

# V A L L E   M P 3   P L A Y E R



A L B E R T O   R E Y   M O R E N O  
DESARROLLO DE APLICACIONES MULTIPLATAFORMA  
IES VALLE DEL JERTE - PLASENCIA  
CURSO 2018/19 -NOVIEMBRE 2018

## **ÍNDICE DE LA MEMORIA.**

<b>1. DESCRIPCIÓN</b>	<b>PÁG. 02</b>
<b>2. OBJETIVOS</b>	<b>PÁG. 02</b>
<b>3. JUSTIFICACIÓN</b>	<b>PÁG. 03</b>
<b>4. DESCRIPCIÓN TÉCNICA</b>	<b>PÁG. 04</b>
<b>a. MEDIOS</b>	<b>PÁG. 04</b>
<b>b. DESARROLLO DE LA APLICACIÓN</b>	<b>PÁG. 10</b>
<b>c. FUTURO DE LA APLICACIÓN</b>	<b>PÁG. 29</b>
<b>d. PROBLEMAS EN EL DESARROLLO</b>	<b>PÁG. 29</b>
<b>e. MANUAL DEL USUARIO</b>	<b>PÁG. 32</b>
<b>5. MEDIOS UTILIZADOS</b>	<b>PÁG. 34</b>
<b>6. BIBLIOGRAFÍA</b>	<b>PÁG. 35</b>

## **1. DESCRIPCIÓN.**

En unas líneas muy generales el proyecto consiste y ha consistido en desarrollar una aplicación móvil para la reproducción de los archivos MP3 (MPEG-1 o MPEG-2 Audio Layer III) del dispositivo. Dando igual la procedencia de los archivos internos o externos del terminal.

## **2. OBJETIVOS.**

Se pretende conseguir una aplicación móvil para los dispositivos móviles que contengan el sistema operativo Android a partir de la versión *Lollipop* (5.0) hasta la última versión disponible actualmente en el mercado que pueda ejecutar con rapidez, soltura y esplendor un reproductor de archivos MP3 del almacenamiento tanto interno como externo.

Tiene un objetivo claro de llegar al mayor número de usuarios jóvenes que buscan una alternativa al reproductor de música de *Google Play* – el cual actualmente te invade con numerosos anuncios de suscripción a su servicio de pago – u otros servicios que te dan menos opciones siendo de pago.

### **3. JUSTIFICACIÓN.**

La aplicación se ha desarrollado con la clara idea de crear un reproductor de canciones de ficheros MP3. Estos ficheros se escanearán desde los posibles almacenamientos del terminal, ya sea tanto de forma interna o externa.

Dicho reproductor debería ser desarrollado acorde a las necesidades actuales del mercado de aplicaciones móviles como también tener en cuenta los terminales del público objetivo.

Además, esta aplicación intenta suplir el vacío de aplicaciones de utilidad dejado por los incontables juegos del perfil del instituto en Google Play Store, abriendo un nuevo camino a proyectos venideros.

Es el primer proyecto que no es de carácter lúdico que aparecerá en el perfil del instituto abriendo un nuevo camino para aquellos que quieran crear aplicaciones educativas y no lúdicas en el marco de los Ciclos Formativos del centro.

#### 4. DESCRIPCIÓN TÉCNICA.

El proyecto a realizar es una aplicación multimedia en Android que realice las funciones de escaneado de archivos MP3 del almacenamiento interno como externo mediante los permisos indicados y con una interfaz visual adaptada a las últimas tendencias de uso de estas aplicaciones.

También hay que tener en cuenta que la aplicación a desarrollar debe cumplir con una serie de objetivos tanto técnicos como de rendimiento en terminales de baja gama (por ejemplo, Samsung Galaxy S2), media gama (BQ Aquaris M5) como en los de alta gama (Google Pixel 2).

##### a. MEDIOS.

Para realizar la ardua tarea del desarrollo de la aplicación multimedia de reproducción de archivos MP3 hemos decidido usar varias herramientas desde un gestor de proyectos hasta varias versiones de Android Studio que listamos a continuación:

#### HARDWARE:

- **Samsung Galaxy S2.** Teléfono inteligente anunciado por Samsung en febrero de 2011 en el *MWC (Mobile World Congress)*, y lanzado en mayo del mismo año. Cuenta con procesador de 1,2 GHz doble núcleo "Exynos" SoC (*System on a chip*). También cuenta con 1 GB de RAM LPDDR2

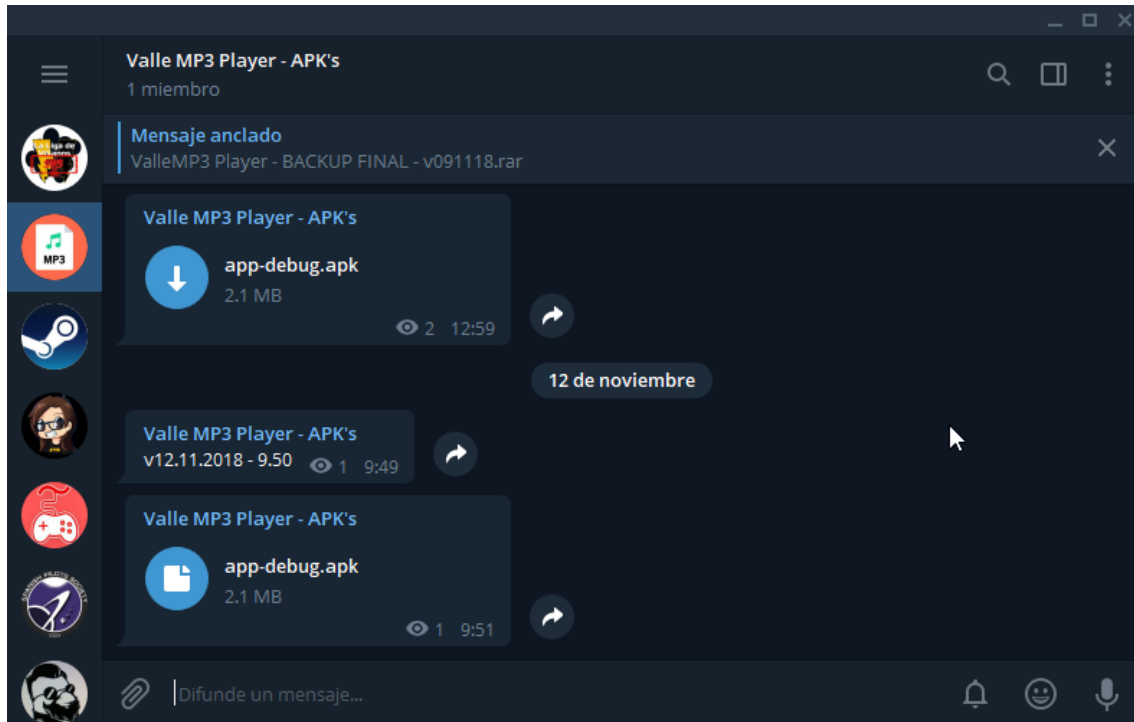
(*Mobile DDR2*), una pantalla de 4,3 pulgadas *WVGA Super AMOLED Plus* recubierta con *Gorilla Glass*.

- **BQ Aquaris M5.** Terminal de la compañía española bq anunciado en mayo de 2015 y lanzado en julio del mismo año, que cuenta con una pantalla de 5 pulgadas con la tecnología *Quantum Color+* y *Assertive Display*, además cuenta con el procesador *Qualcomm Snapdragon 615* (octa-core Core-A53 a 1,5 GHz y GPU Adreno 405), y con varias combinaciones en memoria RAM y almacenamiento interno (2 GB/16GB ó 3 GB/16GB ó 3 GB/32 GB) expandible mediante una tarjeta microSD.
- **Samsung Galaxy S8.** Terminal de gama alta fabricado por Samsung Electronics. El dispositivo móvil fue presentado en marzo de 2017 y se puso a la venta por primera vez en abril del mismo año. Tiene una pantalla *Super AMOLED* con resolución *Quad HD* (1440p 2960x1440), atrapan en él una variante del SoC *Exynos* y en la variante de Norteamérica se utiliza *Qualcomm Snapdragon 835*, ambos de 10nm. Y por último cuentan con dos combinaciones entre memoria RAM y almacenamiento interno (4 GB/64 GB ó 6 GB/128 GB) expandible mediante una tarjeta microSD.

## SOFTWARE:

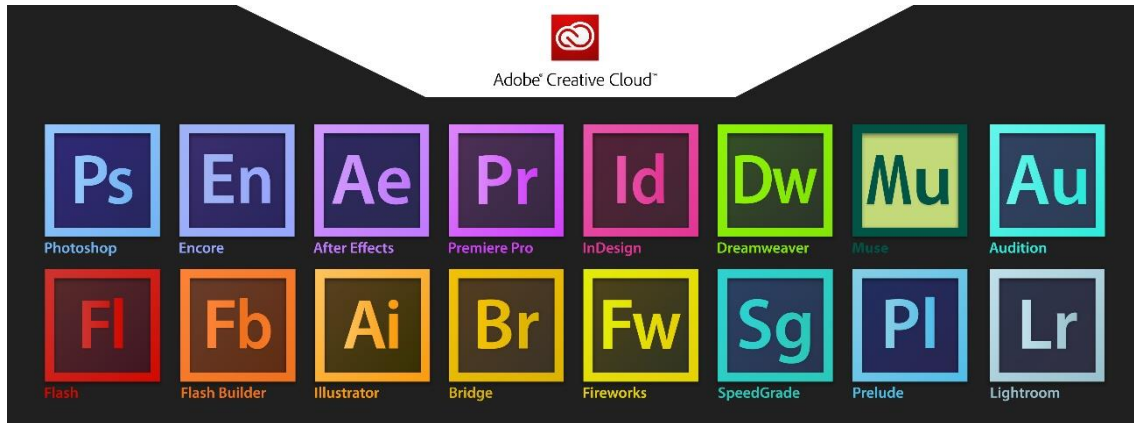


- **Trello.** Es un software de gestión de administración de proyectos con interfaz web, cliente para Android e iOS. Trello es un tablón virtual en el que se pueden colgar ideas, tareas, imágenes o enlaces. Es versátil y fácil de usar pudiendo usarse para cualquier tipo de tarea que requiera organizar información.



- **Telegram.** Aplicación de mensajería y VOIP desarrollada desde el año 2013 por los hermanos Nikolái y Pável Dúrov. Enfocado en la mensajería instantánea, envío de varios archivos y comunicación en masa. El servicio la administra una organización sin ánimo de lucro cuya sede principal opera en Dubái, Emiratos Árabes.





- **Suite Adobe CC (2018).** Es una suite de distintas aplicaciones en forma de talleres y estudios dotados de herramientas y funciones altamente profesionales creada y producida por *Adobe Systems* y que están dirigidas a la publicación impresa, publicación web, postproducción de video y dispositivos móviles.
  - **Adobe Photoshop CC (2018).** Es un editor de gráficos rasterizados y usado principalmente para el retoque de fotografías y gráficos. Es líder mundial del mercado de las aplicaciones de edición de imágenes y domina este sector de tal manera que su nombre es ampliamente empleado como sinónimo para la edición de imágenes en general.
  - **Adobe Illustrator CC (2018).** Es un editor de gráficos vectoriales en forma de taller de arte que trabaja sobre un tablero de dibujo, está destinado a la creación artística de dibujo y pintura para ilustración. Constituye su primer programa oficial de su tipo en ser lanzado por esta compañía definiendo en cierta manera el lenguaje gráfico contemporáneo mediante el dibujo vectorial.

- **Adobe Premiere Pro CC (2018).** Es un software de edición de vídeo orientado a la edición de vídeos profesionales. Lanzado en 2003, luego de una reescritura de código, Adobe Premiere Pro es el sucesor de Adobe Premiere, originalmente lanzado en 1991.



- **Android Studio (versión 2.3.3 y versión 3.2.0).** Es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014 y la última versión estable (3.2.1) fue lanzada el 11 de octubre de 2018.
- **Librerías.**
  - **Picasso.** Es una librería por la cual se nos permite cargar imágenes de manera rápida para todo tipo de terminales y de versiones de Android.

- **Material Dialogs.** Es una librería para la creación de aplicaciones con un diseño *Material Design* para todo tipo de terminales y de versiones de Android.

## b. DESARROLLO DE LA APLICACIÓN

Para empezar el desarrollo de la aplicación multimedia tenemos que abrir Android Studio 3.2.0 y crear un nuevo proyecto.

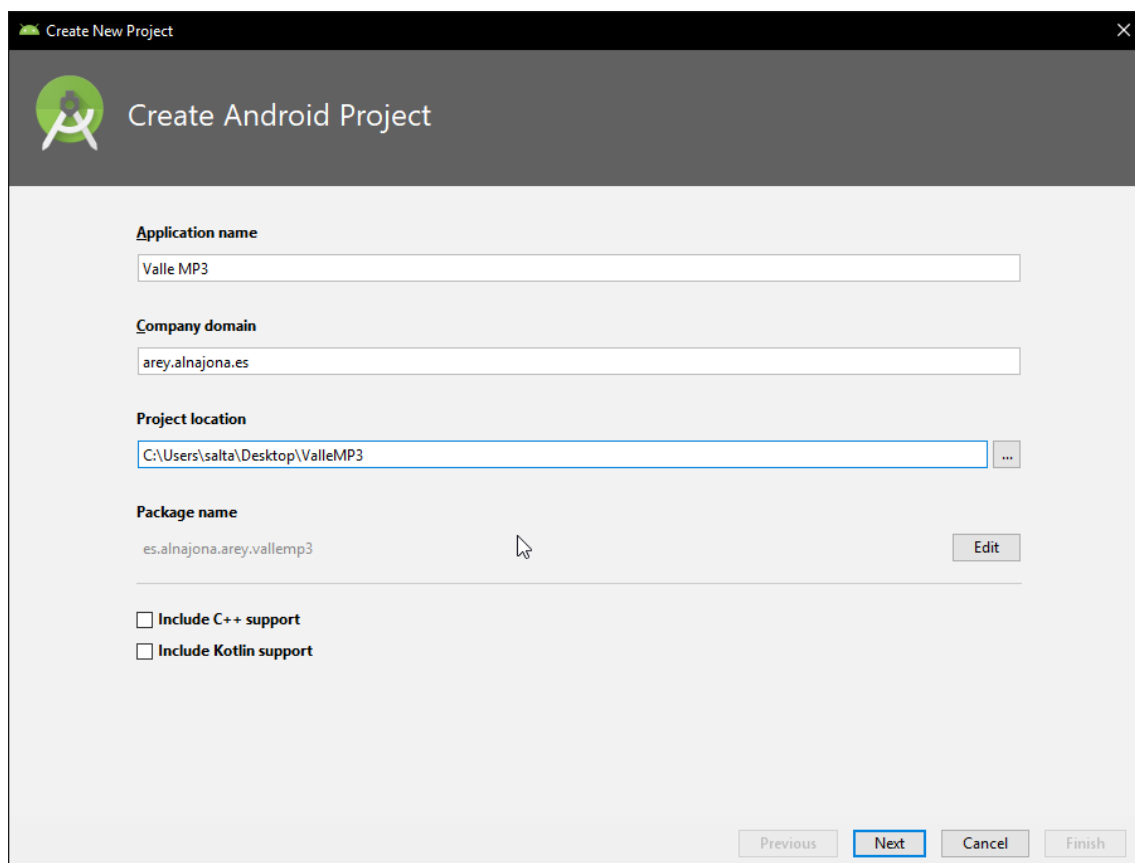


Ilustración 1. Creación del proyecto.

Creamos el proyecto para que se pueda ejecutar en *Phone and Tablet* desde la API 21 – Android 5.0 *Lollipop* – (Interfaz de Programación de Aplicaciones) así llegaremos hasta el 85% de terminales Android que existen.

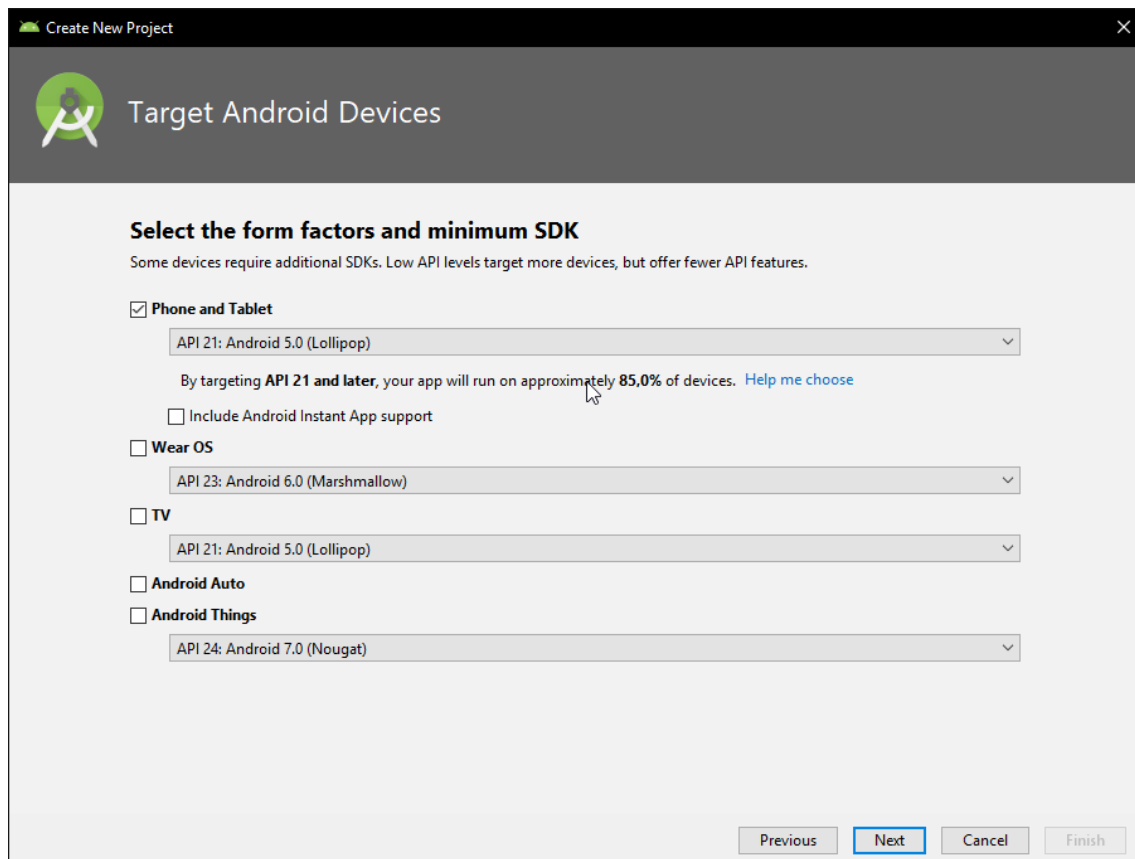


Ilustración 2. Elección del SDK mínimo.

Dejamos que el proyecto *ValleMP3* se afiance, se construya y se sincronice correctamente y proseguimos abriendo el *AndroidManifest.xml* para añadir los permisos que necesitaremos para acceder al almacenamiento tanto interno como externo y a Internet para poder buscar las imágenes de los artistas de los álbumes de música que encontremos.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Ilustración 3. Permisos del desarrollo.

A continuación, creamos la estructura de carpetas que sostendrán nuestro proyecto que son *service*, aquí añadiremos el servicio de reproducción; *task*,

## ALBERTO REY MORENO – VALLE MP3 PLAYER

controlaremos las tareas cuando los archivos se carguen, se proceda a su reproducción o a las propias animaciones del reproductor; *ui*, accederemos a las clases que hacen funcionar a las distintas interfaces y al propio reproductor correctamente; y *utils*, que contienen una serie de clases que controlan funciones básicas del reproductor.

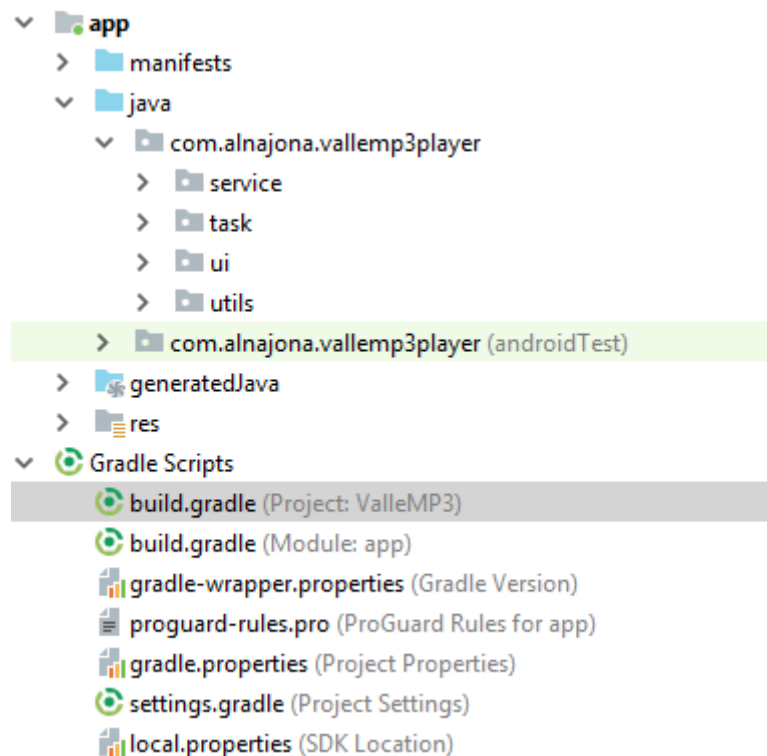
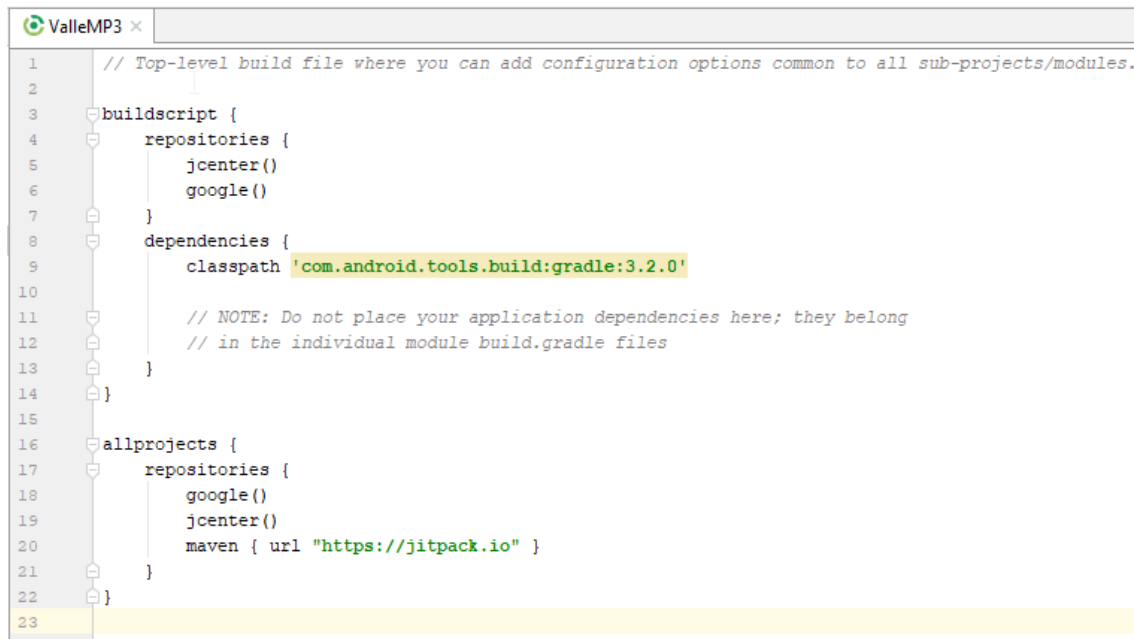


Ilustración 4. Estructura general del proyecto.

Antes de proseguir programando la aplicación debemos crear las dependencias acordes al proyecto que tenemos entre líneas de código para que no falle – como detallaremos en los siguientes puntos del documento – y, además, también completaremos otros archivos esenciales.

Estos archivos son el *build.gradle* (Project: ValleMP3) y *build.gradle* (Module:app).

- *build.gradle* (Project: ValleMP3)



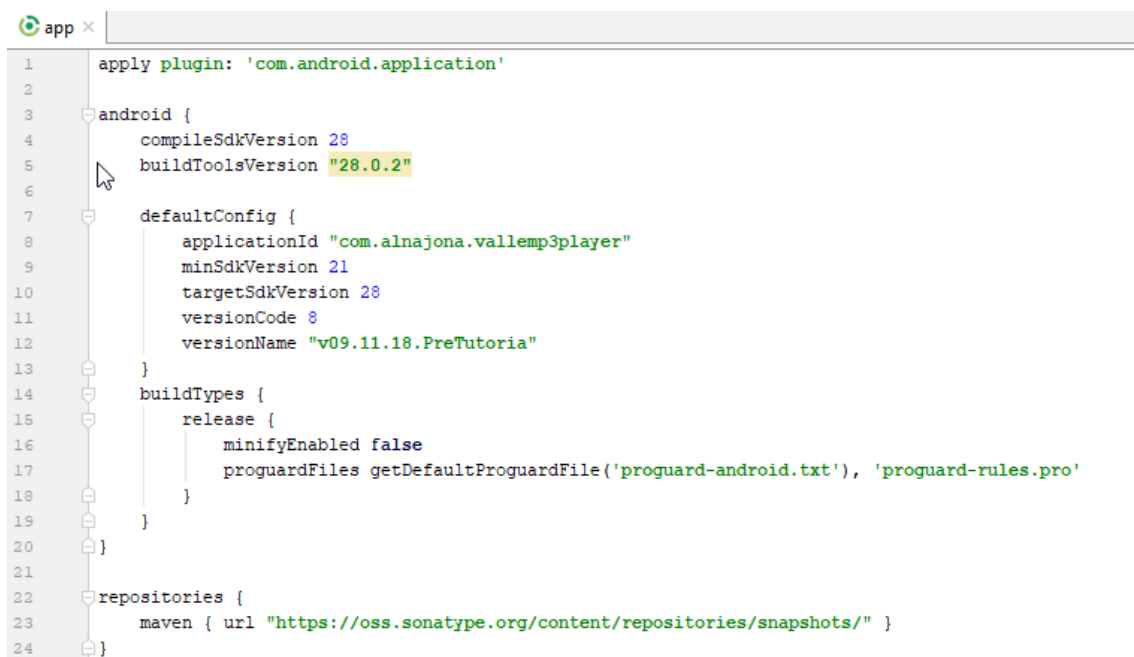
```

1 // Top-level build file where you can add configuration options common to all sub-projects/modules.
2
3 buildscript {
4     repositories {
5         jcenter()
6         google()
7     }
8     dependencies {
9         classpath 'com.android.tools.build:gradle:3.2.0'
10
11         // NOTE: Do not place your application dependencies here; they belong
12         // in the individual module build.gradle files
13     }
14 }
15
16 allprojects {
17     repositories {
18         google()
19         jcenter()
20         maven { url "https://jitpack.io" }
21     }
22 }
23

```

Ilustración 5. *build.gradle* (Project: ValleMP3).

- *build.gradle* (Module:app)



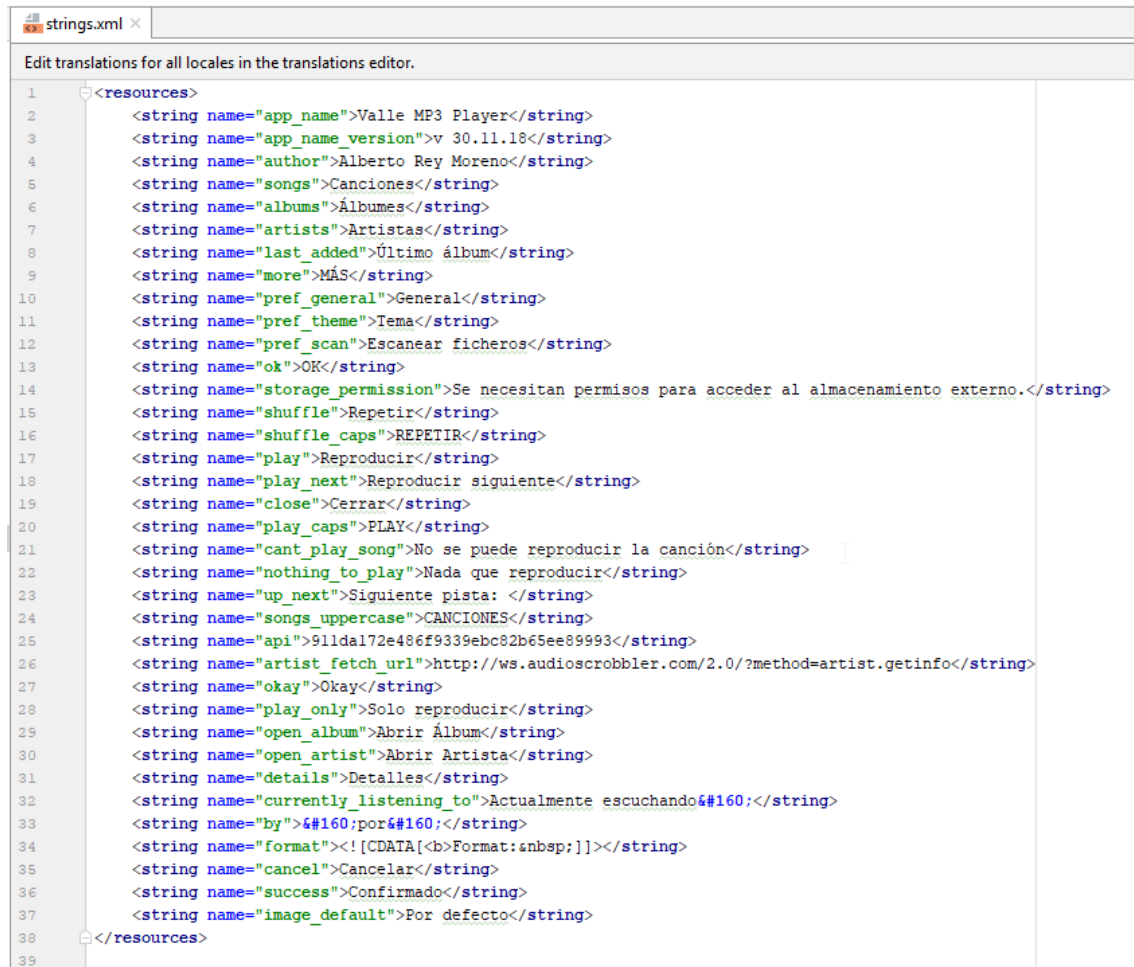
```

1 apply plugin: 'com.android.application'
2
3 android {
4     compileSdkVersion 28
5     buildToolsVersion "28.0.2"
6
7     defaultConfig {
8         applicationId "com.alnajona.vallemp3player"
9         minSdkVersion 21
10        targetSdkVersion 28
11        versionCode 8
12        versionName "v09.11.18.PreTutoria"
13    }
14    buildTypes {
15        release {
16            minifyEnabled false
17            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18        }
19    }
20 }
21
22 repositories {
23     maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
24 }
25

```

Ilustración 6. *build.gradle* (Module:app).

- Variables Strings XML.



```

1  <resources>
2      <string name="app_name">Valle MP3 Player</string>
3      <string name="app_name_version">v 30.11.18</string>
4      <string name="author">Alberto Rey Moreno</string>
5      <string name="songs">Canciones</string>
6      <string name="albums">Álbumes</string>
7      <string name="artists">Artistas</string>
8      <string name="last_added">Último álbum</string>
9      <string name="more">MÁS</string>
10     <string name="pref_general">General</string>
11     <string name="pref_theme">Tema</string>
12     <string name="pref_scan">Escanear ficheros</string>
13     <string name="ok">OK</string>
14     <string name="storage_permission">Se necesitan permisos para acceder al almacenamiento externo.</string>
15     <string name="shuffle">Repetir</string>
16     <string name="shuffle_caps">REPETIR</string>
17     <string name="play">Reproducir</string>
18     <string name="play_next">Reproducir siguiente</string>
19     <string name="close">Cerrar</string>
20     <string name="play_caps">PLAY</string>
21     <string name="cant_play_song">No se puede reproducir la canción</string>
22     <string name="nothing_to_play">Nada que reproducir</string>
23     <string name="up_next">Siguiente pista: </string>
24     <string name="songs_uppercase">CANCIONES</string>
25     <string name="api">911dal72e486f9339ebc82b65ee89993</string>
26     <string name="artist_fetch_url">http://ws.audioscrobbler.com/2.0/?method=artist.getinfo</string>
27     <string name="okay">Okay</string>
28     <string name="play_only">Solo reproducir</string>
29     <string name="open_album">Abrir Álbum</string>
30     <string name="open_artist">Abrir Artista</string>
31     <string name="details">Detalles</string>
32     <string name="currently_listening_to">Actualmente escuchando<#160;</string>
33     <string name="by"><#160;por<#160;</string>
34     <string name="format"><![CDATA[<b>Format:<#160;]]></string>
35     <string name="cancel">Cancelar</string>
36     <string name="success">Confirmado</string>
37     <string name="image_default">Por defecto</string>
38 </resources>
39

```

Ilustración 7. Strings.xml (ES).

En las siguientes páginas vamos a desgranar las clases, interfaces y métodos más importantes e interesantes para entender como hemos desarrollado la aplicación de *Valle MP3 Player*.

- **PlayerService.**

Con dicha clase controlamos todos los aspectos y acciones de la reproducción de la aplicación, aunque se complementa con *utils* → *PlayerDBHandler*, *utils* → *PlayerHandler* y *utils* → *ListSongs*.

```
public class PlayerService extends Service {

    public static final String ACTION_PLAY_SINGLE = "ACTION_PLAY_SINGLE";
    public static final String ACTION_PLAY_ALL_SONGS = "ACTION_PLAY_ALL_SONGS";
    public static final String ACTION_PLAY_ALBUM = "ACTION_PLAY_ALBUM";
    public static final String ACTION_PLAY_PLAYLIST = "ACTION_PLAY_PLAYLIST";
    public static final String ACTION_PLAY_ARTIST = "ACTION_PLAY_ARTIST";
    public static final String ACTION_GET_SONG = "ACTION_GET_SONG";
    public static final String ACTION_NOTI_CLICK = "ACTION_NOTI_CLICK";
    public static final String ACTION_NOTI_REMOVE = "ACTION_NOTI_REMOVE";
    public static final String ACTION_CHANGE_SONG = "ACTION_CHANGE_SONG";
    public static final String ACTION_SEEK_SONG = "ACTION_SEEK_SONG";
    public static final String ACTION_NEXT_SONG = "ACTION_NEXT_SONG";
    public static final String ACTION_PREV_SONG = "ACTION_PREV_SONG";
    public static final String ACTION_PAUSE_SONG = "ACTION_PAUSE_SONG";
    public static final String ACTION_ADD_QUEUE = "ACTION_ADD_QUEUE";
    private static final String TAG = "PlayerService-TAG";
    private PlayerHandler musicPlayerHandler;
    private Context context;
    private ListType listType;
    private NotificationHandler notificationHandler;
    private BroadcastReceiver playerServiceBroadcastReceiver = (context, intent) → {
        try {
            handleBroadcastReceived(context, intent);
        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(context, PlayerService.this, "No se puede reproducir la canción", Toast.LENGTH_SHORT).show();
        }
    };
};
```

Ilustración 8. PlayerService Class

Como vemos en la captura de pantalla anterior, hemos establecido las variables de todas las acciones de nuestro reproductor. Adelantándonos en el código podemos observar que las interacciones entre el usuario y la aplicación se ha creado mediante un *switch case*.



```
private void handleBroadcastReceived(Context context, final Intent intent) throws IOException {
    switch (intent.getAction()) {
        case ACTION_PLAY_SINGLE:
            musicPlayerHandler.playSingleSong(intent.getLongExtra( name: "songId", defaultValue: 0));
            listType = ListType.SINGLE;
            updatePlayer();
            break;
        case ACTION_PLAY_ALL_SONGS:
            musicPlayerHandler.playAllSongs(intent.getLongExtra( name: "songId", defaultValue: 0));
            listType = ListType.ALL;
            updatePlayer();
            break;
        case ACTION_PLAY_ALBUM:
            musicPlayerHandler.playAlbumSongs(intent.getLongExtra( name: "albumId", defaultValue: 0),
                intent.getIntExtra( name: "songPos", defaultValue: 0));
            listType = ListType.ALBUM;
            updatePlayer();
            break;
        case ACTION_GET_SONG:
            try {
                updatePlayer();
            } catch (ArrayIndexOutOfBoundsException e) {
                e.printStackTrace();
            }
            break;
        case ACTION_NEW_SONG:
    
```

Ilustración 9. Switch Case ACTIONS.

Para actualizar el reproductor de las notificaciones o el mismo reproductor de la aplicación usamos dos métodos distintos que llaman a su vez a las propiedades de *PlayerHandler* (del que ya hablaremos en puntos más adelante).

```
public void updatePlayer() {
    Intent i = new Intent();
    i.setAction(PlayerActivity.ACTION_RECIEVE_SONG);
    i.putExtra( name: "songId", musicPlayerHandler.getCurrentPlayingSongId());
    i.putExtra( name: "songName", musicPlayerHandler.getCurrentPlayingSong().getName());
    i.putExtra( name: "albumId", musicPlayerHandler.getCurrentPlayingSong().getAlbumId());
    i.putExtra( name: "seek", musicPlayerHandler.getMediaPlayer().getCurrentPosition());
    i.putExtra( name: "pos", musicPlayerHandler.getCurrentPlayingPos());
    sendBroadcast(i);
    updateNotificationPlayer();
}

private void updateNotificationPlayer() {
    if (!notificationHandler.isNotificationActive())
        notificationHandler.setNotificationPlayer(false);
    notificationHandler.changeNotificationDetails(musicPlayerHandler
        .getCurrentPlayingSong().getName(), musicPlayerHandler
        .getCurrentPlayingSong().getArtist(), musicPlayerHandler
        .getCurrentPlayingSong().getAlbumId(), musicPlayerHandler
        .getMediaPlayer().isPlaying());
}

```

Ilustración 10. Actualizaciones del reproductor y notificaciones.

- **PlayerDBHandler**

Con esta clase creamos, modificamos y actualizamos la pequeña base de datos integrada que usamos para listar las canciones del terminal en la aplicación gracias a *SQLiteDatabase*.

```
public class PlayerDBHandler extends SQLiteOpenHelper {

    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "PlaybackDB";
    public static final String TABLE_PLAYBACK = "songs";
    public static final String SONG_KEY_ID = "song_id";
    public static final String SONG_KEY_REAL_ID = "song_real_id";
    public static final String SONG_KEY_LAST_PLAYED = "song_last_played";
    private static final String TAG = "PlayerDBHandler-TAG";
    private Context context;
    private int fetchedPlayingPos = -1;

    public PlayerDBHandler(Context context) {
        super(context, DATABASE_NAME, factory: null, DATABASE_VERSION);
        this.context = context;
    }
}
```

Ilustración 11. PlayerDBHandler. SQLiteDatabase.

Para poder establecer la imagen del álbum listado debemos crear dentro del método *setAlbumArt*, un cursor al que le indicaremos que ejecute una sentencia por la cual nos devolverá la imagen del álbum con la ID que le surtimos.

```
public String setAlbumArt(long albumId) {
    Cursor cursor = context.getContentResolver().query(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI,
        new String[]{MediaStore.Audio.Albums._ID, MediaStore.Audio.Albums.ALBUM_ART},
        selection: MediaStore.Audio.Albums._ID + "=?",
        new String[]{String.valueOf(albumId)},
        sortOrder: null);
    if (cursor != null && cursor.moveToFirst()) {
        return cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Albums.ALBUM_ART));
    }
    return null;
}
```

Ilustración 12. Método de como mostrar la imagen de los álbumes.

- **ListSongs**

Con la clase *ListSongs* obtendremos las listas de artistas, álbumes o canciones en varios *ArrayList <Object>* para poder acceder a ellas de una manera más rápida al listarlas en el *Layout* correspondiente de la aplicación.

```
public static ArrayList<Album> getAlbumList(Context context) {
    final ArrayList<Album> albumList = new ArrayList<>();
    System.gc();
    final String orderBy = MediaStore.Audio.Albums.ALBUM;
    Cursor musicCursor = context.getContentResolver().
        query(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI, projection: null, selection: null, selectionArgs: null, orderBy);

    if (musicCursor != null && musicCursor.moveToFirst()) {
        //get columns
        int titleColumn = musicCursor.getColumnIndex
            (android.provider.MediaStore.Audio.Albums.ALBUM);
        int idColumn = musicCursor.getColumnIndex
            (android.provider.MediaStore.Audio.Albums._ID);
        int artistColumn = musicCursor.getColumnIndex
            (android.provider.MediaStore.Audio.Albums.ARTIST);
        int numofSongsColumn = musicCursor.getColumnIndex
            (android.provider.MediaStore.Audio.Albums.NUMBER_OF_SONGS);
        int albumArtColumn = musicCursor.getColumnIndex
            (android.provider.MediaStore.Audio.Albums.ALBUM_ART);
        //add albums to list
        do {
            albumList.add(new Album(musicCursor.getLong(idColumn),
                musicCursor.getString(titleColumn),
                musicCursor.getString(artistColumn),
                fav: false, musicCursor.getString(albumArtColumn),
                musicCursor.getInt(numofSongsColumn)));
        } while (musicCursor.moveToNext());
    }
    return albumList;
}
```

Ilustración 13. Obtener la lista de álbumes del reproductor.

- **PlayerHandler**

Con la clase *PlayerHandler* damos funcionalidad a todas las acciones ya determinadas en *PlayerService*, o sea, creamos las funcionalidades de reproducir, parar, pausar, pasar hacia adelante o atrás la pista, repetir pista o lista o reproducir aleatoriamente la lista de reproducción.

```

public void playOrStop(NotificationHandler notificationHandler) {
    boolean state;
    if (mediaPlayer.isPlaying()) {
        mediaPlayer.pause();
        state = false;
        service.stopForeground( removeNotification: false);
        notificationHandler.setNotificationPlayer(true);
    } else {
        mediaPlayer.start();
        state = true;
        notificationHandler.setNotificationPlayer(false);
    }
    notificationHandler.updateNotificationView();
    notificationHandler.changeNotificationDetails (getCurrentPlayingSong().getName(),
        getCurrentPlayingSong().getArtist(), getCurrentPlayingSong().getAlbumId(), state);
}

```

Ilustración 14. Método para Reproducir y Parar.

Para configurar el reproductor usamos el siguiente método por el cual se para aquella reproducción si la hubiera en marcha, y establece un reinicio de todas las variables con la clase *MediaPlayer* y sus propios métodos.

```

public void configurePlayer(final int pos) throws IOException {
    stopPlayer();
    setPlayingPos(pos);
    mediaPlayer.reset();
    mediaPlayer.setDataSource(currentPlayingSongs
        .get(pos).getPath());
    mediaPlayer.prepare();
    mediaPlayer.start();
    mediaPlayer.setOnCompletionListener((mediaPlayer) → {
        try {
            playNextSong(getNextSongPosition(pos));
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}

```

Ilustración 15. Configuración del reproductor.

## ALBERTO REY MORENO – VALLE MP3 PLAYER

En los siguientes métodos podemos comprobar como reproducimos la siguiente o anterior pista de audio. Si vamos a reproducir la siguiente pista solo tenemos que configurar la siguiente posición del álbum; y si queremos reproducir la anterior hay que reiniciar y establecer el servicio del reproductor en la pista anterior del álbum.

```
public void playNextSong(final int nextSongPos) throws IOException {
    if (nextSongPos < currentPlayingSongs.size()) {
        configurePlayer(nextSongPos);
    } else if (preferenceHandler.isRepeatAllEnabled()) {
        configurePlayer( pos: 0);
    }
    service.updatePlayer();
}

public void playPrevSong(final int prevSongPos) throws IOException {
    if ((mediaPlayer.getCurrentPosition() / 1000) <= 2) {
        int position;
        if (prevSongPos == -1 && preferenceHandler.isRepeatAllEnabled()) {
            position = getCurrentPlayingSongs().size() - 1;
        } else if (prevSongPos == -1) {
            position = 0;
        } else
            position = prevSongPos;
        final int pos = position;
        stopPlayer();
        setPlayingPos(pos);
        mediaPlayer.reset();
        mediaPlayer.setDataSource(currentPlayingSongs.get(pos).getPath());
        mediaPlayer.prepare();
        mediaPlayer.start();
        mediaPlayer.setOnCompleteListener((mediaPlayer) -> {
            try {
                playNextSong(getNextSongPosition(prevSongPos));
            } catch (IOException e) {
                e.printStackTrace();
                Toast.makeText(context, "No se puede reproducir la canción",
                    Toast.LENGTH_SHORT).show();
            }
        });
    } else {
        mediaPlayer.seekTo( msec: 0);
    }
}
```

Ilustración 16. Reproducción de las pistas posteriores y anteriores en el reproductor.

- **Animaciones en la aplicación**

Con las clases *AlbumItemLoad*, con dicha clase cargamos las animaciones creadas por las siguientes clases como también los nombres y carátulas de los álbumes en su lugar correspondiente; *ColorChangeAnimation*, con esta clase se decide el color predominante de la carátula del álbum o pone un color por defecto si no encuentra uno adecuado; e *ImageBlurAnimator*, case por la cual creamos las animaciones y la duración de las mismas (por ejemplo, el degradado entre los colores de pistas o álbumes, y su duración). Un claro ejemplo de esto es cuando pasamos de un álbum en reproducción a otro.

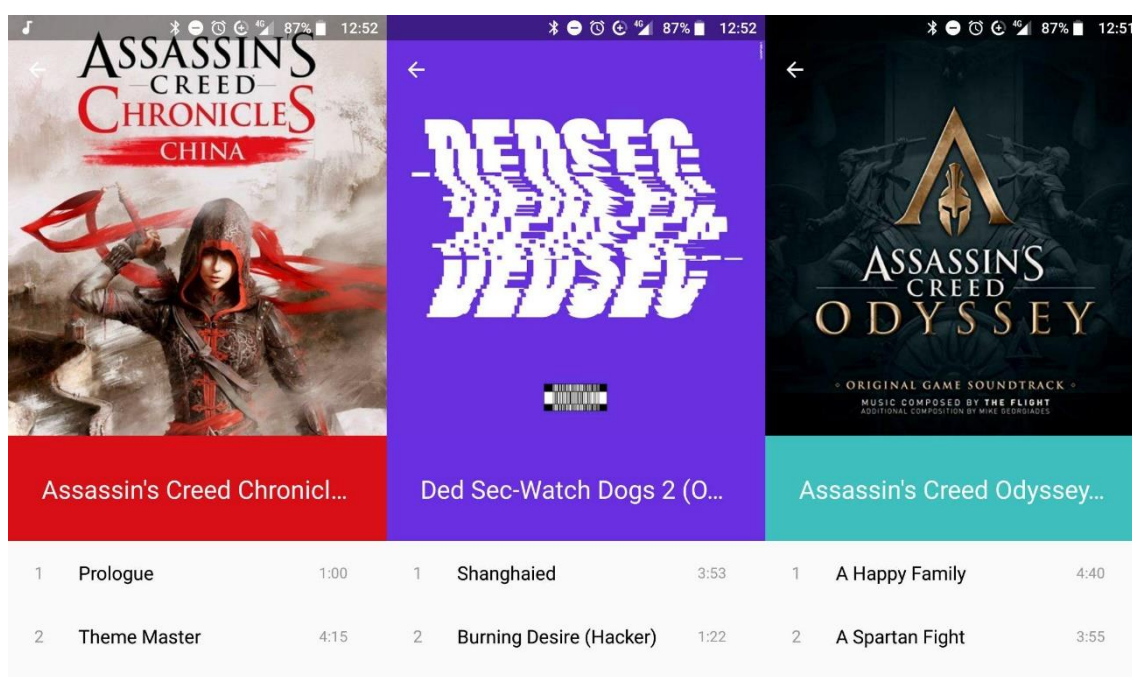


Ilustración 17. Ejemplo del cambio de color gracias a las animaciones incluidas.

Solo necesitamos crear un objeto por el cual le vamos a dar la opción de que lea la imagen del álbum e intente dar con el color predominante de la misma imagen o uno que se aproxime para poder hacer la animación degradada entre el inicio

del reproductor y cuando se pone en marcha la reproducción o cuando se intercambian pistas, álbumes o artistas en la aplicación.

```
public abstract class ColorChangeAnimation extends Action {

    private Context context;
    private LinearLayout detailHolder;
    private String artPath;
    private boolean noBitmap;
    private Integer colorFrom;
    private ValueAnimator colorAnimation;

    public ColorChangeAnimation(Context context, LinearLayout detailHolder, String artPath) {
        this.context = context;
        this.detailHolder = detailHolder;
        this.artPath = artPath;
        colorFrom = ((ColorDrawable) detailHolder.getBackground()).getColor();
        noBitmap = false;
    }
}
```

Ilustración 18. ColorChangeAnimation - Constructor.

```
@Nullable
@Override
protected Object run() throws InterruptedException {
    try {
        Bitmap bmp = BitmapFactory.decodeFile(artPath);
        Palette.from(bmp).generate(
            (palette) -> {
                Integer colorTo = palette.getVibrantColor(palette.getDarkVibrantColor(
                    palette.getDarkMutedColor(palette.getMutedColor(
                        ContextCompat.getColor(context,
                            R.color.colorPrimary)))));
                onColorFetched(colorTo);
                colorAnimation = ValueAnimator.ofObject(new ArgbEvaluator(), colorFrom, colorTo);
                colorAnimation.setDuration(2000);
                colorAnimation.addUpdateListener((animator) -> {
                    detailHolder.setBackgroundColor((Integer) animator.getAnimatedValue());
                });
                colorAnimation.start();
            }
        );
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
        noBitmap = true;
    }
    return null;
}
```

Ilustración 19. Ejecución del run de ColorChangeAnimation.

La misma animación de colores se aplica cuando minimizamos la aplicación y la intentamos abrir desde el panel de tareas, como mostramos en la imagen que está en la siguiente página de este manual.



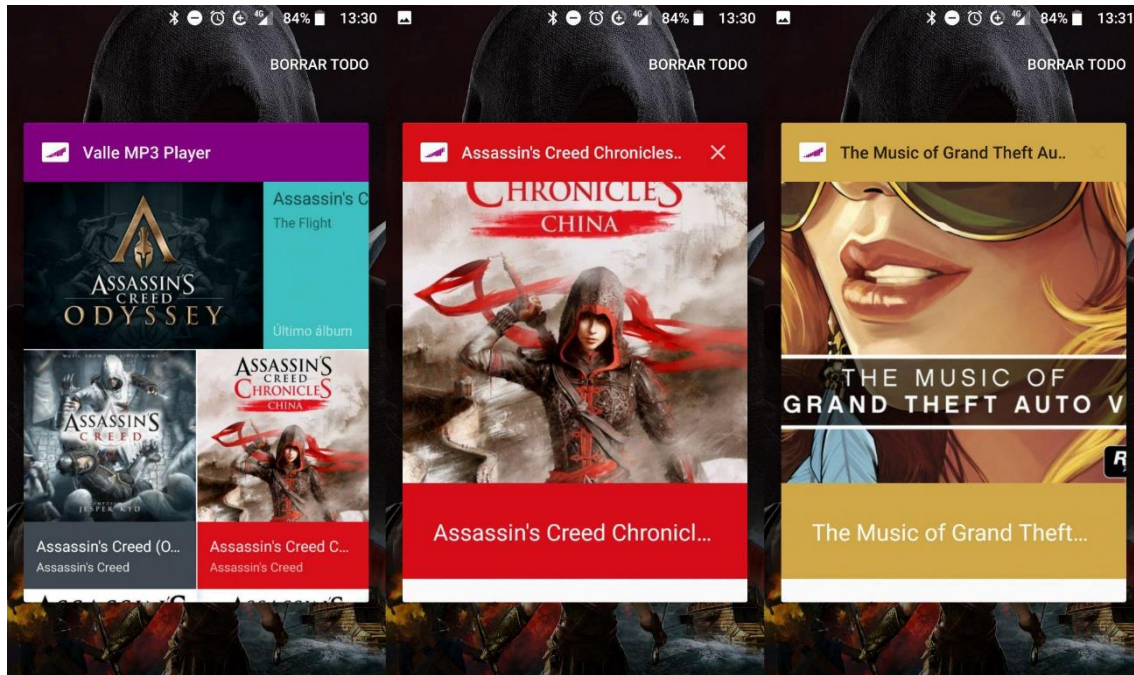


Ilustración 20. Animación de color en la vista de tareas.

- **PermissionChecker**

Pero todo el desarrollo se lleva centrandose en la reproducción de medios externos a la aplicación y para ello se necesita que el usuario del terminal acepte que se escaneen y se listen los archivos MP3 mediante una simple pregunta o permiso, según la versión de Android instalada en el terminal.



```

public void check(final String permission, final String customMsg, final OnPermissionResponse response) {
    this.response = response;
    new Thread((Runnable) () -> {
        if (ContextCompat.checkSelfPermission(context, permission) ==
            PackageManager.PERMISSION_GRANTED) {
            activity.runOnUiThread() -> {
                response.onAccepted();
            };
        } else {
            Log.v(TAG, msg: "Waiting");
            if (ActivityCompat.shouldShowRequestPermissionRationale((Activity) context,
                permission)) {
                Snackbar.make(baseView, customMsg,
                    Snackbar.LENGTH_INDEFINITE)
                    .setAction("OK", (view) -> {
                        ActivityCompat.requestPermissions((Activity) context,
                            new String[]{permission},
                                REQUEST_CODE);
                    })
                    .show();
            } else {
                ActivityCompat.requestPermissions(((Activity) context),
                    new String[]{permission},
                        REQUEST_CODE);
            }
        }
    }).start();
}

```

Ilustración 21. Método para pedir la aceptación de permisos al usuario final.

Los permisos se piden de dos maneras, al instalar la aplicación salta una notificación *pop-up* la cual nos avisa de que la aplicación necesita de nuestro permiso para poder acceder tanto al almacenamiento interno como externo del terminal.

Si aceptamos el requerimiento la aplicación se inicia correctamente para empezar a reproducir; si por el contrario denegamos el requerimiento, la aplicación se cerrará y habrá que volver a abrirla para que se nos muestre una notificación en la parte de abajo del layout de inicio para que se nos vuelva a mostrar el *pop-up* de requerimiento de permisos.

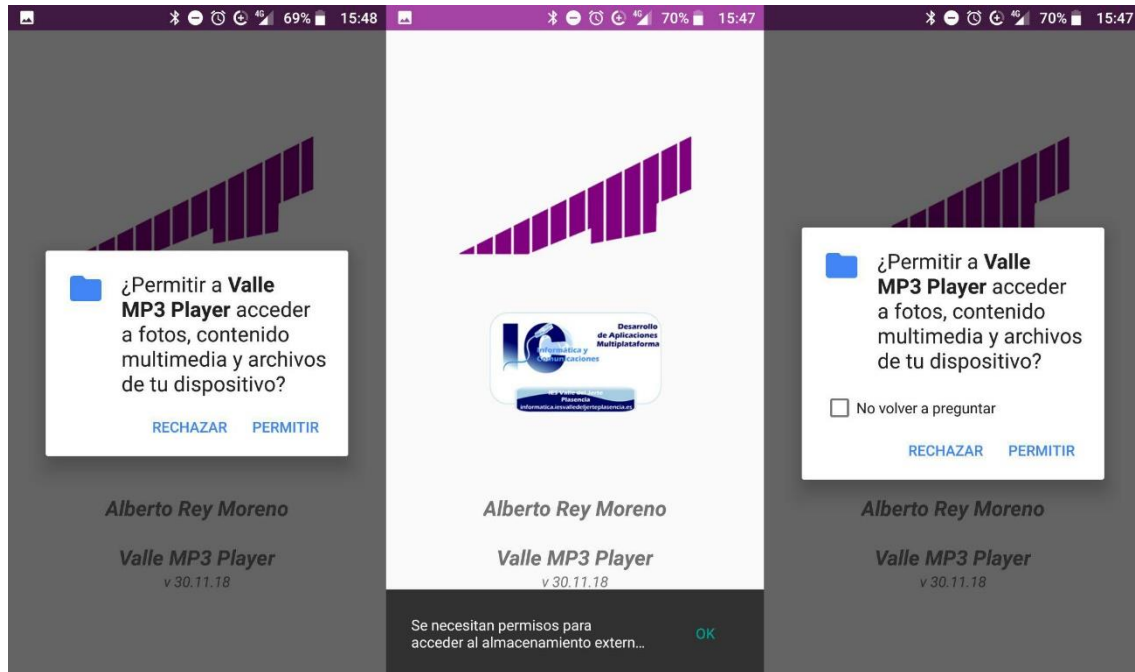
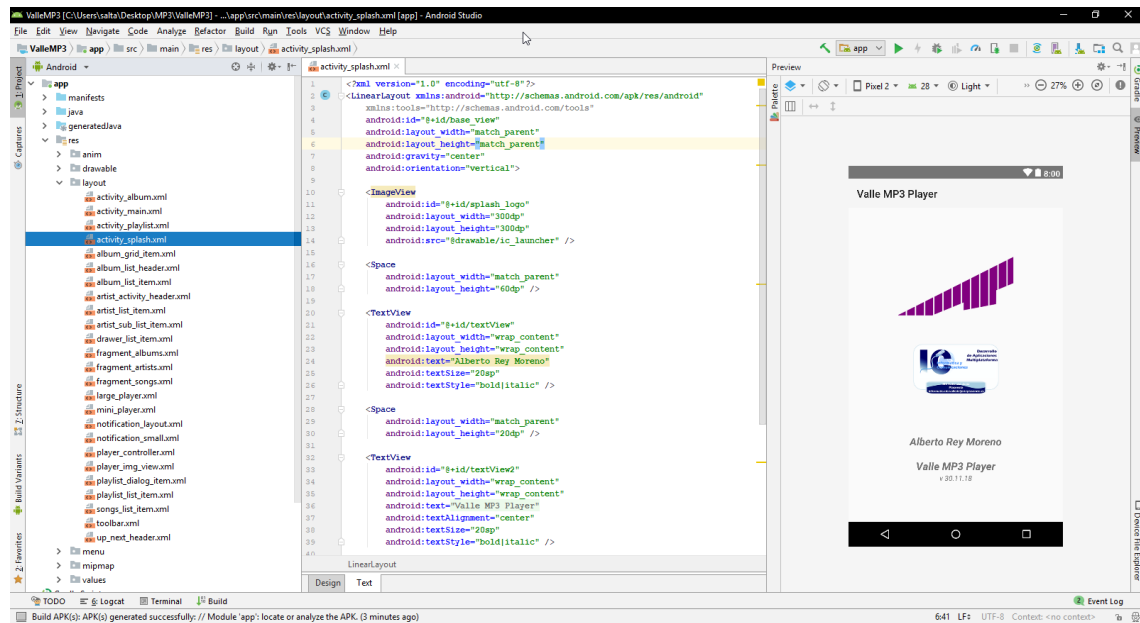


Ilustración 22. Proceso de requerimiento de acceso al contenido para la aplicación.

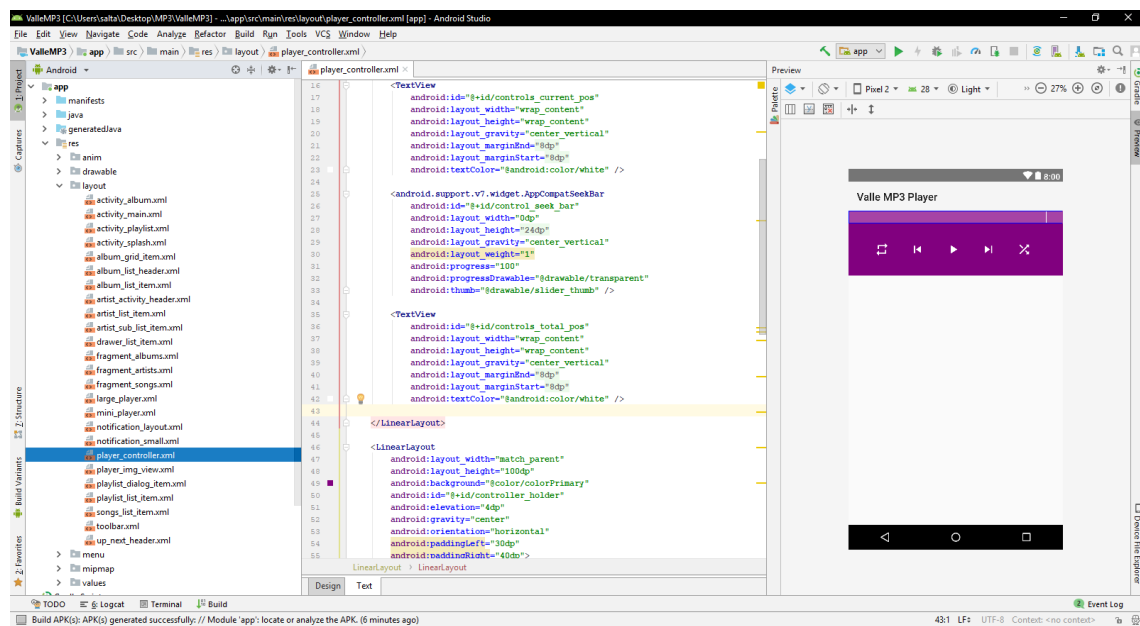
Con estas clases explicadas con bastante detalle ya sabemos como funciona el reproductor de música que tenemos entre manos y como todo proyecto, ahora tendremos que mostrar como va vestido para la ocasión, o sea, los *layouts* más importantes del proyecto.

Comenzamos con el *Layout* con el que se inicia la aplicación y sirve para que el reproductor se ponga en marcha en unos diez o doce segundos, hablamos del *activity\_splash.xml*.

- Layout → activity\_splash.xml

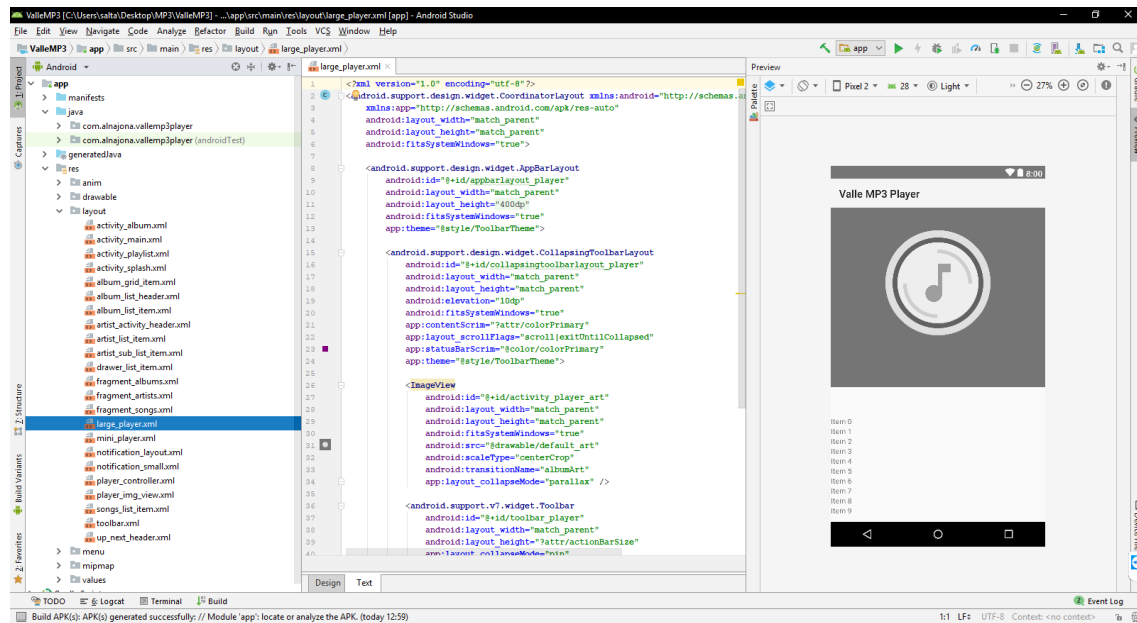


- Layout → player\_controller.xml

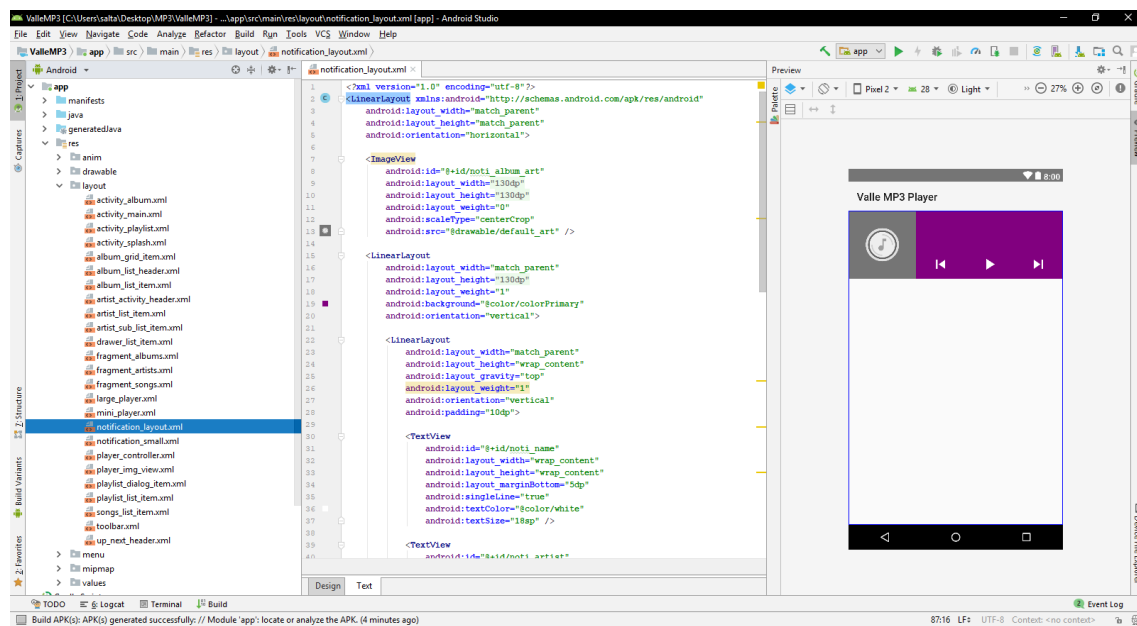


## ALBERTO REY MORENO – VALLE MP3 PLAYER

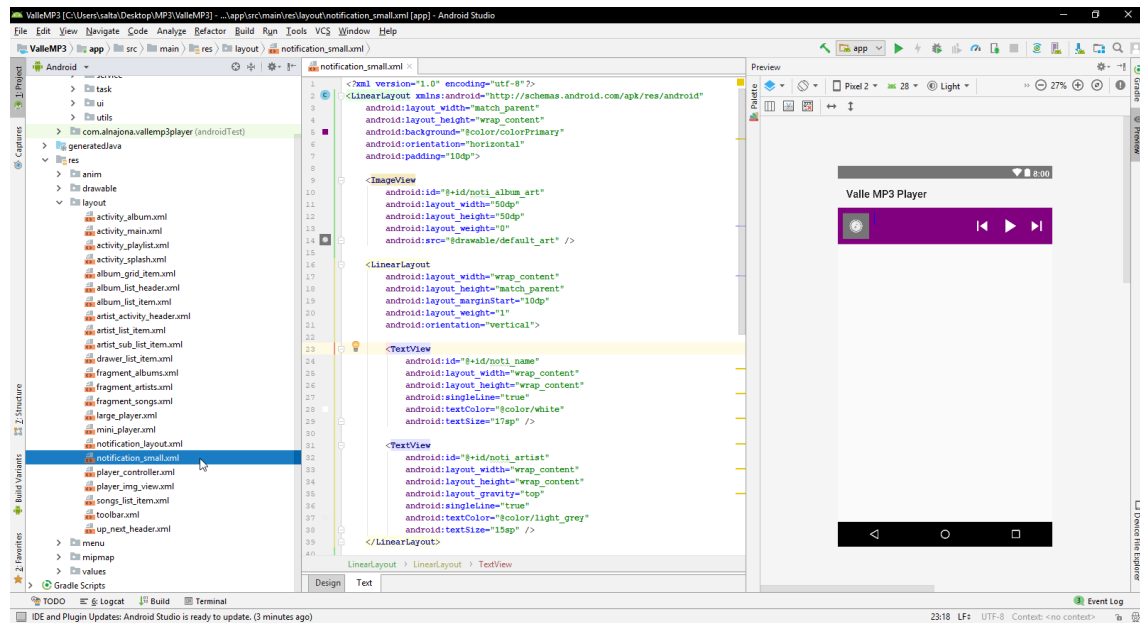
- Layout → large\_player.xml



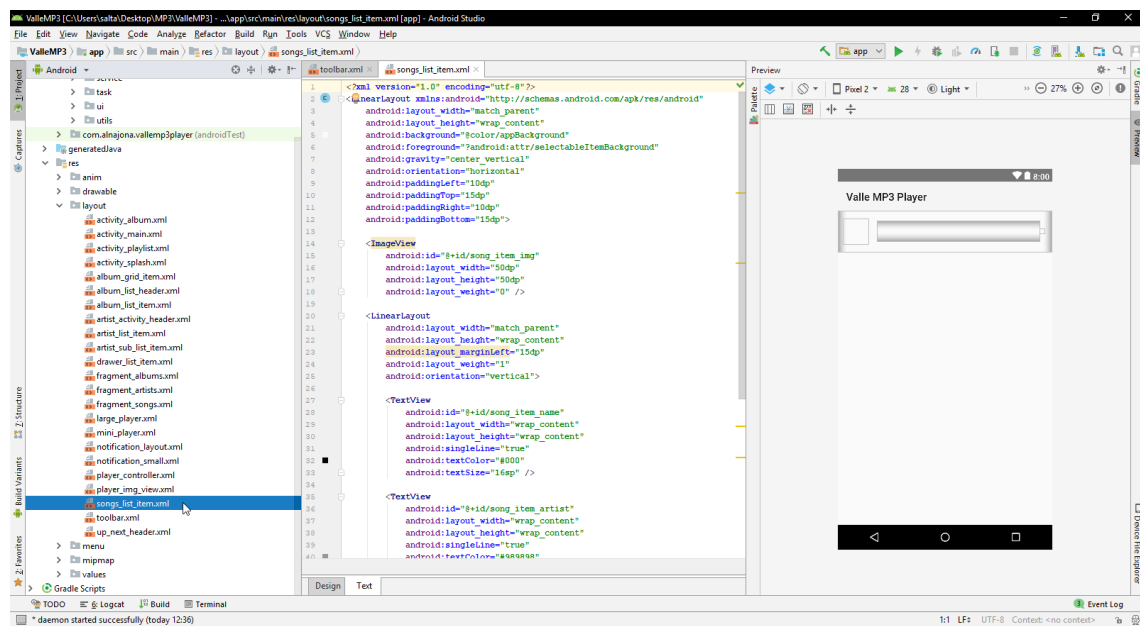
- Layout → notification\_layout.xml



- Layout → notification\_small.xml



- Layout → song\_list\_item.xml



Con esto termina nuestra explicación del desarrollo de la aplicación *Valle MP3 Player*.

### **c. FUTURO DE LA APLICACIÓN**

En un futuro nos gustaría portar la aplicación de Android a iOS, como desarrollar otras funciones en la misma aplicación que por el limitado tiempo de desarrollo han sido imposible de implementar.

Posiblemente, lo que más nos ha costado dejar atrás ha sido la posibilidad de meter un menú con opciones de un tema oscuro (nocturno), la inclusión de las listas de reproducción con creación de las mismas y con posibilidad de importar/exportar en formato *XML* (Extensible Markup Language – Lenguaje de Marcado Extensible), o la inclusión de un campo capaz de la reproducción de medios online como *YouTube*.

### **d. ERRORES DEL DESARROLLO**

Durante todo el desarrollo de la aplicación nos hemos tenido que enfrentar a más de un error debido a problemas con las versiones de Android Studio, con el código compilado y con algunas decisiones del desarrollo.

Principalmente, los errores a los que nos hemos enfrentado han sido al actualizar Android Studio de versión. Pasar de la versión 2.3.3 (162.4069837 – 8 de junio de 2017) a la versión más reciente 3.2.1 (181.5056338 – 11 de octubre de 2018) comprendía un gran salto en el proyecto y hacia fallar todo lo que ya teníamos desarrollado por lo que dictaminamos que sería mejor instalar una versión menos

reciente de Android Studio para ser exactos la versión 3.2.0 (181.5014246 – 24 de septiembre de 2018).

Otro de los principales errores de actualizar cualquier proyecto creado con Android Studio antes de las versiones 3.0 de este año son las dependencias. Gracias al desarrollo de Android Studio se ha mejorado estas dependencias y por ende, su forma de desarrollo.

```
buildscript {
    repositories {
        jcenter()
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
        maven { url "https://jitpack.io" }
    }
}
```

Aquí están las imágenes para su comparación, entre la imagen de arriba perteneciente a la versión 2.3.3 de Android Studio y la imagen de abajo perteneciente a la versión 3.2.0 del mismo programa.

## ALBERTO REY MORENO – VALLE MP3 PLAYER

```
buildscript {
    repositories {
        jcenter()
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.0'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        maven { url "https://jitpack.io" }
    }
}
```

Además, de los problemas con las versiones del programa Android Studio hemos tenido otros problemas como la elección del lenguaje de programación base entre Java hasta ahora o Kotlin.





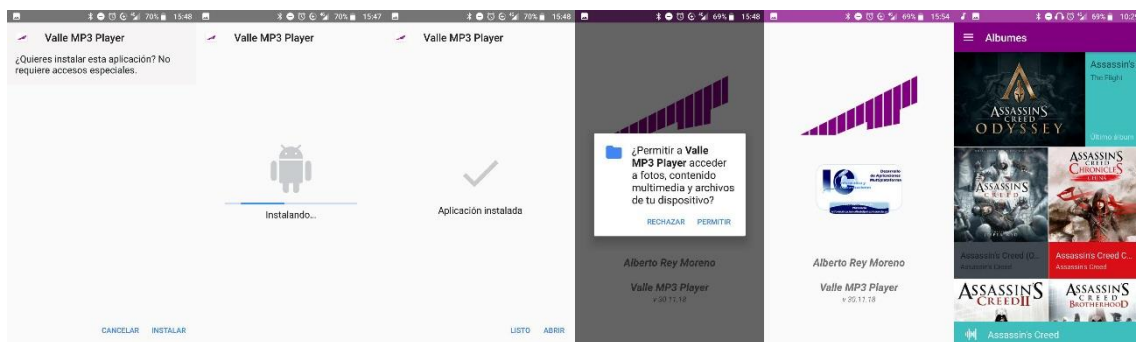
## ALBERTO REY MORENO – VALLE MP3 PLAYER

Otro de los errores durante el desarrollo ha sido que tras las muchas demos instaladas, actualizadas y desinstaladas de nuestro terminal hemos tenido problemas para reinstalar la aplicación dando lugar al error de *Aplicación no instalada* causada por la cache de la misma en los terminales de prueba. Teniendo que desactivar en los terminales más actuales la opción de *Google Protect* de la *Play Store*.

### e. MANUAL DEL USUARIO

Para utilizar nuestra aplicación solo tendrás que ir al perfil de *IES Valle del Jerte Plasencia*, descargarte la aplicación *Valle MP3 Player* y disfrutar de este magnífico reproductor.

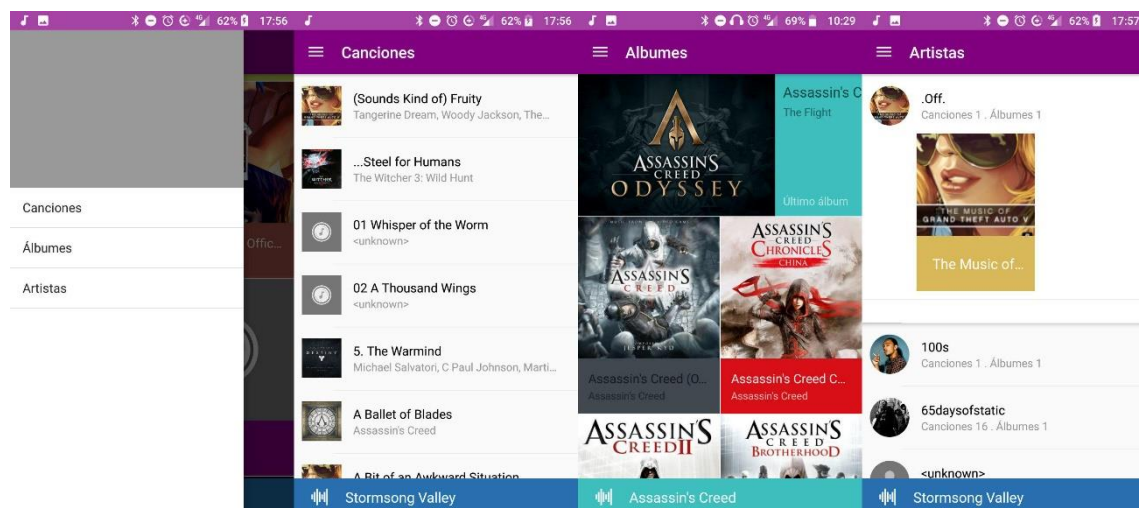
Al iniciar nos pedirá que resolvamos la pregunta requerida de permisos, después empezará a escanear y listar nuestros álbumes de música mostrándose los álbumes obtenidos en la búsqueda.



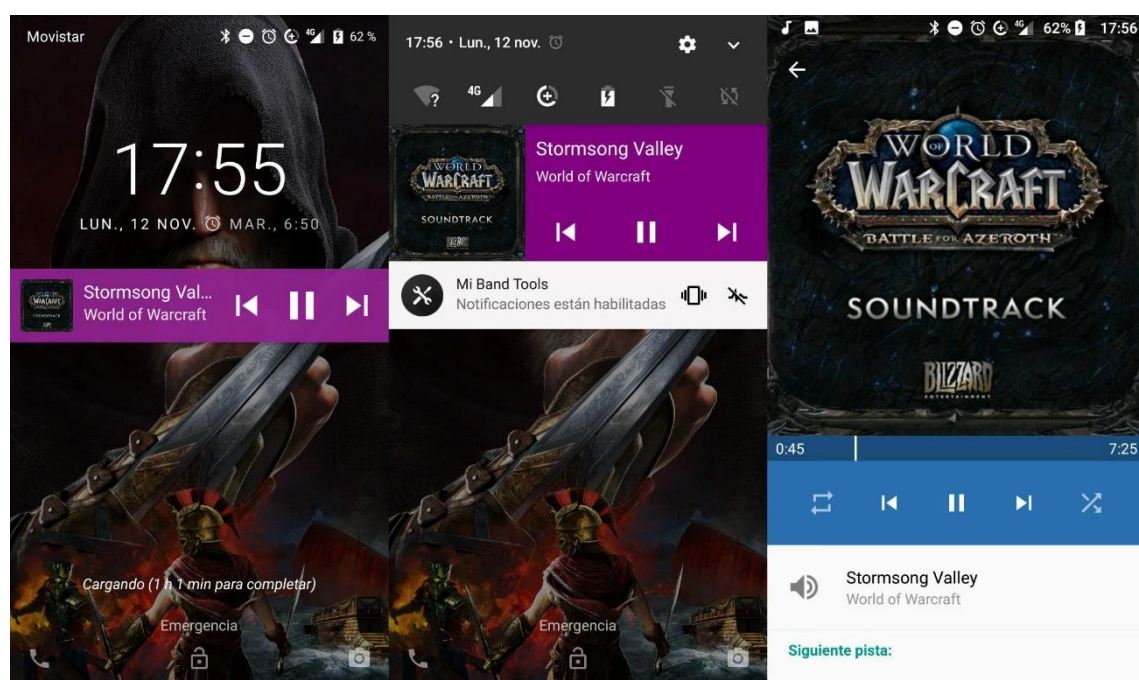
A partir de aquí, solo necesitaremos acceder a cualquiera de los álbumes para reproducir cualquier pista de audio de cualquier álbum listado.

## ALBERTO REY MORENO – VALLE MP3 PLAYER

Desde el menú accederemos a las distintas vistas de la aplicación que son *Canciones*, *Álbumes* y *Artistas*.



Además, desde el símbolo de las ondas accederemos si estamos reproduciendo cualquier pista de audio al gran reproductor y si bloqueamos el terminal accederemos al reproductor desde el panel de notificaciones o desde la propia pantalla bloqueante.



## **5. MEDIOS UTILIZADOS.**

### **HARDWARE:**

- Samsung Galaxy S2. Mayo de 2011.
- BQ Aquaris M5. Julio de 2015.
- Samsung Galaxy S8. Abril de 2017.

### **SOFTWARE:**

- Android Studio 2.3.3
- Android Studio 3.2.0
- Adobe Suite CC
  - Adobe Photoshop CC
  - Adobe Illustrator CC
  - Adobe Premiere Pro CC
- Notepad ++
- Trello
- Librerías:
  - Picasso
  - Material Dialogs

## 6. BIBLIOGRAFÍA.

- *Android Developers – MediaPlayer*
  - <https://developer.android.com/reference/android/media/MediaPlayer>
- *Android Developers – MediaStore*
  - <https://developer.android.com/reference/android/provider/MediaStore>
- *Android Developers – SQLiteDatabase*
  - <https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase>
- *Android Developers – Guardar datos con SQLiteDatabase*
  - <https://developer.android.com/training/data-storage/sqlite>
- *Android Developers – Animation Overview*
  - <https://developer.android.com/training/animation/overview>
- *Android Developers – View Animation*
  - <https://developer.android.com/guide/topics/graphics/view-animation>
- *Android Developers – Styles*
  - <https://developer.android.com/guide/topics/resources/style-resource>
- *EnvatoTuts+ – Create a Music Player on Android*
  - Parte 1 – <https://code.tutsplus.com/es/tutorials/create-a-music-player-on-android-project-setup--mobile-22764>
  - Parte 2 – <https://code.tutsplus.com/tutorials/create-a-music-player-on-android-user-controls--mobile-22787>

- Parte 3 – <https://code.tutsplus.com/es/tutorials/create-a-music-player-on-android-user-controls--mobile-22787>
- *Picasso – A powerful image downloading and caching library for Android*
  - <https://github.com/square/picasso>
- *Material Dialogs*
  - <https://github.com/afollestad/material-dialogs>
- *Stack Overflow – Errores en las dependencias de Android Studio 3.2.0*
  - [https://developer.android.com/studio/build/dependencies?utm\\_source=android-studio#dependency\\_configurations](https://developer.android.com/studio/build/dependencies?utm_source=android-studio#dependency_configurations)
- *Stack Overflow - ¿Cómo mostrar el ALBUM ART con MediaStore?*
  - <https://stackoverflow.com/questions/17573972/how-can-i-display-album-art-using-mediastore-audio-albums-album-art>
- *TutorialsPoint – Android Animation*
  - [https://www.tutorialspoint.com/android/android\\_animations.htm](https://www.tutorialspoint.com/android/android_animations.htm)
- *YouTube – Leer el almacenamiento externo de un terminal Android*
  - <https://www.youtube.com/watch?v=kf2fxYLOiSo>