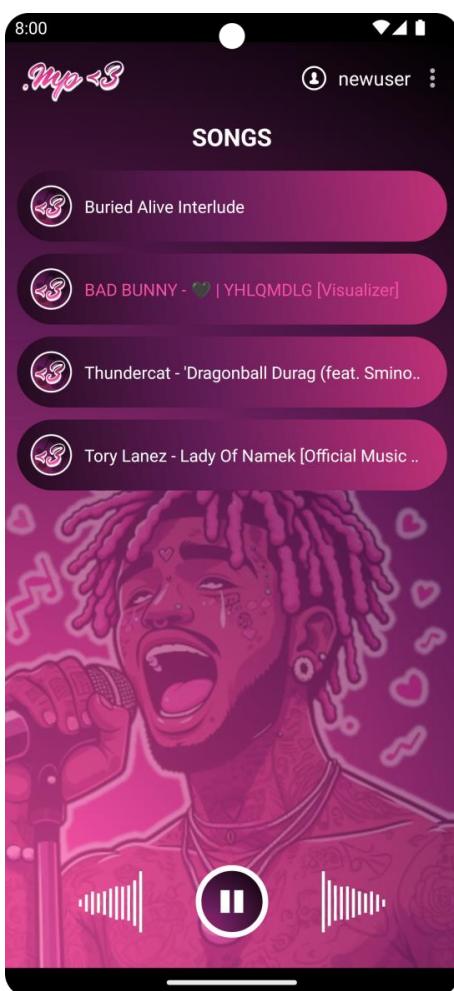
En la parte inferior se encontrarán los típicos botones (que en este caso no serán botones sino imágenes con sus `onClickListener()`) de *pause* o *play*, y los de reproducir la siguiente la canción de la lista o la anterior. Estos últimos no tienen más complicación, su propio nombre indica su función. Al llegar a la última canción, si pulsamos sobre icono de siguiente volverá a la primera canción. El ícono de *pause*, en cambio, cambiará entre el *pause* o *play* según se esté reproduciendo la canción o no. Su función obviamente, será la misma que el ícono que la representa. Además, si la canción se está reproduciendo se mostrará un GIF de un visualizar musical, que, a modo de decoración, nos sirve también como guía visual de que hay música sonando. Este GIF no se mostrará cuando la canción se haya pausado.



Reproductor de música en MusicPlayer.

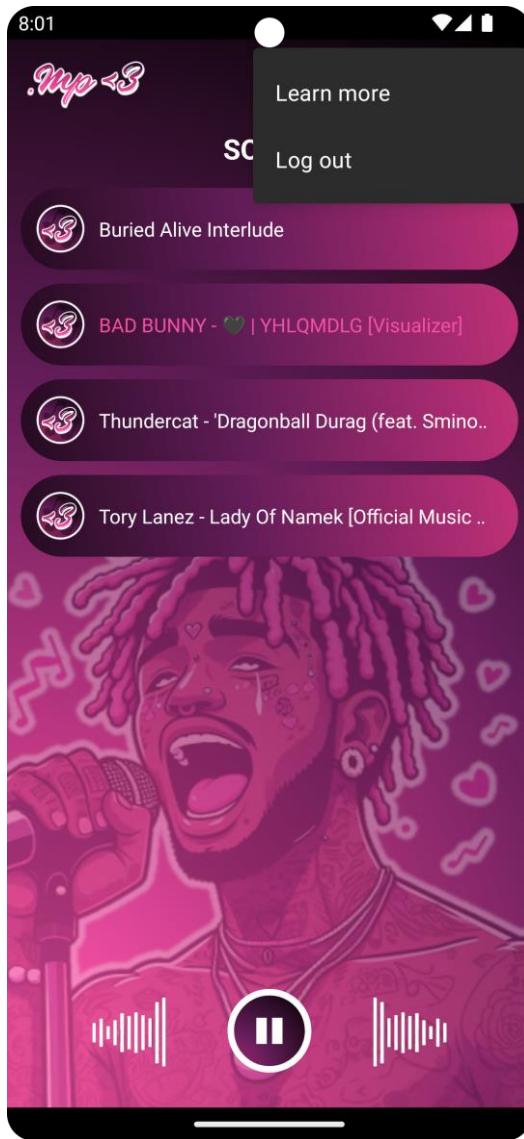
Si pulsamos sobre el botón de propio de Android de ir para atrás, regresaremos a la lista de canciones. Si se está reproduciendo una canción, el nombre de esta habrá cambiado su color a rosa para indicar que está sonando. Si pulsamos sobre cualquier otra canción, empezará a reproducirse esa canción y se abrirá de nuevo la actividad del reproductor. Sin embargo, si pulsamos la misma canción que ya está sonando, simplemente se abrirá el reproductor indicando el minutaje por el que se encuentra.

Además, veremos como el fragmento que mencionamos anteriormente, '**MusicPlayerFragment**', habrá cambiado su aspecto. Al principio, cuando todavía no está sonando ninguna canción, simplemente se mostrará el icono de *play*, sin ser este operativo. Una vez ya hemos pulsado una canción para iniciarla y vuelto a la lista veremos como el icono habrá cambiado al de *pause*, y ahora cada vez que lo pulsemos, podremos pausar la canción y reiniciar la canción a nuestro antojo. Esto es especialmente útil ya que de esta manera podremos pausar si queremos una canción sin necesidad de abrir la otra actividad. Igual que en el reproductor teníamos el GIF de las barras de sonido cuando suena una canción, el Fragment contará con dos GIFs a los lados del icono de *pause* o *play*, que tendrán la misma función.



MusicPlayerFragment en MusicList cuando hay una canción sonando.

Nos hemos centrado en la funcionalidad principal de la aplicación que es, evidentemente, reproducir música. Pero en la misma actividad ‘MusicList’, tendremos también una Toolbar en situada la parte superior de la pantalla. Aquí, se mostrará el logo del proyecto, el nombre de nuestro usuario con el que hemos iniciado sesión y un menú que accionamos con el típico ícono de los tres puntos. Este menú tendrá dos opciones: “Learn more” y “Log out”.



Menú en la parte superior de MusicList.

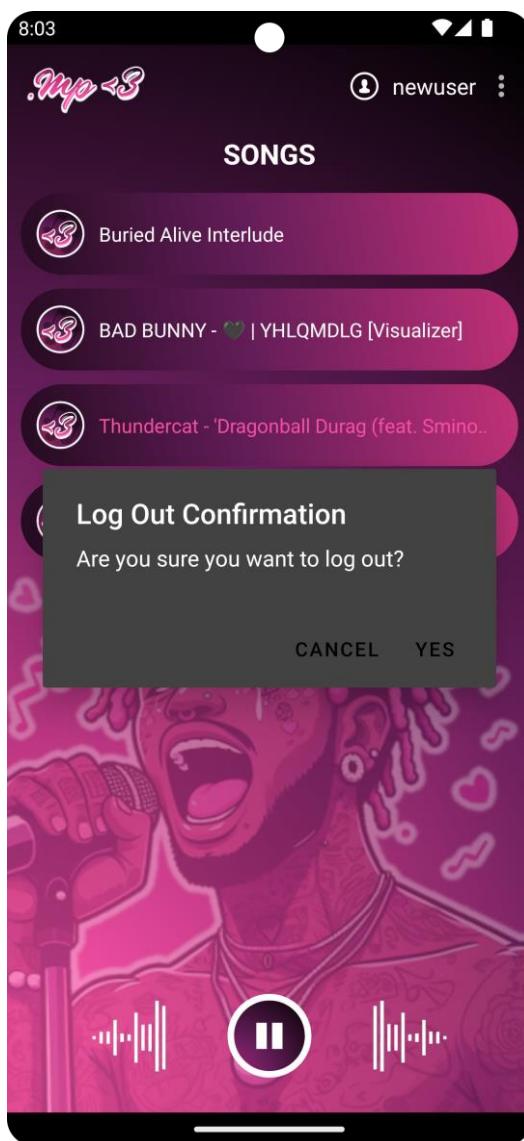
Si pulsamos sobre “Lean more”, nos llevará a un nuevo activity, ‘LearnMore’. En esta actividad simplemente se mostrará alguna información relevante sobre la aplicación, como la descripción de lo que es el proyecto, el nombre del creador y las maneras de contactarlo. Todo esto dentro de un Scrolling Fragment, ‘ScrollingFragmentInfo’, para así poder desplazar hacia abajo y continuar leyendo el texto pero manteniendo la información fuera del fragment en pantalla en todo momento. Además de este fragmento de la descripción de la aplicación, se mostrará un segundo Scrolling Fragment, ‘ScrollingFragmentTerms’, que funcionará

exactamente igual que el primero pero mostrando una lista con los términos y condiciones que tendrá la app. Ambos fragmentos se encuentran dentro de CardViews. Igual que con la actividad 'MusicPlayer', si le damos al botón de ir hacia atrás de Android volveremos a la actividad 'MusicList'.



ScrollingFragmentInfo y ScrollingFragmentTerms en LearnMore.

La segunda opción del menú, "Log out" nos permitirá cerrar sesión con el usuario actual y volver a la pantalla del login de la aplicación que explicamos en primer lugar. Al pulsar sobre esta opción nos mostrará un Dialog que nos preguntará si estamos seguros de que queremos cerrar sesión, mediante un botón de "cancel" que cerrará el diálogo sin realizar ninguna opción, y otra de "yes", que efectuará el cierre de sesión (y si estuviera reproduciéndose una canción, dejará de hacerlo). De esta manera, se cambiarán las preferencias que establecimos cuando se inició sesión y podremos iniciar de nuevo el proceso que aquí se ha explicado.



Dialog anterior al cierre de sesión.

Código a destacar

El proyecto cuenta con gran cantidad de código distribuido en métodos dentro de sus respectivas clases, pero sin duda la parte más importante es la que se encarga de manejar la lógica del **RecyclerView** que lista los archivos .mp3 que tengamos en nuestro dispositivo y la del **MediaPlayer** que nos permitirá escucharlos.

Estas funciones se han logrado programar de forma exitosa gracias al tutorial en YouTube para crear una aplicación para reproducir música en Android Studio del canal de Easy Tuto¹. En este vídeo se explica de manera detallada los pasos que debemos seguir para lograr esta aplicación.

Obviamente, el código del tutorial no ha sido copiado de forma literal, sino que se ha adaptado a las particularidades que considerábamos necesarias para el proyecto, así como se ha visto modificado y sobreescrito con el fin de añadir nuevas funcionalidades con las que el reproductor del tutorial no contaba, o eliminado las que no fueran pertinentes.

Para empezar, en nuestra actividad '**MusicList**', deberemos manejar la forma en la pedir los **permisos** al usuario para poder acceder a los archivos de música y audio:

```
private static final int PERMISSION_REQUEST_READ_MEDIA_AUDIO = 123;

private boolean checkPermission() {
    return ContextCompat.checkSelfPermission(this,
Manifest.permission.READ_MEDIA_AUDIO)
        == PackageManager.PERMISSION_GRANTED;
}

private void requestPermission() {
    ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.READ_MEDIA_AUDIO},
        PERMISSION_REQUEST_READ_MEDIA_AUDIO);
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults);

    if (requestCode == PERMISSION_REQUEST_READ_MEDIA_AUDIO) {
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            // Permisos concedidos, cargar la lista de música
            loadMusicList();
        } else {
            // Permisos denegados, mostrar el mensaje de "No Songs
Found"
            txtNoSongs.setVisibility(View.VISIBLE);
        }
    }
}
```

Y en el **onCreate()**:

```
if (!checkPermission()) {
    requestPermission();
} else {
    loadMusicList();
}
```

Si no se han concedido los permisos (ya sea porque es la primera vez en la aplicación y no se habían solicitado aún o porque se dijo que no anteriormente) se llamará a **requestPermission()** para pedirlos. Si los permisos se han otorgado, desde el **onCreate()** o desde **onRequestPermissionsResult()**, se llamará al método **loadMusicList()**:

```

private ArrayList<AudioModel> songList = new ArrayList<>();

private void loadMusicList() {
    songList.clear();

    String[] projection = {
        MediaStore.Audio.Media.TITLE,
        MediaStore.Audio.Media.DATA,
        MediaStore.Audio.Media.DURATION
    };

    String selection = MediaStore.Audio.Media.IS_MUSIC + "!= 0 AND " +
        MediaStore.Audio.Media.MIME_TYPE + "=?";

    String[] selectionArgs = new String[]{"audio/mpeg"};

    Cursor cursor = getContentResolver().query(
        MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
        projection,
        selection,
        selectionArgs,
        null
    );

    while (cursor.moveToNext()) {
        AudioModel songData = new AudioModel(
            cursor.getString(1),
            cursor.getString(0),
            cursor.getString(2)
        );

        if (new File(songData.getPath()).exists()) {
            songList.add(songData);
        }
    }

    if (songList.size() == 0) {
        txtNoSongs.setVisibility(View.VISIBLE);
    } else {
        txtNoSongs.setVisibility(View.GONE);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        recyclerView.setAdapter(new MusicListAdapter(songList,
getApplicationContext()));
        recyclerView.getAdapter().notifyDataSetChanged();
    }
}
}

```

Este método se encarga de llenar el ArrayList de AudioModel (clase que sirve como modelo de audio) songList, donde almacenaremos el título, datos y la duración de los archivos que haya en las rutas especificadas para audio en Android, comprobando que cumplen los requisitos. Si no hay canciones, se indica esto mostrando el TextView txtNoSongs; si sí las hay, establecemos el MusicListAdapter para recyclerView.

Primero mostremos el contenido de **AudioModel**, una clase con tres atributos, un constructor y sus getters y setters:

```
public class AudioModel implements Serializable {

    String path;
    String title;
    String duration;

    public AudioModel(String path, String title, String duration) {
        this.path = path;
        this.title = title;
        this.duration = duration;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDuration() {
        return duration;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }
}
```

Ahora, veamos **MusicListAdapter**, donde, entre otras cosas que posibilitan el buen funcionamiento del **RecyclerView**, tendremos un onClick sobre los **item** del listado:

```
@Override
public void onBindViewHolder(ViewHolder holder,
@SuppressLint("Recyclerview") int position) {
    AudioModel songData = songList.get(position);
    holder.txtMusicTitle.setText(songData.getTitle());

    if (MyMediaPlayer.currentIndex == position) {
        holder.txtMusicTitle.setTextColor(Color.parseColor("#F851A5"));
    } else {
        holder.txtMusicTitle.setTextColor(Color.parseColor("#FFFFFF"));
    }
}
```

```

        }

        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (MyMediaPlayer.currentIndex != position) {
                    MyMediaPlayer.getInstance().reset();
                    MyMediaPlayer.currentIndex = position;
                    Intent intent = new Intent(context,
MusicPlayer.class);
                    intent.putExtra("LIST", songList);
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    context.startActivity(intent);

                } else {
                    // Si la canción clicada es la misma que la
actualmente en reproducción,
                    // simplemente abre la actividad MusicPlayer sin
reiniciar la reproducción.
                    Intent intent = new Intent(context,
MusicPlayer.class);
                    intent.putExtra("LIST", songList);
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    intent.putExtra("RESUME_PLAYBACK", true);
                    context.startActivity(intent);
                }
            }
        });
    }
}

```

Aquí crearemos un intent a la actividad MusicPlayer, pasándole la lista de canciones y reiniciando una nueva clase auxiliar, **MyMediaPlayer**, solo si se pulsa en un item que no sea la canción que está sonando en ese momento, en cuyo caso no reiniciamos la instancia del **MediaPlayer**.

```

public class MyMediaPlayer {
    static MediaPlayer instance;

    public static MediaPlayer getInstance() {
        if (instance == null) {
            instance = new MediaPlayer();
        }
        return instance;
    }

    public static int currentIndex = -1;
}

```

En esta clase se crean las instancias MediaPlayer que se reproducirán en **MusicPlayer**:

```

void setResourcesWithMusic() {
    currentSong = songList.get(MyMediaPlayer.currentIndex);

    txtSongTitle.setText(currentSong.getTitle());
}

```

```
totalTime.setText(convertToMMSS(currentSong.getDuration()));

pause_play.setOnClickListener(v -> pausePlay());
next.setOnClickListener(v -> playNextSong());
previous.setOnClickListener(v -> playPreviousSong());

// Solo inicia la reproducción si no estás resumiendo la
reproducción
if (!getIntent().getBooleanExtra("RESUME_PLAYBACK", false)) {
    playMusic();
}
}

private void playMusic() {

    mediaPlayer.reset();
    try {
        mediaPlayer.setDataSource(currentSong.getPath());
        mediaPlayer.prepare();
        mediaPlayer.start();

        seekBar.setProgress(0);
        seekBar.setMax(mediaPlayer.getDuration());

        // Inicia la animación del GIF cuando comienza la reproducción
        startGifAnimation(visualizer_bars,
R.drawable.visualizer_bars);

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

En esta actividad se desarrolla toda la lógica que maneja la reproducción de música. Entre otros métodos y partes del código, **setResourcesWithMusic()** (llamado en el `onCreate`) y **playMusic()** son los más necesarios de mencionar. En el primero de ellos se establecen todos los datos necesarios sobre los elementos del layout, para mostrar la información de la canción que se va a reproducir y se llama a `playMusic()`. Este otro se encarga de iniciar la reproducción y que se escuche finalmente la música.

Esto es (de manera simplificada pues hay más código detrás) el proceso para lograr hacer funcionar el reproductor de música con los archivos de tu propio dispositivo.

A parte de esto, la aplicación contiene otros trozos de código que son dignos de una mención especial como toda la lógica del **manejo de usuarios**, o de las **animaciones** para mostrar o no elementos del layout.

Diseño

Uno de los aspectos en los que más nos hemos centrado es en el **diseño**, porque una buena aplicación debe verse bien y tener un diseño con sentido.

Empecemos por lo más básico, que no menos importante: el **nombre**. El nombre escogido para la aplicación, como ya hemos mencionado, es **.mp<3**. Se trata de una obvia referencia a la extensión de archivo más conocida para almacenar canciones, .mp3, y a uno de los primeros emoticonos de la era *Internet*, el simple pero legendario '<3'. Esta fue en su momento una popular manera de mostrar expresiones de afecto y amor virtualmente, dado al parecido que guarda este símbolo matemático con un corazón. De esta manera logramos un ingenioso nombre que mezcla el ámbito técnico de la música con el cariño y amor que le tenemos a ésta.

Además, como extra, el nombre utiliza las letras 'mp', que son las iniciales del creador de la aplicación, Miguel Pardo.

A pesar de todo esto, el símbolo '<' es un carácter reservado en la mayoría de sistemas informáticos, y, por lo tanto, el nombre técnico de la aplicación no ha podido ser el deseado, ya que no es posible usar este símbolo. Nos hemos tenido que conformar con llamar a la aplicación **MpLove3**, que, si bien no es lo mismo, mantiene la misma idea y es un nombre de archivo admitido. Pero esto se trata únicamente del nombre técnico para aquellos casos en los que no sea posible escribir **.mp<3**.



Representación visual del emoticono '<3', que simboliza un corazón.

Con el nombre claro, podemos pasar al **logo**. En este caso, nuestra aplicación cuenta con dos logotipos distintos: el principal, que es una representación del nombre; y el circular, que nos facilita la inserción en lugares más pequeños sin tener que escribir el nombre completo y usando solo '<3'.

El logo principal es, en definitiva, el nombre de la aplicación en colores rosados y bordes blancos, escrito con la fuente "*Rolling Beat*"², que otorga al logo de un estilo amoroso muy acorde al diseño e idea del proyecto.



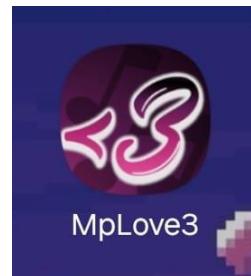
Logo principal del proyecto.

Sin embargo, este logo no es adaptable a la forma redondeada o circular que tienen los iconos de las aplicaciones en nuestros móviles. Por ello necesitábamos también de un segundo logotipo que se ajuste mejor a estas características. Este logo secundario funciona como **ícono** de la aplicación, la imagen que veremos antes de abrir la app junto al resto de las que tengamos instaladas. Se trata, por lo tanto de un diseño redondo



Logo secundario.

cuya principal atracción es el símbolo '<3', que hemos explicado anteriormente. El fondo mantiene los colores rosas y morados que cubren la aplicación y el borde blanco que también se puede apreciar en otros elementos del diseño. Como detalle, para completar el fondo se añadieron dos notas musicales con cierta transparencia, para que no capten demasiado la atención pero que quede constancia de que se trata de una aplicación de música. Ambos logos fueron diseñados desde la aplicación de Microsoft Office PowerPoint.



Icono de la aplicación en la pantalla de inicio de un dispositivo Android.

Si bien ya no es un logotipo como tal, otra imagen muy recurrente en la aplicación es el dibujo diseñado con la Inteligencia Artificial de creación de imágenes de Microsoft Bing³ que representa a un **cantante** mientras canta de forma apasionada. La imagen está inspirada en el rapero norteamericano Lil Uzi Vert⁴, artista que encaja muy bien estilísticamente con el diseño pensado para el proyecto. Mucha de su discografía versa sobre temas amorosos, la forma de presentar su arte es considerada como alternativa y propia del estilo de *Internet*, predominando el uso de colores rosas tanto en su obra como en su forma de vestir, en concreto en el pelo.



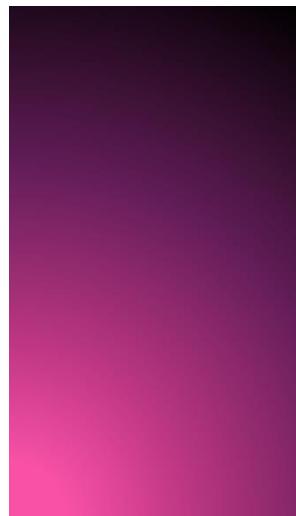
Lil Uzi Vert, artista que inspira el diseño de la aplicación.



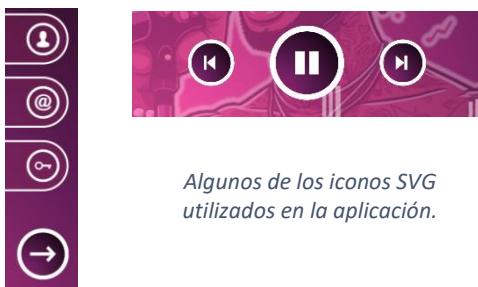
Imagen recurrente en la aplicación. Hecha con la IA de Bing.

Otro detalle muy importante es el **fondo** que hemos utilizado para los layouts de la app. Es un diseño simple pero limpio y elegante, formado por un degradado radial de colores desde **rosa** (#F44DA2) en la esquina inferior izquierda, hasta **negro** (#000000) en la superior derecha, pasando por otro tono de **rosa** (#BE3783) y **morado** (#631D58) en el medio. Desde un primer momento se consideró que el color principal del proyecto debía ser el rosa, siendo este el color característico del amor. Entorno a este color giran de forma análoga el resto de los empleados en el diseño, utilizando el negro como color auxiliar y el blanco para fuentes y bordes. El fondo no es más que una forma rectangular con dimensiones de pantalla de móvil hecha en PowerPoint.

Un detalle del diseño que hemos escogido es la tendencia por los bordes redondeados. Esto nos ayuda a dar a la aplicación un diseño mas moderno y atractivo. Se puede observar desde en los botones, hasta en los cuadros de texto, y por supuesto, en los **iconos**. Los iconos que nos ayudan a indicar las distintas funcionalidades que tienen los elementos de la aplicación siguen todos el mismo estilo, y es que todos son propiedad del mismo diseñador, Alessio Atzeni. Mediante la web SVGRepo⁵, Alessio ofrece un amplio catálogo de iconos de descarga gratuita. Los iconos han sido implementados como SVG.



Fondo utilizado para la aplicación.



Algunos de los iconos SVG utilizados en la aplicación.

En cuanto al diseño de los **botones**, **cuadros de texto** y **cardviews**, se ha logrado la forma deseada gracias a la ayuda que ofrece la página web AngryTools⁶ para la creación de botones para Android. Esta página nos asiste en la creación de un “shape” que usar como background para el elemento que queramos. Nosotros hemos escogido los colores propios del resto de la aplicación.



Botones, cuadros de texto, ítems del recycler y cardviews utilizando estilos similares mediante la implementación de archivos xml ‘shapes’ como background.



GIFs de visualizers musicales.

Otras imágenes utilizadas son los **GIFs** que simbolizan las ondas sónicas que muestran que hay música en reproducción. Estos GIFs se han insertado en la aplicación gracias a la librería Glide. La imagen de las ondas que aparecen bajo la lista de canciones fue encontrada en la web Gifer⁷, mientras que la imagen de las barras que aparece en el reproductor fue cogida de una librería de visualizers musicales para Andorid en GitHub⁸. Esta librería, que generaba las barras en función del sonido que estuviese sonando, se intentó implementar en el proyecto, pero se encontraba desactualizada y ya no está operativa. Sin embargo, los GIFs de muestra en la página oficial cumplen la función que necesitamos.

Requisitos del proyecto

1. Utilizar la última versión de Android Studio Hedgehog 2023.1.1 (imprescindible).

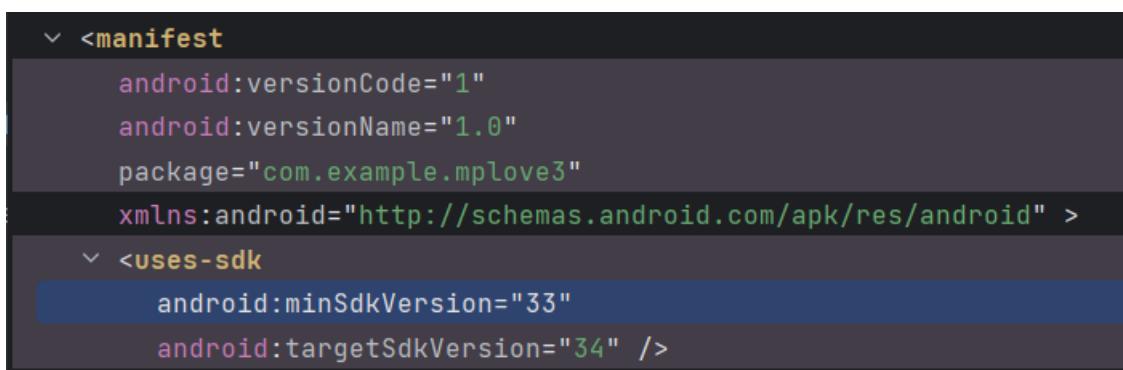
El proyecto fue creado y desarrollado en la versión de **Android Studio Hedgehog**.

2. Crear proyecto con lenguaje Java (imprescindible).

El proyecto fue creado y desarrollado con lenguaje **Java**.

3. Crear proyecto con SDK mínimo API 33 (“Tiramisu”; Android 13.0).

El proyecto fue creado con SDK mínimo **API 33**.

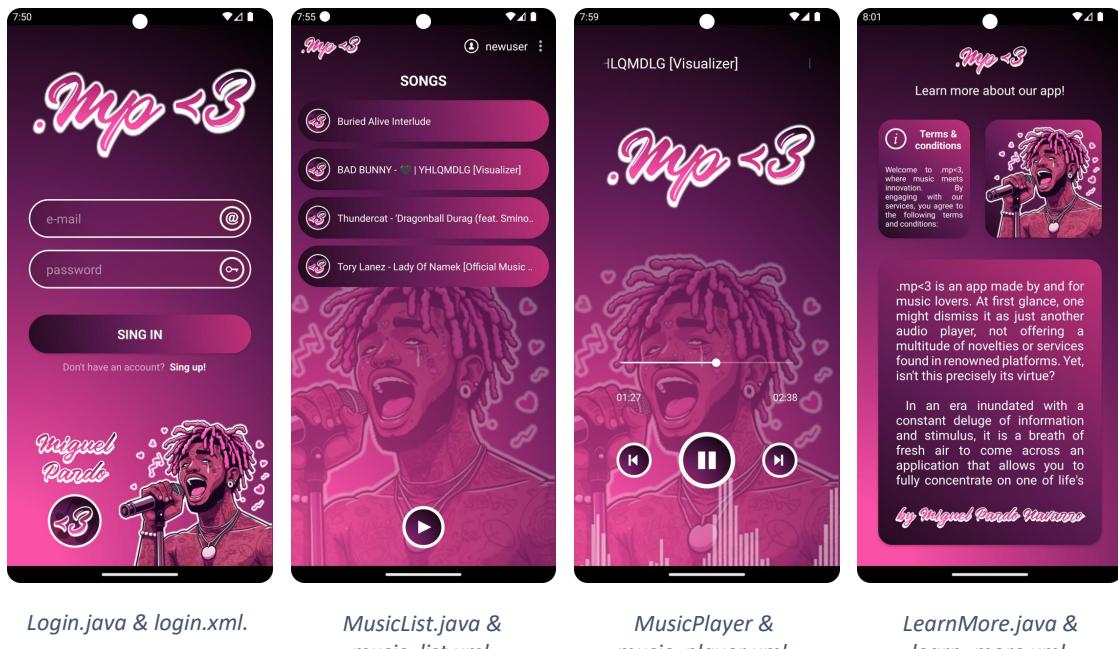


```
<manifest>
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.mplove3"
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <uses-sdk
        android:minSdkVersion="33"
        android:targetSdkVersion="34" />
```

Captura del Merged Manifest donde se observa la versión mínima de SDK.

4. Debe utilizar mínimo tres Activity diferentes con sus tres layouts.

Utiliza, concretamente, **cuatro Activities**, con sus correspondientes **layouts**.



5. Debe utilizar al menos un Intent o PendingIntent.

Utiliza **múltiples Intent** para abrir otras actividades.

En este ejemplo se observa el Intent que abre la actividad MusicList desde Login si se cumple la condición:

```
if(isLogin){
    // Si el usuario se logeó previamente, se abre la actividad
    MusicList directamente
    Intent intent = new Intent(Login.this, MusicList.class);
    startActivity(intent);
}
```

6. Debe utilizar al menos un Layout hecho con ConstraintLayout

Utiliza **múltiples ConstraintLayout**, tanto como layout padre como hijo.

Por ejemplo, login.xml usa ConstraintLayout como layout principal. En music_player.xml, tenemos un ConstraintLayout dentro del RelativeLayout principal, que nos permite lograr que la imagen tenga un width mayor al del layout principal para no mostrarla entera y que quede mejor en el layout.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    tools:context=".Login">

    <ImageView
        android:id="@+id/imgLogoMain"
        android:layout_width="375dp"
        android:layout_height="190dp"
        android:layout_marginTop="50dp"
        android:src="@drawable/logo_horizontal"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/tvwVacioParaAydarmeEnElConstraint"
        android:layout_width="0dp"
        android:layout_height="24dp"
        android:layout_marginBottom="688dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tvxPress2Continue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="172dp" />
```

Captura donde se observa el ConstraintLayout de login.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MusicPlayer"
    android:background="@drawable/background">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/singerFaded"
        android:layout_width="match_parent"
        android:layout_height="475dp"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true">

        <ImageView
            android:layout_width="470dp"
            android:layout_height="470dp"
            android:src="@drawable/singer_faded"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent" />

        <ImageView
            android:id="@+id/visualizer_bars"
            android:layout_width="450dp"
            android:layout_height="240dp"
            android:layout_marginTop="300dp"
            android:src="@drawable/visualizer_bars"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.512"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:tint="#80FFFFFF" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Captura donde se observa el ConstraintLayout de music_player.xml.

7. Debe contener un Login donde si un usuario ya ha sido logueado no tenga que volver a repetir el login mediante el uso de preferencias.

Contiene un **login** dentro de la actividad Login, que mediante métodos de FireBase, comprueba que el email y la contraseña coinciden con los existentes en la base de datos, permitiendo entonces el login y guardando mediante **preferencias** su email y su condición de logeado don un boolean.

```
btnSingIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String email = txtEmail.getText().toString();
        String password = txtPassword.getText().toString();

        // Realiza la consulta a la colección 'users'
        FirebaseFirestore db = FirebaseFirestore.getInstance();
        db.collection("users")
            .whereEqualTo("email", email)
            .whereEqualTo("password", password)
            .get()
            .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()) {
                        // Verifica si hay documentos que coincidan con las credenciales
                        if (!task.getResult().isEmpty()) {
                            // Credenciales válidas, inicia la actividad MusicList
                            Intent intent = new Intent(Login.this, MusicList.class);
                            startActivity(intent);

                            // Guarda las preferencias del inicio de sesión
                            SharedPreferences sharedPref =
                            getSharedPreferences("PreferencesLogin", Context.MODE_PRIVATE);
```

```
        Sharedpreferences.Editor editor = sharedPref.edit();
        editor.putString("Preferences_email", email);
        editor.putBoolean("Preferences_login", true);
        editor.apply();
    } else {
        // Credenciales inválidas, muestra un mensaje de error
        Toast.makeText(Login.this, "Invalid credentials.",
                      Toast.LENGTH_SHORT).show();
    }
} else {
    // Maneja el error en caso de que la consulta falle
    Toast.makeText(Login.this, "Error querying database.",
                  Toast.LENGTH_SHORT).show();
}
});
```

Comprueba si se ha iniciado sesión, y si es afirmativo, no muestra la actividad Login sino que muestra directamente MusicList.

```
SharedPreferences sharedPref =
getSharedPreferences("PreferencesLogin", Context.MODE_PRIVATE);
boolean isLogin = sharedPref.getBoolean("Preferences_login", false);

if(isLogin){
    // Si el usuario se logeó previamente, se abre la actividad
MusicList directamente
    Intent intent = new Intent(Login.this, MusicList.class);
    startActivity(intent);
}
```

8. Debe contener algún cuadro de diálogo personalizado que use dos botones.

Contiene un **cuadro de diálogo** al pulsar sobre la opción “Log out” del menú, que, antes de ejecutar el cierre de sesión, nos muestra el Dialog preguntando si estamos seguros de querer cerrar sesión. Las opciones serán “Cancel”, que cerrará el diálogo y no hará nada; y “Yes”, que llamará al método logOut() para cerrar sesión.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.log_out) {
        showLogoutConfirmationDialog();
        return true;
    } else if (item.getItemId() == R.id.learn_more) {
        Intent intent = new Intent(MusicList.this, LearnMore.class);
        startActivity(intent);
        return true;
    } else {

```

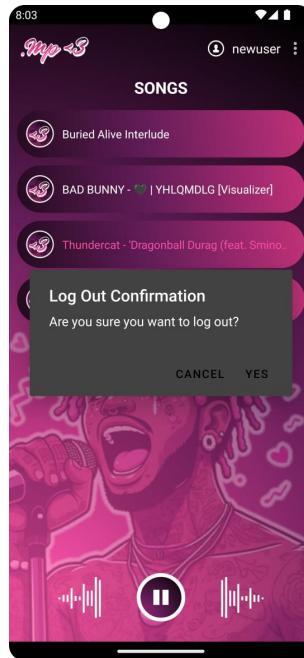
```

        return super.onOptionsItemSelected(item);
    }

}

private void showLogoutConfirmationDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage("Are you sure you want to log out?")
        .setTitle("Log Out Confirmation")
        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) {
                // Si el usuario hace clic en "Sí", realiza el logout
                logOut();
            }
        })
        .setNegativeButton("Cancel", new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int id) {
            // Si el usuario hace clic en "Cancelar", cierra el
            diálogo y no hace logout
            dialog.dismiss();
        }
    });
    AlertDialog dialog = builder.create();
    dialog.show();
}
}

```



Cuadro de diálogo en MusicList, donde se encuentra el menú.

9. Debe usar algún tipo de notificación según la funcionalidad de la app.

Envía una **notificación** (si se ha otorgado el permiso para hacerlo) cuando se crea un nuevo usuario en la aplicación, a modo de mensaje de bienvenida.

```

imgSingUp.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Verifica si el CheckBox está marcado
        if (chkTerms.isChecked()) {
            // Obtiene los valores de los EditText
            String name = txtNameSingUp.getText().toString();
            String email = txtEmailSingUp.getText().toString();
            String password = txtPasswordSingUp.getText().toString();

            // Verifica si todos los campos están llenos
            if (!name.isEmpty() && !email.isEmpty() && !password.isEmpty()) {
                // Crea un nuevo mapa con los datos del usuario
                Map<String, Object> newUser = new HashMap<>();
                newUser.put("name", name);
                newUser.put("email", email);
                newUser.put("password", password);

                // Añade el nuevo usuario a la colección "users"
                FirebaseFirestore db = FirebaseFirestore.getInstance();
                db.collection("users")
                    .add(newUser)
                    .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                        @Override
                        public void onSuccess(DocumentReference documentReference) {
                            // Éxito al añadir el usuario
                            Toast.makeText(Login.this, "User registered successfully!",
                                Toast.LENGTH_SHORT).show();
                            sendWelcomeNotification(name);

                            // Credenciales válidas, inicia la actividad MusicList
                            Intent intent = new Intent(Login.this, MusicList.class);
                            startActivity(intent);

                            //Guarda las preferencias del inicio de sesión
                            SharedPreferences sharedPref =
                                getSharedPreferences("PreferencesLogin", Context.MODE_PRIVATE);
                            SharedPreferences.Editor editor = sharedPref.edit();
                            editor.putString("Preferences_email", email);
                            editor.putBoolean("Preferences_login", true);
                            editor.apply();
                        }
                    })
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            // Error al añadir el usuario
                            Toast.makeText(Login.this, "Error registering user.",
                                Toast.LENGTH_SHORT).show();
                        }
                    });
            } else {
                // Muestra un Toast si alguno de los campos está vacío
                Toast.makeText(Login.this, "Please fill in all fields.",
                    Toast.LENGTH_SHORT).show();
            }
        } else {
            // Muestra un Toast si el CheckBox no está marcado
            Toast.makeText(Login.this, "Please accept the terms and conditions.",
                Toast.LENGTH_SHORT).show();
        }
    }
});

// Método para enviar la notificación de bienvenida
private void sendWelcomeNotification(String name) {
    String channelId = "welcome_channel";
    String channelName = "Welcome Channel";

    NotificationManager notificationManager = (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);

    if (notificationManager == null) {
        // Manejar el caso en que no se pueda obtener el NotificationManager
        return;
    }

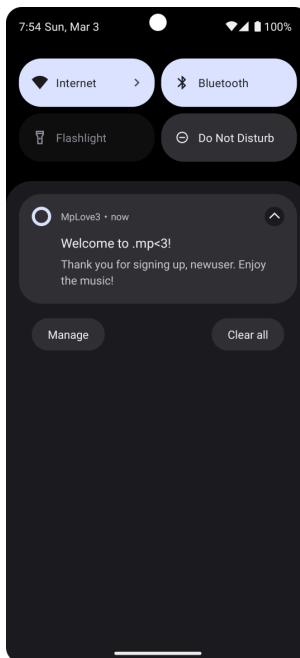
    NotificationChannel channel = new NotificationChannel(channelId, channelName,
        NotificationManager.IMPORTANCE_DEFAULT);
}

```

```
notificationManager.createNotificationChannel(channel);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this, channelId)
    .setSmallIcon(R.mipmap.ic_launcher_mp3_round)
    .setContentTitle("Welcome to .mp<3!")
    .setContentText("Thank you for signing up, " + name + ". Enjoy the music!")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);

Notification notification = builder.build();
notificationManager.notify(1, notification); // Use a unique ID for each notification
}
```



Notificación enviada cuando creamos un nuevo usuario (por algún motivo, en el emulador no se muestra la imagen del logo, en el móvil sí).

10. Debe incluir al menos dos de estos seis elementos propios de Material Design: Toolbar, Tab, Navigation Drawer, Floating Action Button, Snackbar o Card.

Incluye una **Toolbar** en MusicList que nos abre un menú del que se despliegan dos opciones: ir a la actividad LearnMore o cerrar sesión.

En LearnMore, la información mostrada en dos ScrollingFragments y una imagen se sitúan dentro de **CardViews**, para dar algo de cohesión al diseño.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MusicList"
    android:background="@drawable/background"
    >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="50dp"
        android:id="@+id/header"
        android:layout_marginTop="5dp">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    
```

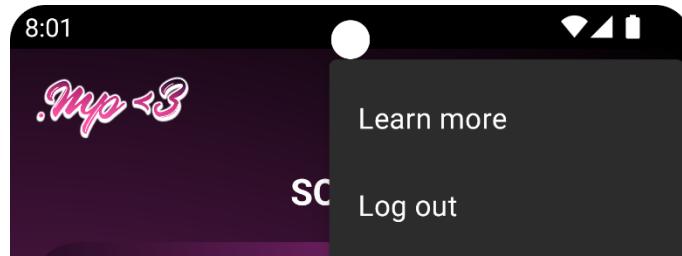
Captura donde se observa la Toolbar de music_list.xml.

```
<androidx.cardview.widget.CardView
    android:id="@+id/imgSinger"
    android:layout_width="165dp"
    android:layout_height="168dp"
    android:layout_marginTop="30dp"
    android:layout_marginEnd="30dp"
    app:cardCornerRadius="20dp"
    android:layout_alignParentEnd="true"
    android:layout_below="@+id/learn_more">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/cardshape"
        android:src="@drawable/singer" />

</androidx.cardview.widget.CardView>
```

Captura donde se observa una CardView en learn_more.xml.



Toolbar en MusicList, sin pulsar en el menú y tras pulsar en él desplegarlo.

CardViews en LearnMore.

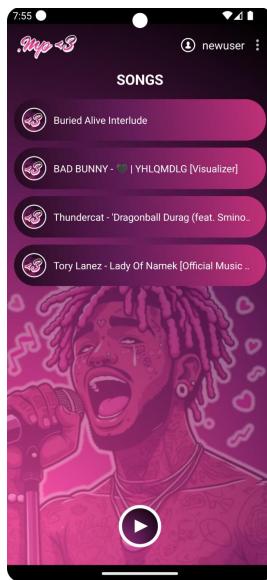
11. Debe contener alguna lista realizada mediante un RecyclerView.

Contiene una lista realizada mediante **RecyclerView**, que lista los archivos .mp3 del dispositivo del usuario permitiéndonos escoger qué canción queremos reproducir.

El código referente al RecyclerView y la función que cumple fue explicado con detalle con en el apartado [Código a destacar](#).

```
<androidx.recyclerview.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/recyclerView"
    android:layout_below="@+id/txtSongs"
    android:layout_above="@+id/fragmentMusicPlayer"
    />
```

RecyclerView en music_list.xml.



RecyclerView que muestra el listado de canciones en MusicList.

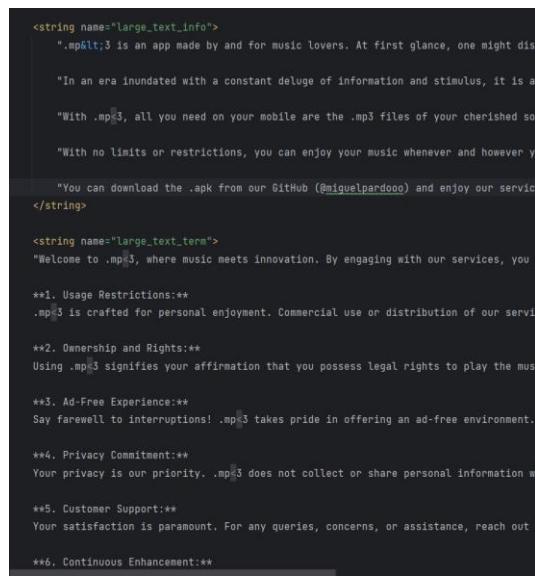
12. Debe usar al menos una Scrolling Activity.

Utiliza, en su lugar, **dos ScrollingFragments** en la actividad LearnMore. Los utilizamos para mostrar información a modo de descripción de la aplicación en el caso del **ScrollingFragmentInfo**, y para listar los términos y condiciones, en el caso de **ScrollingFragmentTerms**.

En estos fragmentos de contenido desplazable, podemos mostrar gran cantidad de texto de manera dinámica pues al hacer scroll podemos ver el texto que al principio estaba oculto.



Captura donde se observan los FragmentContainerView de los ScrollingFragments en learn_more.xml.



Captura donde se observan los strings de los ScrollingFragments en strings.xml.

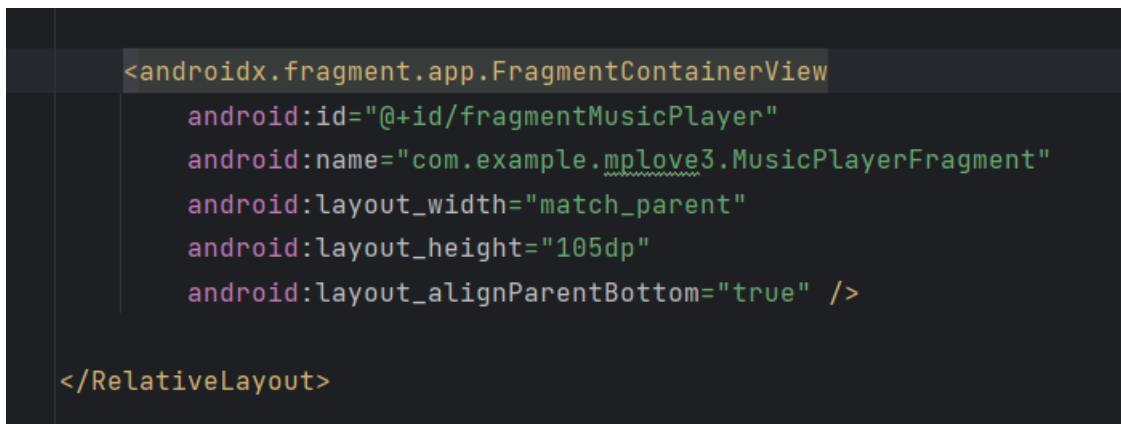


ScrollingFragmentInfo y ScrollingFragmentTerms en LearnMore.

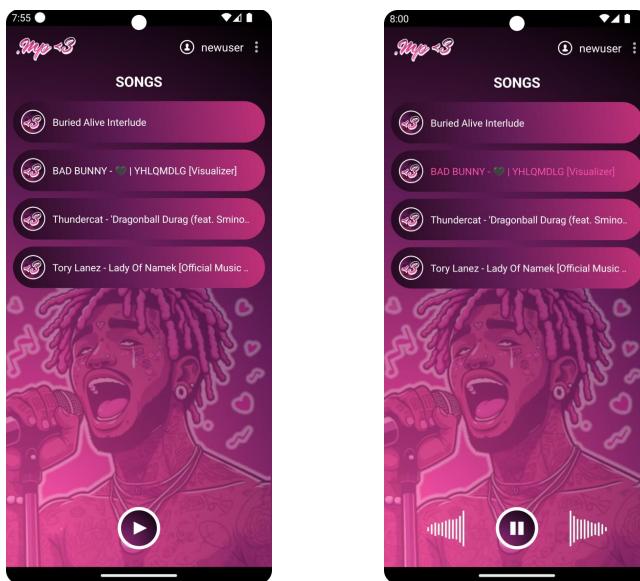
13. Debe incluir el uso de al menos dos fragments distintos.

Como hemos mencionado en el requisito anterior, se hace uso de **dos fragmentos** para incluir el requisito del scrolling.

A parte, se incluye un **tercer fragmento**, ‘MusicPlayerFragment’, que funciona como una versión muy reducida de la actividad MusicPlayer en la parte inferior de la actividad MusicList, donde podremos pausar o reanudar la reproducción de la música.



FragmentContainerView de MusicPlayerFragment en music_list.xml



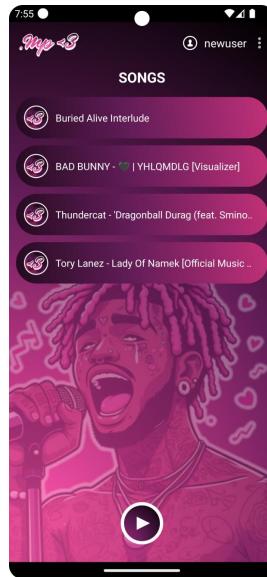
MusicPlayerFragment en MusicList, cuando no se reproduce una canción y cuando sí, respectivamente.

14. Debe hacer uso de una colección mediante el servicio de Cloud Firestore Database de Firebase.

La aplicación hace uso de **una colección de Cloud Firestore Database de Firebase**, llamada '**users**', que contará con los campos "email", "name" y "password". En ellos almacenaremos la información de los usuarios que iniciarán sesión en nuestra aplicación. Podremos iniciar sesión si los datos que introduzquemos en el formulario de Login son coreespondientes con los existentes en la colección, o bien podremos añadir un nuevo documento cuando creamos un nuevo usuario. En MusicList, aparecerá en campo "name", indicando el nombre del usuario conectado.

The screenshot shows the Firebase Cloud Firestore interface. On the left, there's a sidebar with various project management and developer tools like Firestore Database, Storage, Extensions, Release Monitor, Compilations, Analytics, and Participation. The main area is titled 'Cloud Firestore' and shows the 'Datos' tab selected. It displays a hierarchical view of the 'users' collection under '(default)'. A specific document named 'r0DPrfPH1SloS04N1a1p' is selected, showing its fields: email ('nw@mail.com'), name ('newuser'), and password ('1234'). There are also options to 'Añadir colección' (Add collection) and 'Añadir campo' (Add field).

Colección 'users' en Cloud Firestore de Firebase.



En MusicList, podemos observar el nombre del usuario conectado en la parte superior derecha de la pantalla.

15. Usar algo extra y propio que nos sirva para la aplicación: algún servicio adicional de Firebase, uso de API, uso de librería externa, etc.

La aplicación hace uso **una librería externa, Glide⁹**, que nos permitirá insertar GIFs y realizar acciones con ellos de manera fácil y eficiente. Es utilizada para los GIFs de los visualizers musicales que adornan un poco la app cuando una canción está sonando. Además, nos permite rotarlos y ajustar la transparencia, ajustando los GIFs perfectamente a las necesidades de la aplicación.

Métodos que inician y paran la animación en MusicPlayer:

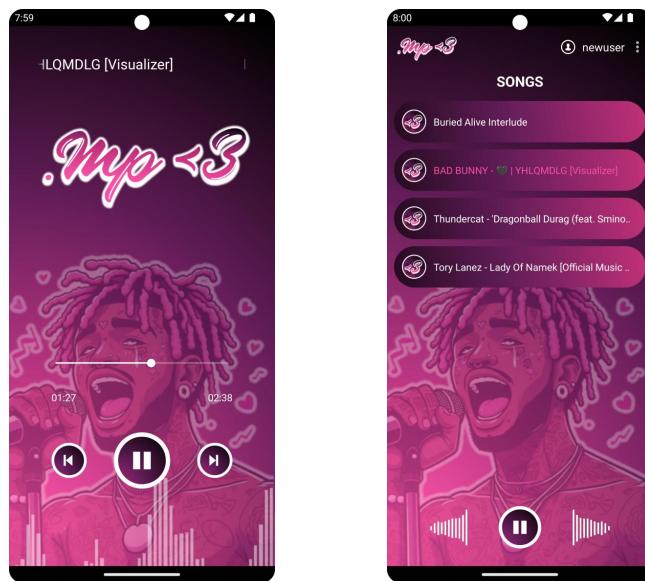
```
private void startGifAnimation(ImageView imageView, int gifResource) {
    // Asegúrate de que la Activity no esté finalizada
    if (isFinishing()) {
        return;
    }

    Glide.with(getApplicationContext()) // Usa el contexto de la
    aplicación o getActivity() según sea necesario
        .asGif()
        .load(gifResource)
        .into(imageView);

    // Aplica la transparencia
    imageView.setAlpha(0.5f);
}

private void stopGifAnimation(ImageView imageView, Activity activity)
{
    // Asegúrate de que la Activity no esté finalizada
    if (activity == null || activity.isFinishing()) {
        return;
    }

    // Limpia la animación Glide, pero deja la imagen visible
    Glide.with(activity).clear(imageView);
}
```



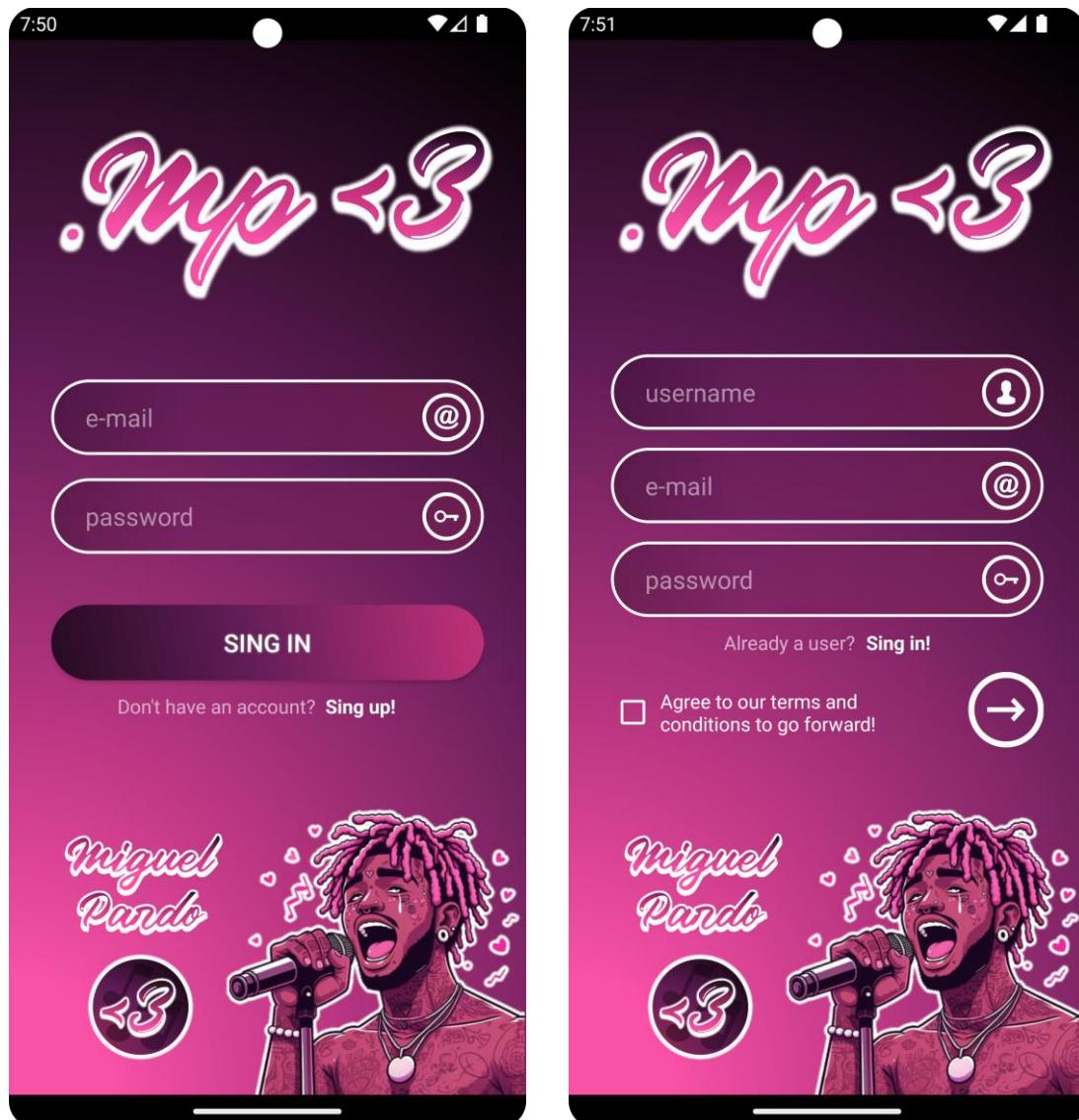
GIFs en funcionamiento en MusicPlayer y MusicPlayerFragment, respectivamente.

Manual de usuario

IMPORTANTE: El usuario deberá contar con archivos .mp3 en su dispositivo para que la aplicación los liste y permita su escucha. Por motivos legales, no enseñaremos cómo conseguirlos.

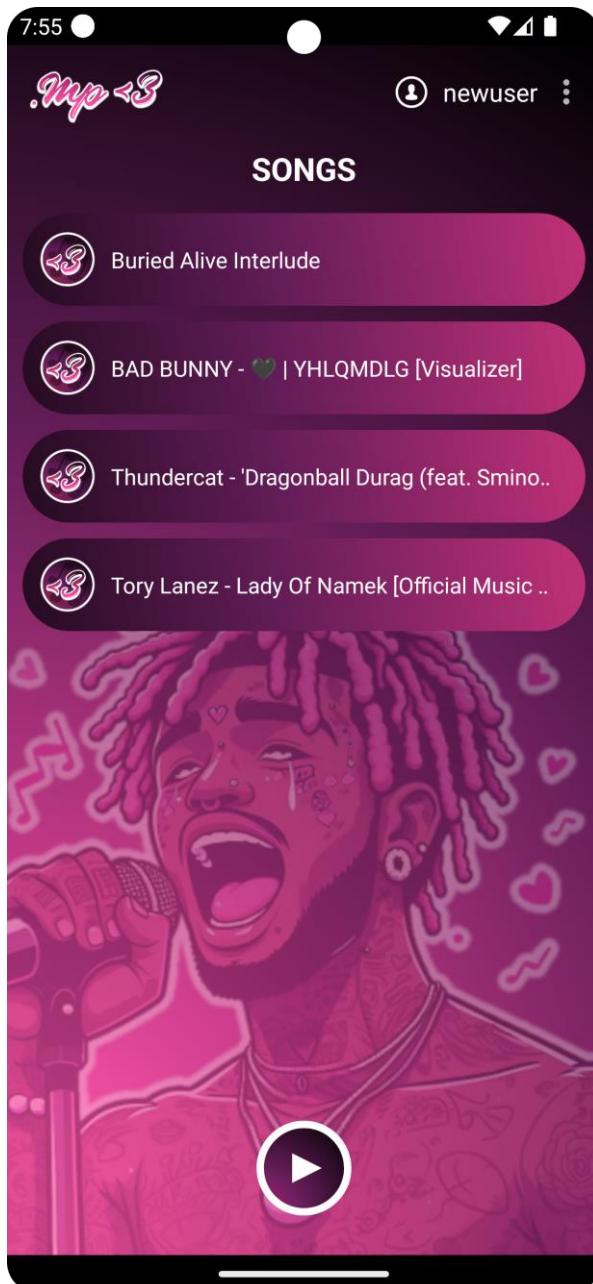
Paso 1. Iniciar sesión.

Lo primero que debemos hacer para utilizar nuestra aplicación será registrarnos con nuestro usuario (indicando correo y contraseña), o bien crear uno desde cero pulsando sobre "Sing up!".



Paso 2. Escoger una canción.

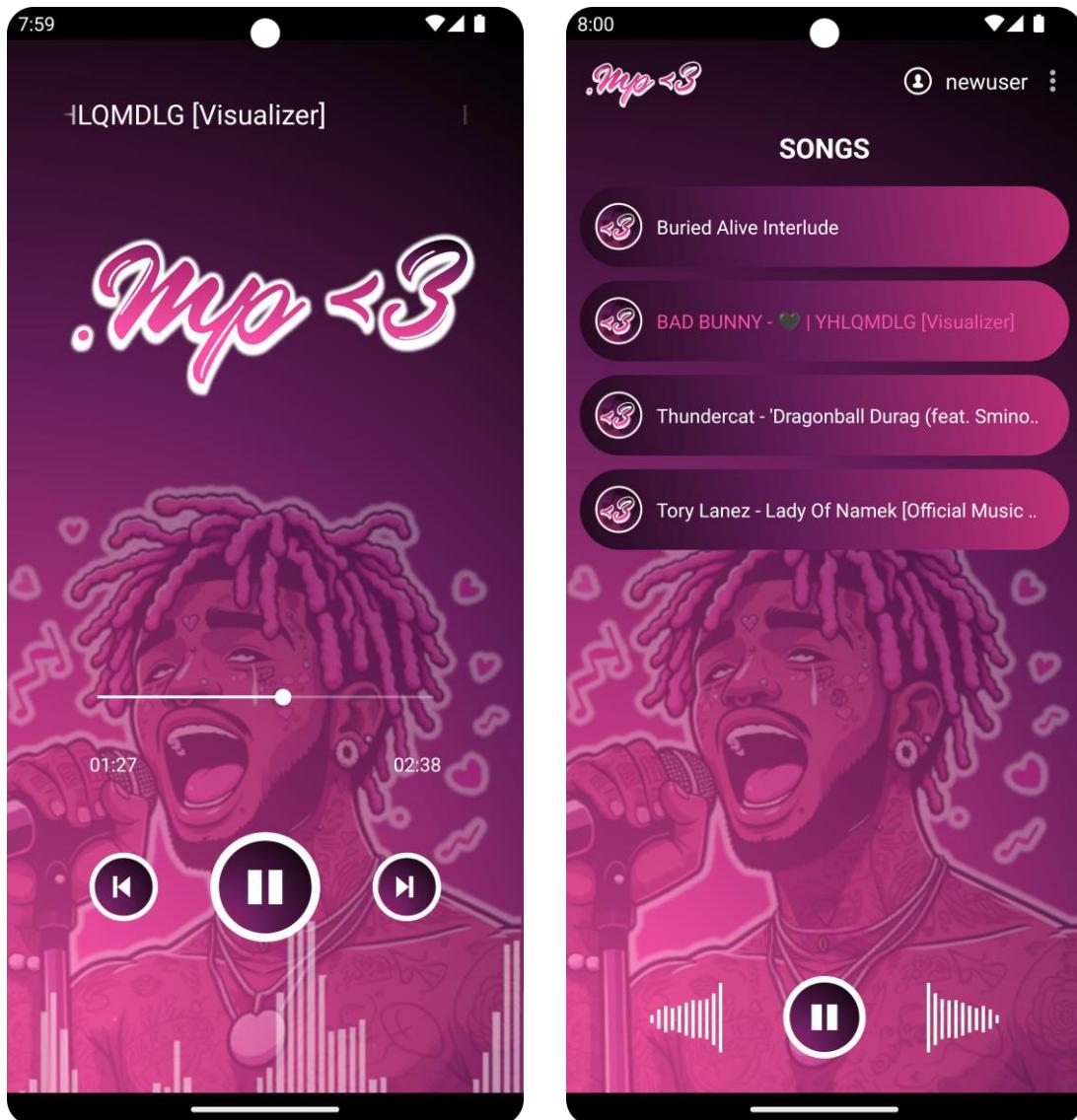
Si tenemos archivos .mp3 en nuestro móvil, nuestra aplicación se encargará de listar todos, y, ahora, será tu labor escoger la canción que deseas escuchar. Cuando lo hayas decidido, simplemente pulsa sobre ella.



Paso 3. Disfrutar de la música.

Una vez pulsemos sobre una canción, está comenzará a reproducirse automáticamente. Si deseásemos cambiar de canción, el botón de Android de ir hacia atrás nos devolverá a la lista de canciones, y, mientras la anterior sigue sonando, seremos libres de escoger otra en todo momento.

Ahora, lo único que tenemos que hacer es disfrutar de nuestras canciones favoritas con una aplicación que hará todo lo que pueda porque esto sea posible. <3



Fuentes utilizadas

¹ Easy Tuto. (2021, 31 octubre). Music Player Application | Android Studio Tutorial | 2024 [Vídeo]. YouTube. <https://www.youtube.com/watch?v=1D1Jo1sLBMo>

² Rolling Beat | Dafont.com. (s. f.). <https://www.dafont.com/es/rolling-beat.font>

³ Bing. (s. f.). Bing. <https://www.bing.com/images/create?FORM=GENILP>

⁴ Lil uzi vert. (s. f.). Spotify. <https://open.spotify.com/intl-es/artist/4O15NlyKLIASxsJOPrXPfz>

⁵ Repo, S. (s. f.). Metrize Circled Icons Collection - SVG Repo. SVG Repo. <https://www.svgrepo.com/collection/metrize-circled-icons/>

⁶ Android Button maker. (s. f.). <https://angrytools.com/android/button/>

⁷ design GIF on GIFER. (s. f.). GIFER. <https://i.gifer.com/Nt6v.gif>

⁸ Gauravk. (s. f.). GitHub - gauravk95/audio-visualizer-android:  [Android Library] A light-weight and easy-to-use Audio Visualizer for Android. GitHub. <https://github.com/gauravk95/audio-visualizer-android>

⁹ Bumptech. (s. f.). GitHub - bumptech/glide: An image loading and caching library for Android focused on smooth scrolling. GitHub. <https://github.com/bumptech/glide>