Group Name: Beta

Course Name: Software Engineering

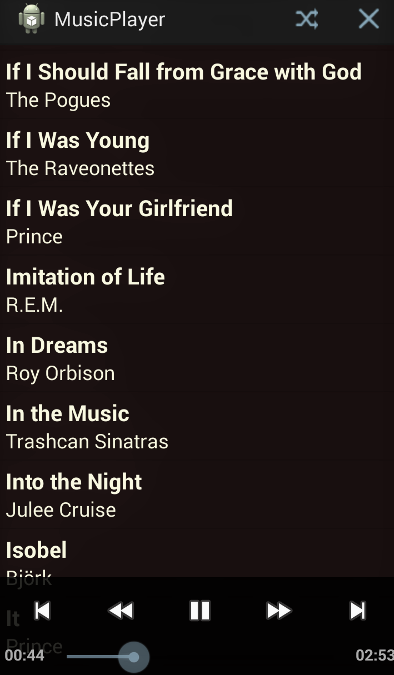
Project Name: Music Player Application

# **What this project is about?**

The music player application for android we’re creating will present a list of songs on the user device. The user can select songs to play from the list. The app will also present controls for interacting with playback and if the user moves away from the app, it will continue playing with a notification displayed while playback elapses.

# **Design: The user interface of the application**

**The basic preview of the final app**

****

We’ll create a new **Activity** class and layout file. We need API level of 16.

We’ll add the following attributes to the activity element to set the screenOrientation and launchMode:

1. <activity
2. android:name="com.example.musicplayer.MainActivity"
3. android:label="@string/app\_name"
4. android:launchMode="singleTop"
5. android:screenOrientation="portrait" >

We will stick to portrait orientation for simplicity. The lauchMode will aid the process of navigating back to the app after moving away from it. We will display a notification indicating the song currently being played, tapping the notification will take the user back to the app. We are also going to use a Service class for music playback.

The layout we’re creating:

1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2. xmlns:tools="http://schemas.android.com/tools"
3. android:layout\_width="fill\_parent"
4. android:layout\_height="fill\_parent"
5. android:orientation="vertical"
6. android:background="#FF330000"
7. tools:context=".MainActivity" >
8. <ListView
9. android:id="@+id/song\_list"
10. android:layout\_width="fill\_parent"
11. android:layout\_height="wrap\_content" >
12. </ListView>
13. </LinearLayout>

The layout includes a ListView in which we will present the list of songs.

We will also use an icon to display in the playback notification.

.

# **Libraries and Algorithm we’re using:**

The app should query the user's device for audio files. So we add a new class named **song** to our project. We will use this class to model the data for a single audio file. Inside the class declaration, add three instance variables for the data we want to store for each track:

1. private long id;
2. private String title;
3. private String artist;

We’ll add a constructor method public Song(long songID, String songTitle, String songArtist)

Then we’ll get methods for the instance variables.

In the main Activity class we’ll add the following imports:

1. import java.util.ArrayList;
2. Import java.util.Collections;
3. import java.util.Comparator;
4. import android.net.Uri;
5. import android.content.ContentResolver;
6. import android.database.Cursor;
7. import android.widget.ListView;

Then we declared the following instance variables:

1. private ArrayList<Song> songList;
2. private ListView songView;

We will store the songs in a list and display them in the ListView instance in the main layout.

In onCreate, after setting the content view, retrieve the ListView instance using the ID we gave it in the main layout:

1. songView = (ListView)findViewById(R.id.song\_list);

Then we instantiate the list as below:

1. songList = new ArrayList<Song>();

Then we create a helper method to retrieve the audio file information. In this method, we’ll create a ContentResolver instance, retrieve the URI for external music files, and create a Cursor instance using the ContentResolver instance to query the music files.

After that we can iterate over the results, first checking that we have valid data:

1. if(musicCursor!=null && musicCursor.moveToFirst()){
2. //get columns
3. int titleColumn = musicCursor.getColumnIndex
4. (android.provider.MediaStore.Audio.Media.TITLE);
5. int idColumn = musicCursor.getColumnIndex
6. (android.provider.MediaStore.Audio.Media.\_ID);
7. int artistColumn = musicCursor.getColumnIndex
8. (android.provider.MediaStore.Audio.Media.ARTIST);
9. //add songs to list
10. do {
11. long thisId = musicCursor.getLong(idColumn);
12. String thisTitle = musicCursor.getString(titleColumn);
13. String thisArtist = musicCursor.getString(artistColumn);
14. songList.add(new Song(thisId, thisTitle, thisArtist));
15. }
16. while (musicCursor.moveToNext());
17. }

We first retrieve the column indexes for the data items that we are interested in for each song, then we use these to create a new Song object and add it to the list, before continuing to loop through the results.

After that we can display the list of songs in the user interface. In the onCreate method, after calling the helper method, we’ll sort the data so that the songs are presented alphabetically.

We use the title variable in the Song class, using the **get** methods we added, to implement a compare method, sorting the songs by title.

Then we define a layout to represent each song in the list. So we add a new file to our project's res/layout folder, naming it song.xml. Each song in the list will be represented by title and artist text strings, so we will use the TextViews to display this data. Notice that the LinearLayout opening tag lists an onClick attribute. We will use this method in the main Activity class to respond to user taps on the songs in the list, playing the song represented by the list item that was tapped.

We will use an Adapter to map the songs to the list view. We create a new class to our app, naming it **SongAdapter** . When creating the class, we gave it the superclass android.widget.BaseAdapter.

We add the following imports:

1. import java.util.ArrayList;
2. import android.content.Context;
3. import android.view.LayoutInflater;
4. import android.widget.LinearLayout;
5. import android.widget.TextView;

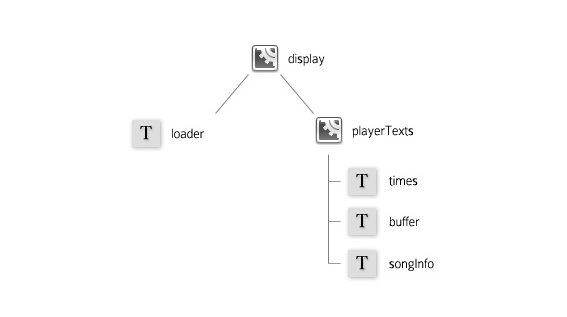
We set the title and artist text by retrieving the correct Song instance from the list using the position index, mapping these strings to the views we added to the song layout file. We also set the position as the view tag, which will let us play the correct song when the user clicks an item in the list. The **song.xml** layout file included an onClick attribute. We will use the method listed there to retrieve the tag in the Activity.

Back in the main Activity class, in the onCreate method after sorting the list, create a new instance of the Adapter class and set it on the ListView :

1. SongAdapter songAdt = new SongAdapter(this, songList);
2. songView.setAdapter(songAdt);

When a user run the app, it should present the list of songs on the device, clicking them will cause the app to throw an exception.

**The basic diagram of our app is given below:**



**Conclusion**

We are hopeful that we’ll complete our project very soon. We are working on it every day. We've now set the app up to read songs from the user device. Next, we will begin playback when the user selects a song using the MediaPlayer class. We will implement playback using a Service class so that it will continue as the user interacts with other apps. Finally, we will use a MediaController class to give the user control over playback.