Introducción a la Programación de Dispositivos Móviles

Trabajo Final

Reproductor de Música para Android

Profesor: Juan Manuel Rodriguez

Alumno: Gustavo Tajes Genga

Índice

Introducción 2
Requerimientos Funcionales 3
Arquitectura 3
Diseño 4
    Service 4
    Activities 12
GUI 14
Conclusión 20

Introducción

Entre los avances tecnológicos de las últimas décadas, se destacan los dispositivos que no requieren de una ubicación fija o estática para prestar servicio, conocidos como “Dispositivos móviles”. Entre ellos, se encuentran las laptops, PDAs, smartphones, smartwatches, tablets, etc. Algunas de las características que poseen son recursos de hardware limitados (cpu, memoria, batería), una conexión permanente o intermitente a una red, una pantalla como interfaz de usuario, entre otras.

El siguiente informe describe el desarrollo de un reproductor de música para los dispositivos móviles basados en el sistema operativo Android, a modo de trabajo final de cursada de la cátedra optativa: “Introducción a la Programación de Dispositivos Móviles”.

Requerimientos Funcionales

Los requerimientos funcionales solicitados por la cátedra son:

- ➢ Soporte para crear playlist personalizados.
- ➢ Reproducción en servicio corriendo en foreground.
- ➢ Control básico de la reproducción desde la notificación.
- ➢ Widget y notificación.
- ➢ Descarga y almacenamiento de letras de los temas desde algún sitio Web.

Para mejorar la experiencia del usuario, a los requerimientos mencionados anteriormente se agregaron los siguientes:

- ➢ Controles de reproducción en widget y notificación.
- ➢ Pausado de la reproducción ante una llamada entrante o la desconexión de los auriculares.
- ➢ Reducción temporal del volumen ante la llegada de una notificación.
- ➢ Al iniciar la aplicación desde el widget, continúa reproduciendo la última lista desde la posición donde finalizó anteriormente.

Arquitectura

La aplicación consta de un ***service***[1] *[MediaPlayerService]* que es responsable de controlar la reproducción mediante un objeto ***MediaPlayer***[2], como así también de recibir los eventos de *llamada, notificaciones entrantes, desconexión de auriculares, controles del widget y de la notificación* (propia de la aplicación).
La interfaz visual principal del reproductor, es una ***activity***[3] *[PlayerActivity]* mediante la cual se visualiza la lista de temas en la cola de reproducción, la carátula del álbum (de poseer, sino un set de imágenes predefinido) y un botón para acceder a la letra de la canción, en caso de haberse encontrado.
Adicionalmente, se desarrolló otra ***activity*** *[MainActivity]* que es el entrypoint de la aplicación, que a la vez actúa de container para diferentes ***fragments***[4] *[AlbumsFragment, ArtistsFragment, SongsFragment, PlaylistsFragment]* utilizados para el filtrado de las canciones por álbum, artista, canción y para la manipulación de las playlist respectivamente. Estos fragments son accedidos a través de un *panel lateral de navegación.*

[1] https://developer.android.com/guide/components/services.html?hl=es-419
[2] https://developer.android.com/guide/topics/media/mediaplayer.html
[3] https://developer.android.com/guide/components/activities.html?hl=es-419
[4] https://developer.android.com/guide/components/fragments.html?hl=es-419

Diseño

A continuación se detallan algunos componentes y métodos que se consideran relevantes para la funcionalidad de la aplicación y en la sección posterior, se exponen algunas capturas de pantalla para ampliar la descripción del funcionamiento de la misma.

Service

El *service* consta de los siguientes componentes:

- AudioManager[5]: encargado de gestionar el volumen y el *audio focus* del sistema. Éste último cambia cuando se recibe una notificación, una llamada, o cuando el usuario reproduce algún medio en otra aplicación.

```
private  AudioManager audioManager;

@Override
public void onAudioFocusChange(int focusState) {

   switch (focusState) {
       case AudioManager.AUDIOFOCUS_GAIN:

           if (mediaPlayer == null)
               initMediaPlayer();
           else if (!mediaPlayer.isPlaying())
               mediaPlayer.start();

           mediaPlayer.setVolume(1.0f, 1.0f);
           break;

       case AudioManager.AUDIOFOCUS_LOSS:
           if (mediaPlayer.isPlaying())
               mediaPlayer.stop();

           mediaPlayer.release();
           mediaPlayer = null;
           break;

       case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
           if (mediaPlayer.isPlaying())
               mediaPlayer.pause();
```

[5] https://developer.android.com/reference/android/media/AudioManager.html

```
            break;

        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
            if (mediaPlayer.isPlaying())
                mediaPlayer.setVolume(0.1f, 0.1f);

            break;
    }
}

private boolean requestAudioFocus() {
    audioManager = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
    int result = audioManager.requestAudioFocus(this,
AudioManager.STREAM_MUSIC,
            AudioManager.AUDIOFOCUS_GAIN);

    return result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED;

}

private boolean removeAudioFocus() {
    return AudioManager.AUDIOFOCUS_REQUEST_GRANTED ==
audioManager.abandonAudioFocus(this);
}
```

- BroadcastReceivers[6]:
    - becomingNoisyReceiver: es el encargado de recibir el evento de desconexión de auriculares.

```
private BroadcastReceiver becomingNoisyReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        pauseMedia();
        buildNotification(PlaybackStatus.PAUSED);
    }
};

private void registerBecomingNoisyReceiver() {
    IntentFilter intentFilter = new
IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
```

[6] https://developer.android.com/guide/components/broadcasts.html

```
    registerReceiver(becomingNoisyReceiver, intentFilter);
}
```

- playNewAudio: cuya función es reproducir una nueva canción cuando el usuario lo dispara seleccionando una canción de la lista de reproducción actual.

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

        audioIndex = intent.getIntExtra("audioIndex", audioIndex);
        audioSeek = intent.getLongExtra("audioSeek", audioSeek);
        if (audioIndex != -1 && audioIndex < audioList.size()) {
            activeAudio = audioList.get(audioIndex);
        } else {
            stopSelf();
        }

        stopMedia();
        mediaPlayer.reset();
        initMediaPlayer();
        updateMetaData();
        buildNotification(PlaybackStatus.PLAYING);
    }
};

private void register_playNewAudio() {
    IntentFilter filter = new
IntentFilter(PlayerActivity.Broadcast_PLAY_NEW_AUDIO);
    registerReceiver(playNewAudio, filter);
}
```

- mediaSession[7]: encargado de generar una sesión para transportar los controles y recibir los eventos, como así también construir y enviar la información de la canción a la notificación y al widget.

```
private MediaSessionManager mediaSessionManager;
private MediaSessionCompat mediaSession;
private MediaControllerCompat.TransportControls transportControls;
```

[7] https://developer.android.com/guide/topics/media-apps/working-with-a-media-session.html

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

   mediaSessionManager = (MediaSessionManager)
getSystemService(Context.MEDIA_SESSION_SERVICE);
   mediaSession = new MediaSessionCompat(getApplicationContext(),
"AudioPlayer");
   transportControls =
mediaSession.getController().getTransportControls();
   mediaSession.setActive(true);

mediaSession.setFlags(MediaSessionCompat.FLAG_HANDLES_TRANSPORT_CONTROLS
);

   updateMetaData();

   mediaSession.setCallback(new MediaSessionCompat.Callback() {
       @Override
       public void onPlay() {
           super.onPlay();
           if (activeAudio == null)
               return;
           resumeMedia();
           buildNotification(PlaybackStatus.PLAYING);
       }

       @Override
       public void onPause() {
           super.onPause();
           if (activeAudio == null)
               return;
           pauseMedia();
           buildNotification(PlaybackStatus.PAUSED);
       }

       @Override
       public void onSkipToNext() {
           super.onSkipToNext();
           if (activeAudio == null)
               return;
           skipToNext();
           updateMetaData();
           buildNotification(PlaybackStatus.PLAYING);
       }
```

```
        @Override
        public void onSkipToPrevious() {
            super.onSkipToPrevious();
            if (activeAudio == null)
                return;
            skipToPrevious();
            updateMetaData();
            buildNotification(PlaybackStatus.PLAYING);
        }

        @Override
        public void onStop() {
            super.onStop();
            removeNotification();
            stopSelf();
        }

        @Override
        public void onSeekTo(long position) {
            super.onSeekTo(position);
        }
    });

    Intent intent = new Intent(this, PlayerWidget.class);
    intent.setAction("TOKEN");
    intent.putExtra("tkn", mediaSession.getSessionToken());
    sendBroadcast(intent);
}

private void updateMetaData() {
    Bitmap albumArt = BitmapFactory.decodeResource(getResources(),
            R.drawable.image5);
    if (activeAudio != null) {
        mediaSession.setMetadata(new MediaMetadataCompat.Builder()
                .putBitmap(MediaMetadataCompat.METADATA_KEY_ALBUM_ART,
albumArt)
                .putString(MediaMetadataCompat.METADATA_KEY_ARTIST,
activeAudio.getArtist())
                .putString(MediaMetadataCompat.METADATA_KEY_ALBUM,
activeAudio.getAlbum())
                .putString(MediaMetadataCompat.METADATA_KEY_TITLE,
activeAudio.getTitle())
                .build());

        try {
```

```
            Track response =
musixMatch.getMatchingTrack(activeAudio.getTitle(),
activeAudio.getArtist());
            if (response != null) {

                if (response.getTrack() != null &&
response.getTrack().getHasLyrics() == 1) {
                    Lyrics lyric =
musixMatch.getLyrics(response.getTrack().getTrackId());
                    Intent intent = new Intent();
                    intent.setAction(PlayerActivity.Broadcast_LYRICS);
                    intent.putExtra(PlayerActivity.LYRIC,
lyric.getLyricsBody());
                    sendBroadcast(intent);
                }
            }
        } catch (MusixMatchException e) {
            e.printStackTrace();
        }
    }
}


private void buildNotification(PlaybackStatus playbackStatus) {

    int notificationAction = android.R.drawable.ic_media_pause;
    PendingIntent play_pauseAction = null;

    boolean ongoing = false;
    if (playbackStatus == PlaybackStatus.PLAYING) {
        notificationAction = android.R.drawable.ic_media_pause;
        play_pauseAction = playbackAction(1);
        ongoing = false;
    } else if (playbackStatus == PlaybackStatus.PAUSED) {
        notificationAction = android.R.drawable.ic_media_play;
        play_pauseAction = playbackAction(0);
        ongoing = true;
    }

    Bitmap largeIcon = activeAudio.getAlbumArt() == null ?
BitmapFactory.decodeResource(getResources(),
            R.drawable.image5) :
BitmapFactory.decodeFile(activeAudio.getAlbumArt());

    NotificationCompat.Builder notificationBuilder =
(NotificationCompat.Builder) new NotificationCompat.Builder(this)
```

```
            .setShowWhen(false)
            .setOngoing(ongoing)
            .setStyle(new NotificationCompat.MediaStyle()
                    .setMediaSession(mediaSession.getSessionToken())
                    .setShowActionsInCompactView(0, 1, 2))
            .setColor(getResources().getColor(R.color.colorPrimary))
            .setLargeIcon(largeIcon)
            .setSmallIcon(android.R.drawable.stat_sys_headset)
            .setContentText(activeAudio.getArtist())
            .setContentTitle(activeAudio.getAlbum())
            .setContentInfo(activeAudio.getTitle())
            .addAction(android.R.drawable.ic_media_previous, "previous",
playbackAction(3))
            .addAction(notificationAction, "pause", play_pauseAction)
            .addAction(android.R.drawable.ic_media_next, "next",
playbackAction(2));

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

    AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(this
            .getApplicationContext());

    int[] ids = appWidgetManager.getAppWidgetIds(new
ComponentName(this.getPackageName(), PlayerWidget.class.getName()));

    for (int widgetId : ids) {
        RemoteViews remoteViews = new
RemoteViews(this.getApplicationContext().getPackageName(),
R.layout.player_widget);
        remoteViews.setViewVisibility(R.id.btn_play, View.VISIBLE);
        remoteViews.setViewVisibility(R.id.btn_prev, View.VISIBLE);
        remoteViews.setViewVisibility(R.id.btn_next, View.VISIBLE);
        remoteViews.setTextViewText(R.id.title, activeAudio.getTitle());
        remoteViews.setTextViewText(R.id.artist,
activeAudio.getArtist());

        remoteViews.setOnClickPendingIntent(R.id.btn_play,
play_pauseAction);
        remoteViews.setImageViewBitmap(R.id.btn_play,
BitmapFactory.decodeResource(getResources(),
                notificationAction));
        appWidgetManager.updateAppWidget(widgetId, remoteViews);
    }
```

```
}

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

   int notificationAction;
   PendingIntent play_pauseAction;

   notificationAction = android.R.drawable.ic_media_play;
   play_pauseAction = playbackAction(0);

   AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(this
           .getApplicationContext());

   int[] ids = appWidgetManager.getAppWidgetIds(new
ComponentName(this.getPackageName(), PlayerWidget.class.getName()));

   for (int widgetId : ids) {
       RemoteViews remoteViews = new
RemoteViews(this.getApplicationContext().getPackageName(),
R.layout.player_widget);

       Intent intent = new Intent(this.getApplicationContext(),
MainActivity.class);
       intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
       intent.putExtra(MainActivity.INIT_PLAYER, true);
       PendingIntent pendingIntent =
PendingIntent.getActivity(this.getApplicationContext(), 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

       remoteViews.setOnClickPendingIntent(R.id.widget_layout,
pendingIntent);

       remoteViews.setOnClickPendingIntent(R.id.btn_play,
play_pauseAction);
       remoteViews.setImageViewBitmap(R.id.btn_play,
BitmapFactory.decodeResource(getResources(),
               notificationAction));

       remoteViews.setViewVisibility(R.id.btn_play, View.INVISIBLE);
       remoteViews.setViewVisibility(R.id.btn_prev, View.INVISIBLE);
       remoteViews.setViewVisibility(R.id.btn_next, View.INVISIBLE);
       appWidgetManager.updateAppWidget(widgetId, remoteViews);
```

```
    }
}

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

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

Activities

La actividad *PlayerActivity* consta de los siguientes componentes:

- broadcastReceiver: Utilizado para recibir el evento de la obtención de la letra de la canción

```
private BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String lyric = (String) intent.getSerializableExtra(LYRIC);
        LyricDialogFragment fragment =
LyricDialogFragment.newInstance(PlayerActivity.this, lyric);
        fragment.show(getSupportFragmentManager(), "Dialog");
    }
};

//Metodo onCreate
registerReceiver(broadcastReceiver, new
IntentFilter(PlayerActivity.Broadcast_LYRICS));
```

```
//metodo onDestroy
unregisterReceiver(broadcastReceiver);
```

- método loadAudio: utilizado para cargar las canciones en función de los criterios de filtrado recibidos

```
private void loadAudio() {
   ContentResolver contentResolver = getContentResolver();

   Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
   String selection = MediaStore.Audio.Media.IS_MUSIC + "!= 0" +
           ((PlayerActivity.this.selection != null) ? " AND " +
PlayerActivity.this.selection : "");
   String sortOrder = MediaStore.Audio.Media.TITLE + " ASC";
   Cursor cursor = contentResolver.query(uri, null, selection,
PlayerActivity.this.selectionArgs, sortOrder);

   if (cursor != null && cursor.getCount() > 0) {
       audioList = new ArrayList<>();
       while (cursor.moveToNext()) {
           String data =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.DATA));
           String title =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE));
           String album =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM));
           String artist =
cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST));
           Long duration =
cursor.getLong(cursor.getColumnIndex(MediaStore.Audio.Media.DURATION));

           // Save to audioList
           audioList.add(new Song(title, artist, album, data, albumArt,
duration));
       }
   }
   cursor.close();
}
```

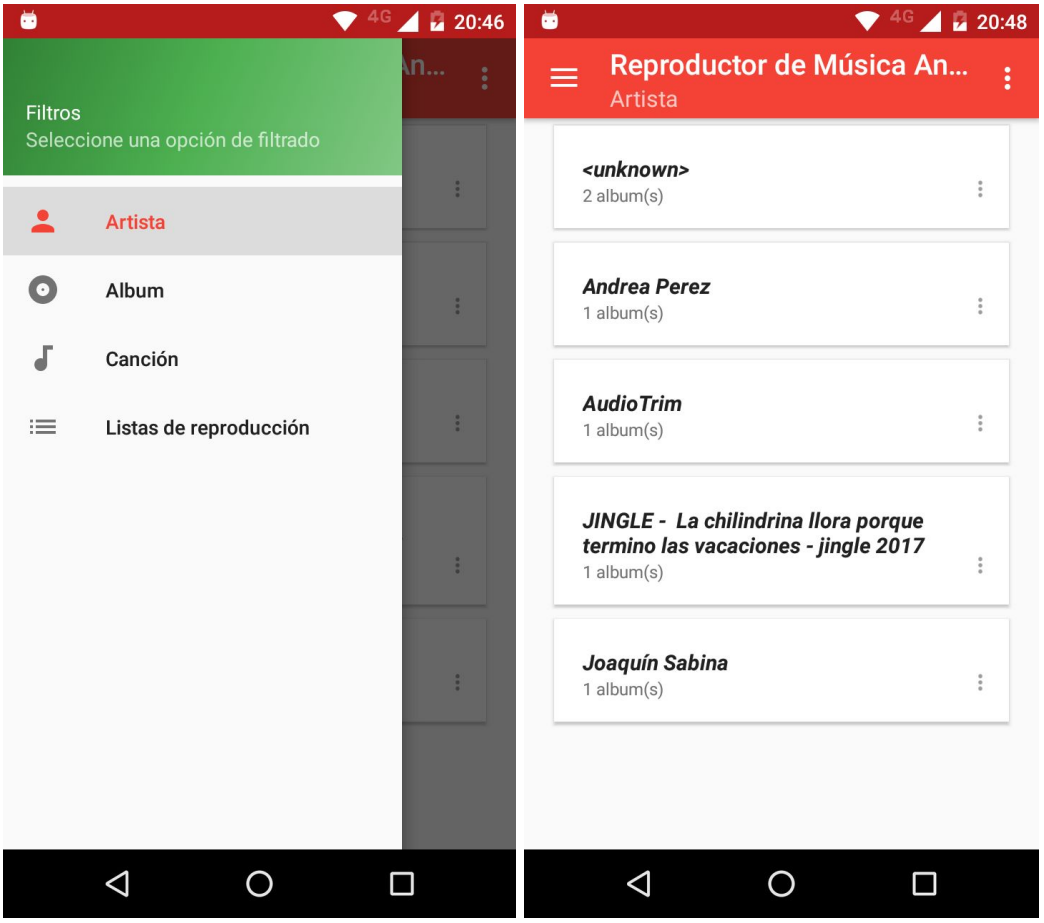
- Carga de audio desde el almacenamiento en el mètodo onCreate: utilizado para cargar la lista de canciones de la última cola reproducida y continuar su ejecución. Esta funcionalidad es activada cuando la aplicación es iniciada desde el widget.

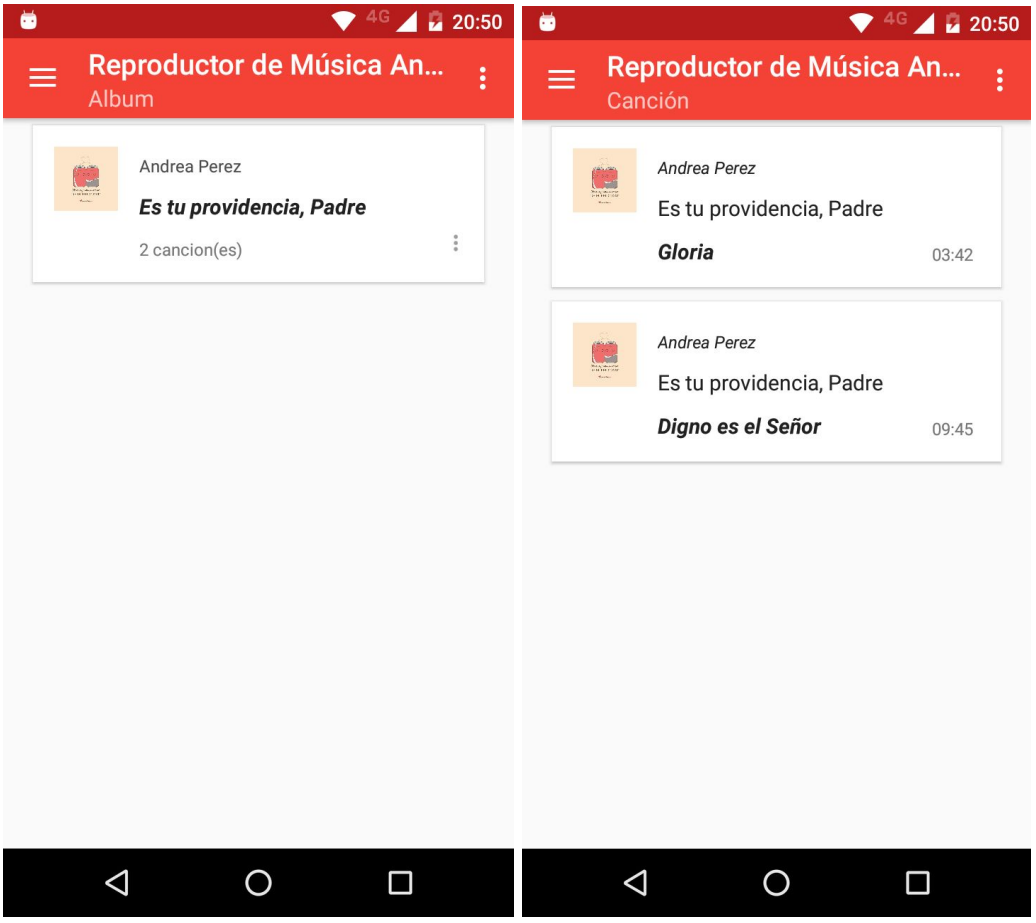
```
//metodo onCreate
if (audioList == null) {
```

```
    if (selection != null)
        loadAudio();
    else {
        StorageUtil st = new StorageUtil(this);
        audioList = st.loadQueue();
        Pair<Integer, Long> positions = st.loadPositions();
        if (positions != null) {
            audioIndex = positions.first;
            audioSeek = positions.second;
        }
    }
}
```

GUI

Como se mencionó anteriormente, la activity *MainActivity* es la contenedora de diversos fragments que permiten filtrar los audios por **artistas, álbumes y canciones**. A continuación se presentan las pantallas correspondientes a la misma:

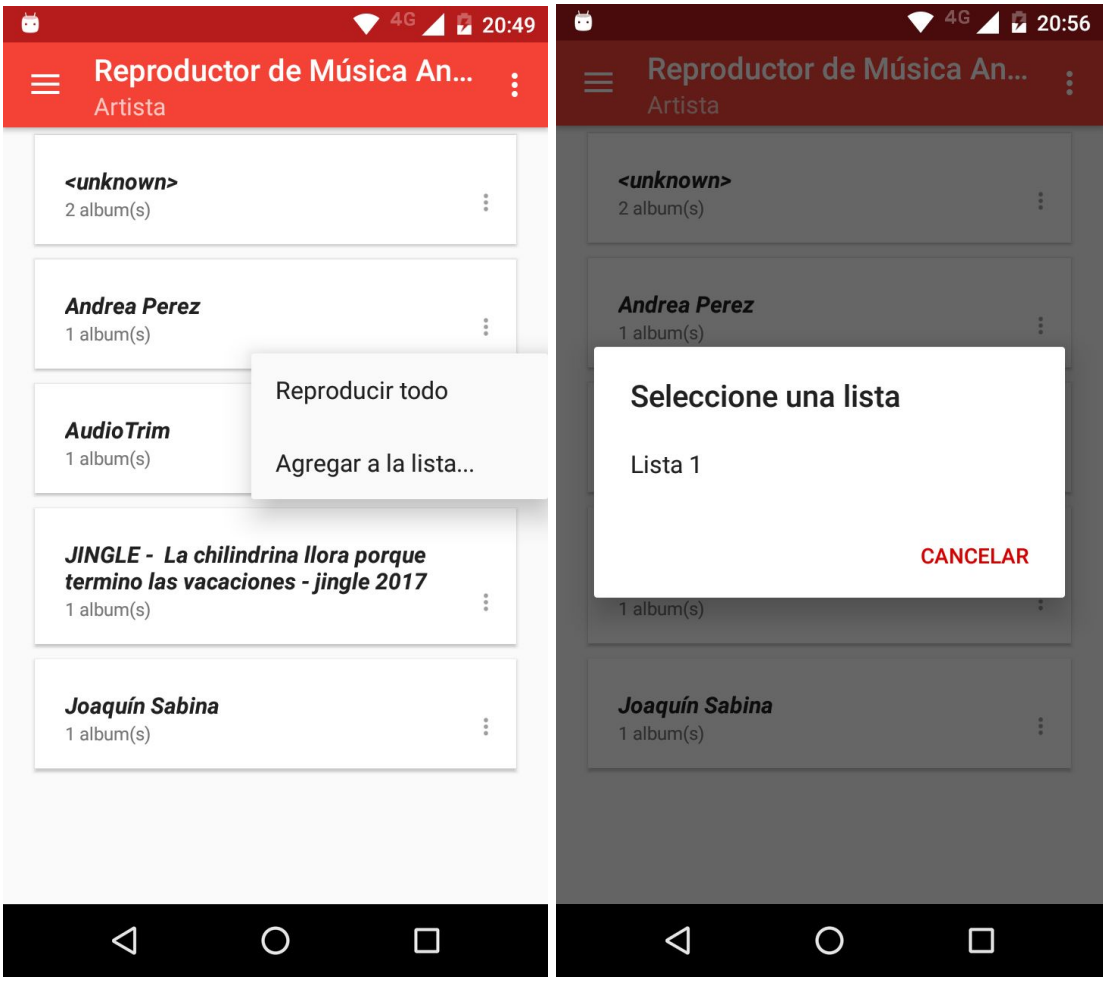




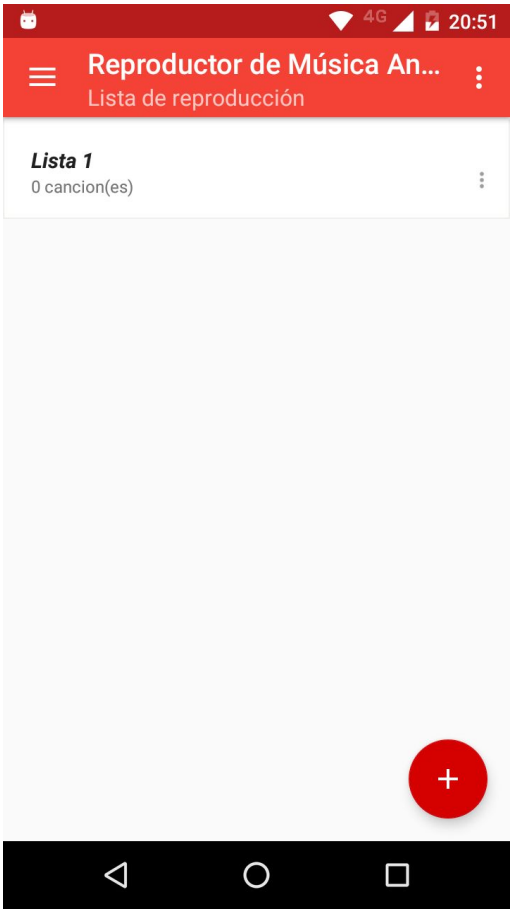
Las listas correspondientes a cada filtro fueron construidas utilizando *RecyclerViews*[8] cuyos ítems se implementaron haciendo uso de *CardViews*[9]. Además del acceso a los filtros a través de la barra lateral, los mismos pueden ser navegados siguiendo el flujo ***Artista*** → ***Álbum*** → ***Canción***. Esto significa que al seleccionar un artista específico, se seleccionan todos los álbumes correspondientes a ese artista. De la misma manera, al elegir un álbum se filtran las canciones de ese álbum.

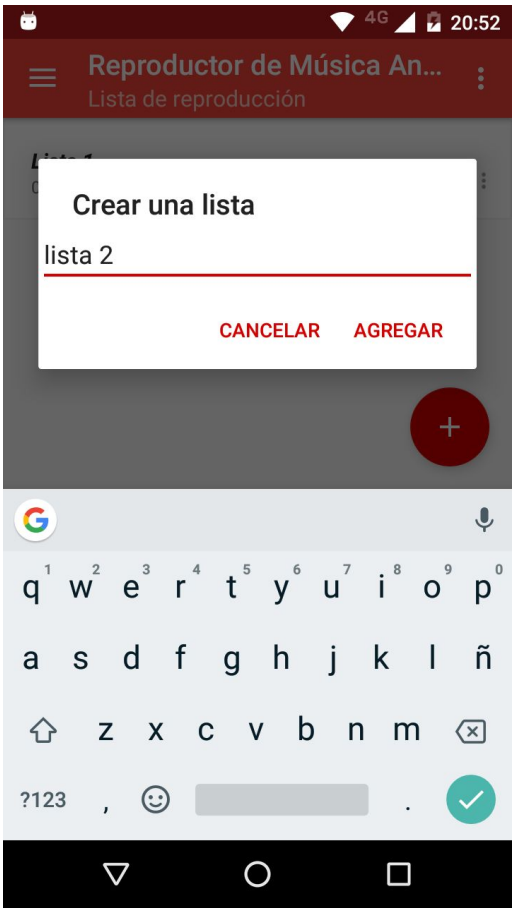
Algunas listas (por ejemplo la de artistas) poseen en sus ítems, un menú contextual que permite reproducir todas sus canciones o añadirlas a una playlist existente; como se muestra a continuación:

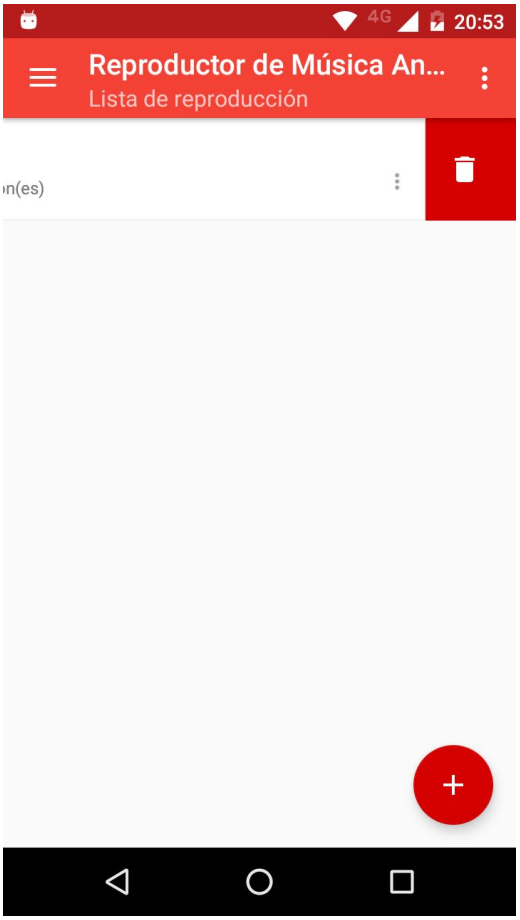
[8] https://developer.android.com/guide/topics/ui/layout/recyclerview.html
[9] https://developer.android.com/training/material/lists-cards.html?hl=es-419

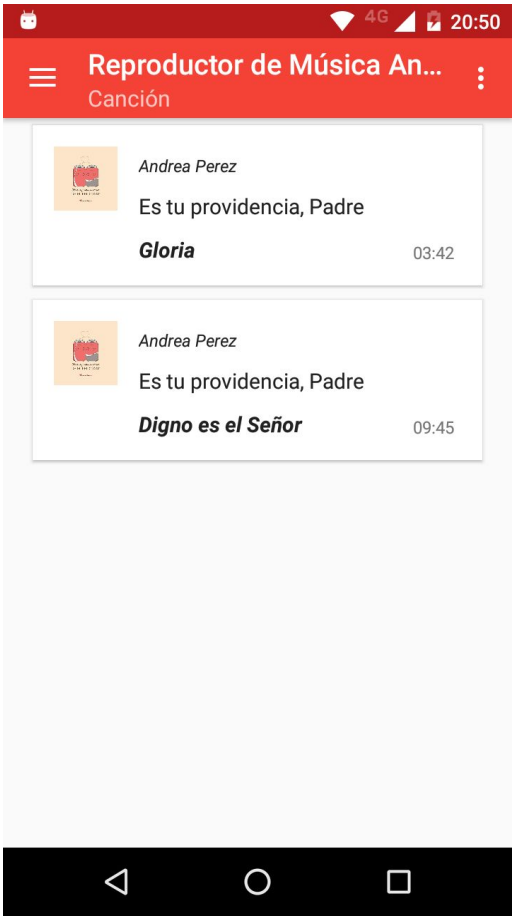


La interfaz visual para administrar *playlists* permite crearlas mediante el botón flotante inferior y eliminarlas con la acción de desplazar hacia la izquierda el ítem de la lista. Además se destaca que muestra las canciones de cada una de ellas.

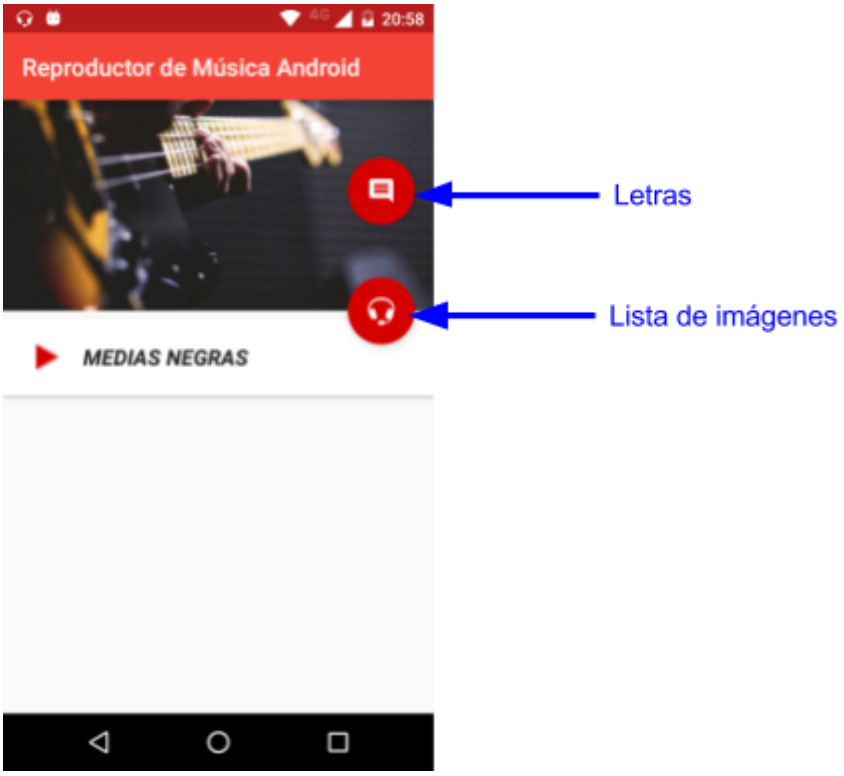

20:51
Reproductor de Música An...
Lista de reproducción
Lista 1
0 cancion(es)

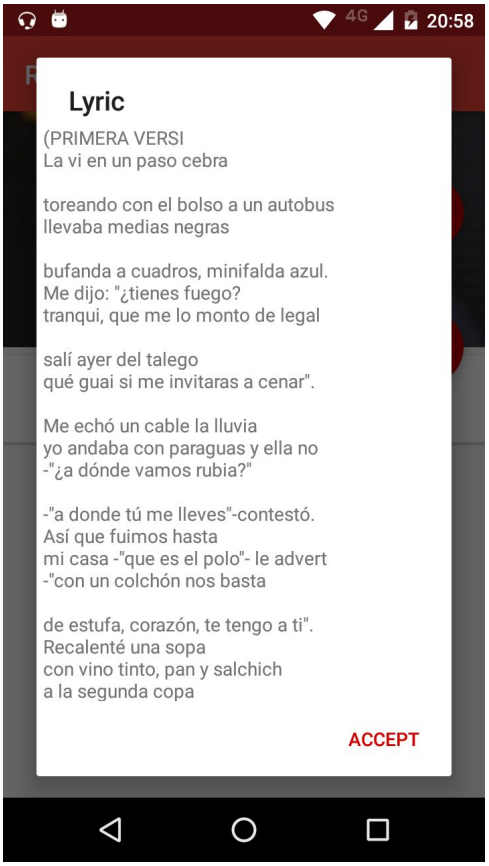

20:52
Reproductor de Música An...
Lista de reproducción
Crear una lista
lista 2
CANCELAR
AGREGAR


20:53
Reproductor de Música An...
Lista de reproducción

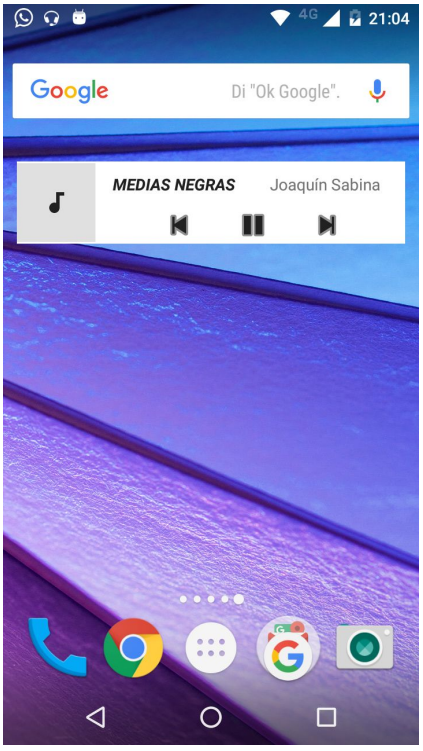

20:50
Reproductor de Música An...
Canción
Andrea Perez
Es tu providencia, Padre
Gloria
03:42
Andrea Perez
Es tu providencia, Padre
Digno es el Señor
09:45

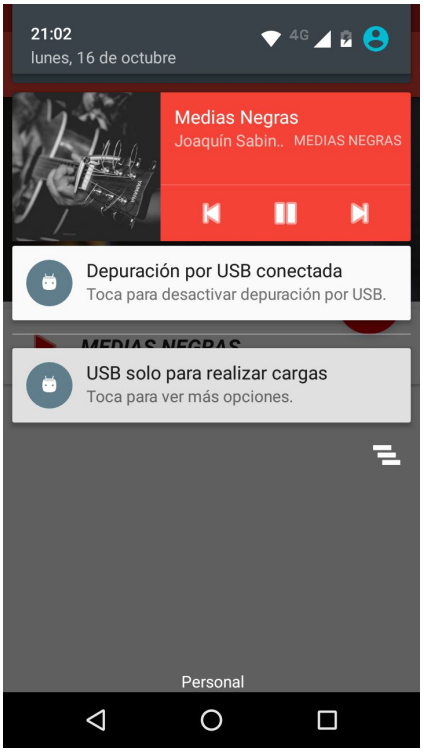
La actividad principal (*PlayerActivity*) permite visualizar la cola de reproducción actual, la carátula del álbum (o una lista de imágenes predefinida) y los botones flotantes para visualizar las letras y lista de imágenes:

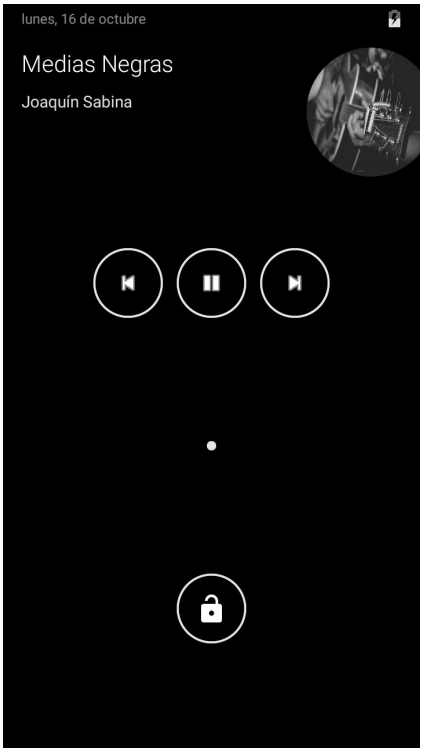




Por último, se presentan las interfaces visuales correspondientes al *Widget*[10] y a la *notification*[11]:







[10] https://developer.android.com/guide/topics/appwidgets/index.html
[11] https://developer.android.com/guide/topics/ui/notifiers/notifications.html

Conclusión

El presente informe describió el desarrollo de un reproductor de música para plataformas móviles android; para ello fue esencial el material didáctico provisto por la cátedra. En este sentido, se destaca que esta propuesta permitió poner en práctica los conceptos relacionados al desarrollo de software para plataformas móviles, es decir, cómo presentar información al usuario mediante diversos componentes gráficos y notificaciones; ejecutar tareas y controlar la aplicación en *background y en foreground* mediante servicios y actividades; y por último cómo interactuar con otros eventos del sistema y también la manera de persistir y recuperar información.

Para el desarrollo se utilizó el SDK provisto por el sitio oficial de android, y el IDE Android Studio, ya que este último incluye herramientas para el diseño de la interfaz, debugger, monitor de logs, etc.

Entre las principales ventajas de este desarrollo se destacan por un lado, que tiene soporte en todos los dispositivos android (dado que se desarrolló utilizando codigo nativo), y por el otro, que respeta los principios de diseño de *Material Design* que son los mismos del SO y no interfiere con el resto de aplicaciones del sistema (no bloquea los eventos de mensajes y llamadas nativos). Una desventaja es que, es solo compatible con android, ya que no se utilizaron librerías cross-platform (por ej.: cordova).