

Chapter1

INTRODUCTION

In order to solve the problem of complex functions and large required memory of mobile phone music player on the current market, a new music player of simple, convenient, less required memory as well as user-friendly is developed. Based on the Android technology, using the Java language and Eclipse programming tools lead to design and coding of music player. The new design mainly realizes six core functions including main play interface, playlists, menus, play settings, file browsing and song search. This player has merits of high performance, simple operation, and run independently on the Android mobile devices. At the same time, the player can also browse and access files in mobile phones

1.1Motivation:

There were many interesting design projects to choose from. My team felt most passionate about notes app. Many students spend hours summarizing large texts. especially in behavioral and medicinal studies. Here at twenty, one needs to learn a lot of content from books consisting of hundreds of pages. summarizing these can become tedious and often involves a lot of copy-pasting from one document to another. an act which does not improve the study process and makes the important process of summarizing less interesting and more time consuming for students. in this project, we wanted to make an application which improves this process. our goal was to support students to study and create notes more effectively. this motivated us to work hard as we had the ability to greatly impact the way students create notes.

1.2Proposed System:

This system will provide an easy approach to play music or create playlists. It will be easy for the users to use this app. The system can used by anyone which is easy to operate. The Objective of Music App is to provide better facility to the users to Music is easily stored and managed on the MP3 player, including skipping tracks or, on some models, arranging playlists.

1.3 Related Work:

The mobile applications that are available in the market are very useful to the smart phone users and make their life easy. The music player is also one of those applications, which much scope in daily life. As there are many similar applications available today we added some innovative features to make our application unique, easy to use and efficient.

Chapter2

REQUIREMENT ANALYSIS AND SYSTEM SPECIFICATION

2.1 Software Requirements

- Android Operating System on the Smartphone.
- Android studio platform.
- The device should have internet connectivity.

2.2 Hardware Requirements

- 8GB RAM
- 500GB HardDisk
- I5Processor

Chapter 3

SYSTEM DESIGN

3.1 Detailed design

Data flow diagram:

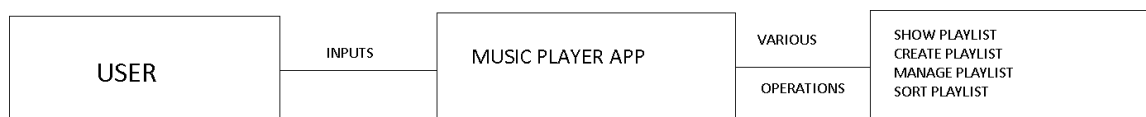






Figure 3.1 Data Flow diagram of Music Player App

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

3.2 Application design

This application like most of the applications will have user login screen and option for registration. The user must register in this application when he/she is using for first time. However, the user who is already registered can login to the application using his/her login credentials that are created by the user at the time of registration.

3.2.1 Introduction to the Layout Edit

1. Palette: Contains various views and view groups that you can drag into your layout.
2. Component Tree: Shows the hierarchy of components in your layout.
3. Toolbar: Click these buttons to configure your layout appearance in the editor and change layout attributes.
4. Design editor: Edit your layout in Design view, Blueprint view, or both.
5. Attributes: Controls for the selected view's attributes.
6. View mode: View your layout in either Code , Design , or Split  modes. Split mode shows both the Code and Design windows at the same time.
7. Zoom and pan controls: Control the preview size and position within the editor.
8. When you open an XML layout file, the design editor opens by default, as shown in figure 1. To edit the layout XML in the text editor, click the Code  button in the top-right corner of the window. Note that the Palette, Component Tree, and Attributes windows are not available

while editing your layout in Code view.

3.2.2 Change the Preview Appearance

The buttons in the top row of the design editor enable you to configure the appearance of your layout in the editor. Corresponding to the numbers in figure 2, the buttons available are as follows.



Fig 3.22 Buttons in the Layout Editor toolbar that configure the layout appearance

1.Design and blueprint: Select how you'd like to view your layout in the editor. Choose Design to see a rendered preview of your layout. Choose Blueprint to see only outlines for each view. Choose Design + Blueprint to see both views side-by-side. You can also press B to cycle through these view types.

2.Screen orientation and layout variants: Choose between landscape and portrait screen orientation, or choose other screen modes for which your app provides alternative layouts, such as night mode. This menu also contains commands for creating a new layout variant. You can also press O to change orientation

3.Device type and size: Select the device type (phone/tablet, Android TV, or Wear OS) and screen configuration (size and density). You can select from several pre-configured device types and your own AVD definitions, or you can create a new AVD by selecting Add Device Definition from the list. You can resize the device size by dragging the bottom-right corner of the layout. You can also press D to cycle through the device list.

4.API version: Select the version of Android on which to preview your layout.

5.App theme: Select which UI theme to apply to the preview. Note that this works only for supported layout styles, so many themes in this list result in an error.

6.Language: Select the language to show for your UI strings. This list displays only the languages available in your string resources. If you'd like to edit your translations, click Edit Translations from the drop-down menu. For more information on working with translations, see

Localize the UI with Translations Editor. Device type and size: Select the device type (phone/tablet, Android TV, or Wear OS) and screen configuration (size and density). You select from several pre-configured device types and your own AVD definitions, or you can create a new AVD by selecting .

3.2.3 Create a new Layout

When adding a new layout for your app, first create a default layout file in your project's default layout/ directory so that it applies to all device configurations. Once you have a default layout, you can create layout variations for specific device configurations, such as for large screens. This work like dairy & save your & your bill record which has been settled or in pending to. We can mark the status of our bill. This is suitable for middle class people who always don't use to settlements and get things on credits

You can create a new layout in one of the following ways:

Use Android Studio's main menu

1. In the Project window, click the module in which you want to add a layout.
2. In the main menu, select File > New > XML > Layout XML File.
3. In the dialog that appears, provide the file name, the root layout tag, and the source set in which the layout belongs.
4. Click Finish to create the layout.

Use Android Studio's main menu

1. In the Project window, click the module in which you want to add a layout.
2. In the main menu, select File > New > XML > Layout XML File.
3. In the dialog that appears, provide the file name, the root layout tag, and the source set in which the layout belongs.
4. Click Finish to create the layout

Use the Projected View

1. Choose the Project view from within the Project window.
2. Right-click the layout directory where you'd like to add the layout.
3. In the context menu that appears, click New > Layout Resource File.

Use the Android view

1. Choose the Android view from within the Project window.
2. Right-click the layout folder.
3. In the context menu that appears, select New > Layout Resource File.

3.3 Use Case diagram:

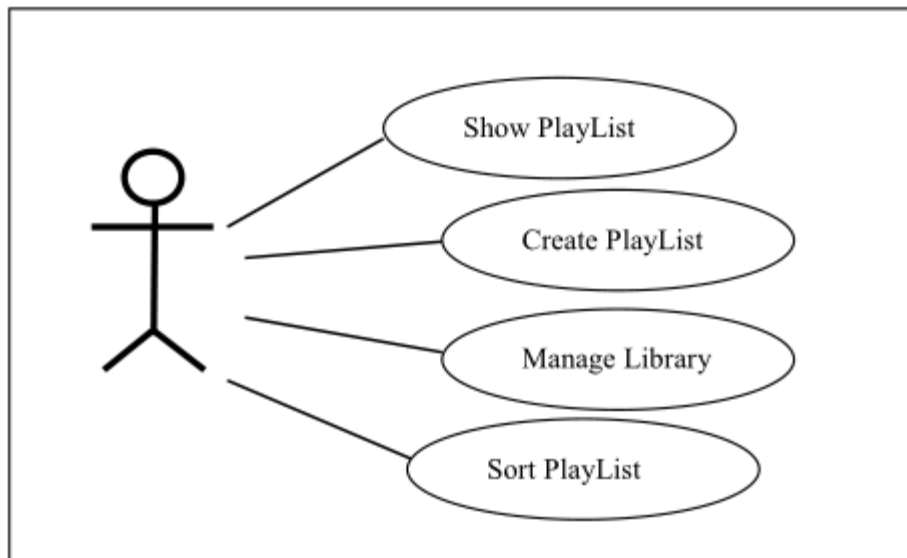


Figure3.3 Use Case diagram of Music Player app

Fig.3.3 Use Case Diagram

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actors use it, but not how the system operates internally.

3.4 Entity Relationship diagram:

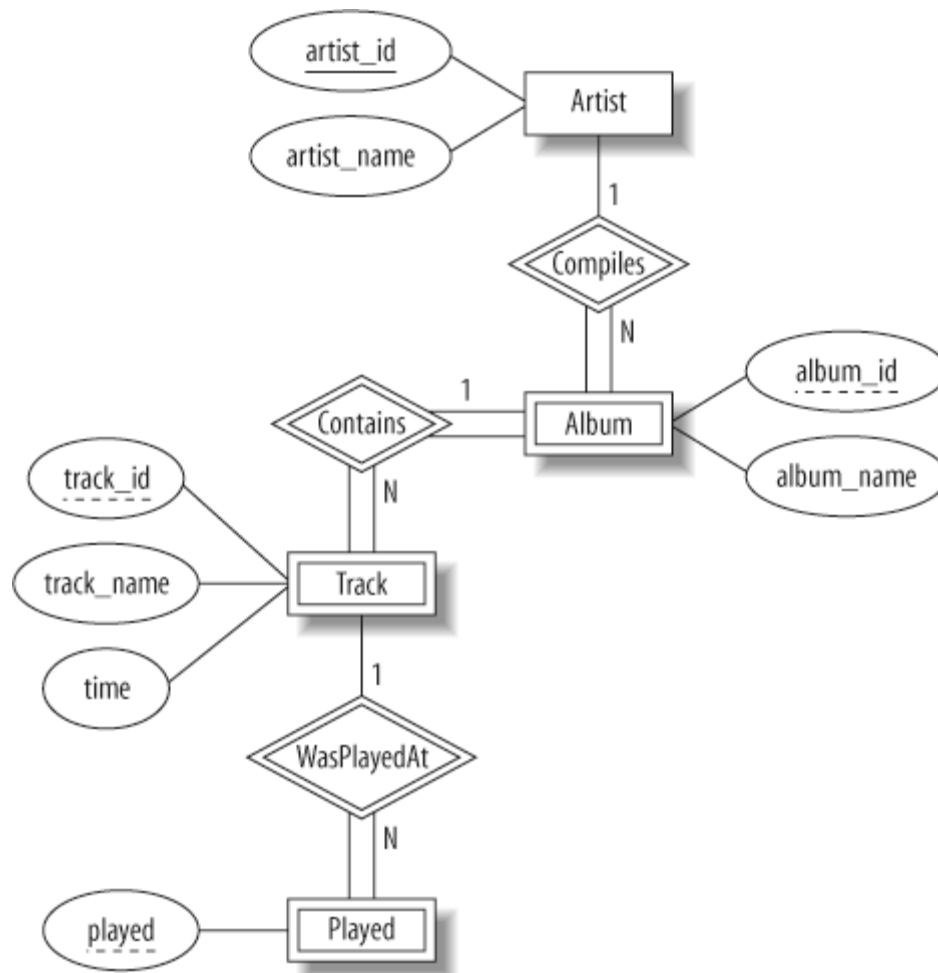


Figure3.4 ER diagram of Music Player App

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

Chapter 4

Implementation And Testing

4.1 Introduction to Programming languages ,IDES,Tools and technologies

4.1.1Java programming languages

Java is a general-purpose, class-based, object-oriented programming language designed for having lesser implementation dependencies. It is a computing platform for application development. Java is fast, secure, and reliable, therefore. It is widely used for developing Java applications in laptops, data centres, game consoles, scientific supercomputers, cell phones, etc. Java Platform is a collection of programs that help programmers to develop and run Java programming applications efficiently. It includes an execution engine, a compiler, and a set of libraries in it. It is a set of computer software and specifications. James Gosling developed the Java platform at Sun Microsystems, and the Oracle Corporation later acquired it

Here are some important Java applications:

- It is used for developing Android Apps
- Helps you to create Enterprise Software
- Wide range of Mobile java Applications
- Scientific Computing Applications

History of java programming language

Here are the important landmarks of java programming language

- The Java language was initially called OAK.
- Originally, it was developed for handling portable devices and set-top boxes. Oak was a massive failure.
- In 1995, Sun changed the name to "Java" and modified the language to take advantage of the burgeoning www (World Wide Web) development business.
- Later,in2009,OracleCorporationacquiredSunMicrosystemsandtookownershipofthreekeySuns software assets: Java, MySQL, and Solaris.

4.1.2 Java Development Kit (JDK)

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and

JDK helps them to code and run Java programs. It is possible to install more than one JDK version on . the same computer.

Here are the main reasons for using JDK:

JDK contains tools required to write Java programs and JRE to execute them. It includes a compiler, Java application launcher, Applet viewer, etc. Compiler converts code written in Java into byte code. Java application launcher opens a JRE, loads the necessary class, and executes its main method.

4.1.3 Android System

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch performance, usability, and other problems
- Pro Guard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components
- A rich layout editor that allows users to drag-and-drop UI components.
- Support for building Android Wear apps
- Built-in support for Google Cloud Platform, enabling integration with Firebase
- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

Project Structure

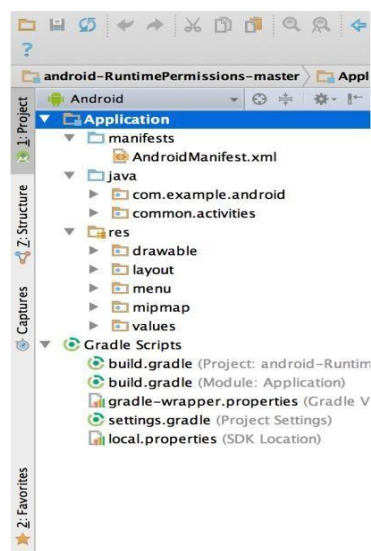


Fig 4.1.3.1 The Project Files in Android View

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

Android app modules

- Library modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- manifests: Contains the AndroidManifest.xml file.
- java: Contains the Java source code files, including JUnit test code.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project dropdown (in figure 1, it's showing as Android).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the Problems view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

The User Interface

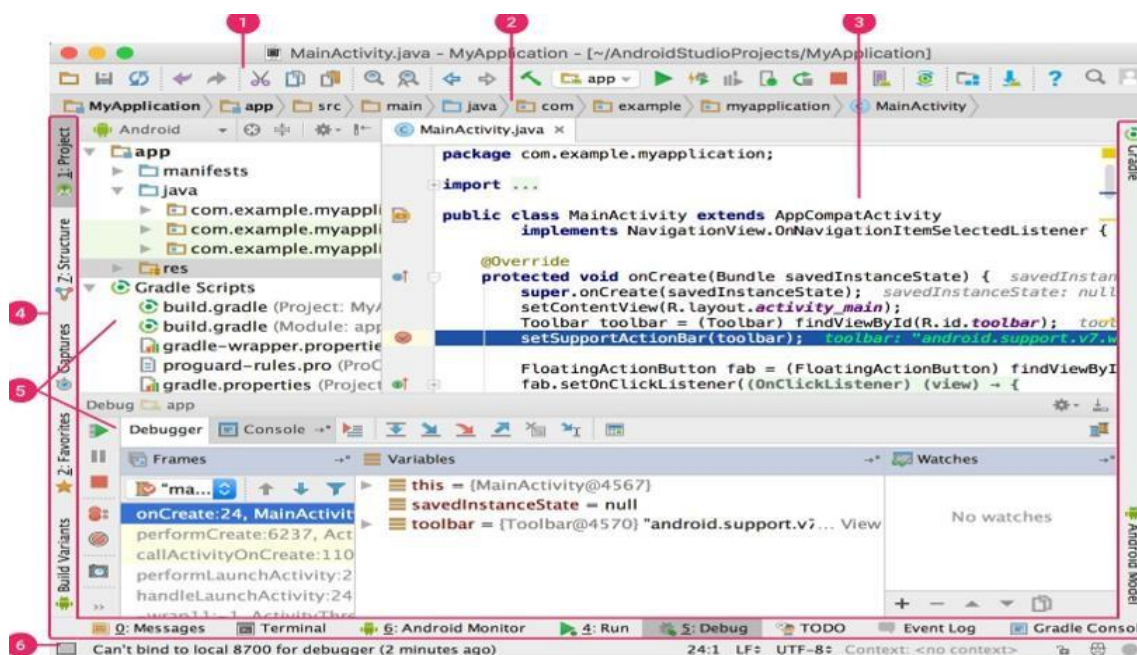


Figure4.1.3.2 The Android Studio main window

The Android Studio main window is made up of several logical areas identified in figure:

- 1.The toolbar lets you carry out a wide range of actions, including running your app and launching Android tools.
- 2.The navigation bar helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the Project window.

3.The editor window is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.

4.The tool window bar runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.

5.The tool windows give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.

6.The status bar displays the status of your project and the IDE itself, as well as any warnings or messages.

7.The toolbar lets you carry out a wide range of actions, including running your app and launching Android tools.

8.The navigation bar helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the Project window.

9.The editor window is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.

10.The tool window bar runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.

11.The tool windows give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.

12.The status bar displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

Gradle Build System

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

Create multiple APKs for your app, with different features using the same project and modules.
Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core

source files. Android Studio build files are named build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

Debug and profile tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline Debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values



Figure 4.1.3.3 An inline variable value

Performance Profiles:

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deal located objects, locate memory leaks, optimize graphics performance, and analyze network requests.

4.2 Test Plan and Test Activities

Android Studio is designed to make testing simple. With just a few clicks, you can set up a JUnit test that runs on the local JVM or an instrumented test that runs on a device. Of course, you can also extend your test

capabilities by integrating test frameworks such as Mockito to test Android API calls in your local unit tests, and Espresso or UI Automator to exercise user interaction in your instrumented tests. You can generate Espresso tests automatically using Espresso Test Recorder.

Test types and location

The location of your test code depends on the type of test you are writing. Android Studio provides source code directories (source sets), for the following two types of tests:

Local unit tests

Located at module-name/src/test/java/.

These are tests that run on your machine's local Java Virtual Machine (JVM). Use these tests to minimize execution time when your tests have no Android framework dependencies or when you can mock the Android framework dependencies.

Instrumented tests

Located at module-name/src/androidTest/java/.

These are tests that run on a hardware device or emulator. These tests have access to Instrumentation APIs, give you access to information such as the Context of the app you are testing, and let you control the app under test from your test code. Use these tests when writing integration and functional UI tests to automate user interaction, or when your tests have Android dependencies that mock objects cannot satisfy.

Because instrumented tests are built into an APK (separate from your app APK), they must have their own AndroidManifest.xml file. However, Gradle automatically generates this file during the build so it is not visible in your project source set. You can add your own manifest file if necessary, such as to specify a different value

Instrumented tests Local unit tests

Located at module-name/src/test/java/.

These are tests that run on your machine's local Java Virtual Machine (JVM). Use these tests to minimize execution time when your tests have no Android framework dependencies or when you can mock the Android framework dependencies.

At runtime, these tests are executed against a modified version of android.jar where all final modifiers have been stripped off. This lets you use popular mocking libraries, like Mockito.

a different value for `minSdkVersion` or register run listeners just for your tests. When building yourapp, Gradle merges multiple manifest files into one manifest. Located at module-name/src/androidTest/java/.

```
public void setUp() throws Exception {
    getTargetContext().deleteDatabase(ExpenseDatabaseHelper.EXPENSE_DB);
    database = new ExpenseDatabaseHelper(getTargetContext());
    database.truncate(ExpenseTypeTable.TABLE_NAME);
    database.truncate(ExpenseTable.TABLE_NAME);
    freezeDate("2015-10-02");
}
```

The Gradle build interprets these test source sets in the same manner as it does for your project's app source sets, which allows you to create tests based on build variants.

When you create a new project or add an app module, Android Studio creates the test source sets listed above and includes an example test file in each. You can see them in the Project window as shown in figure

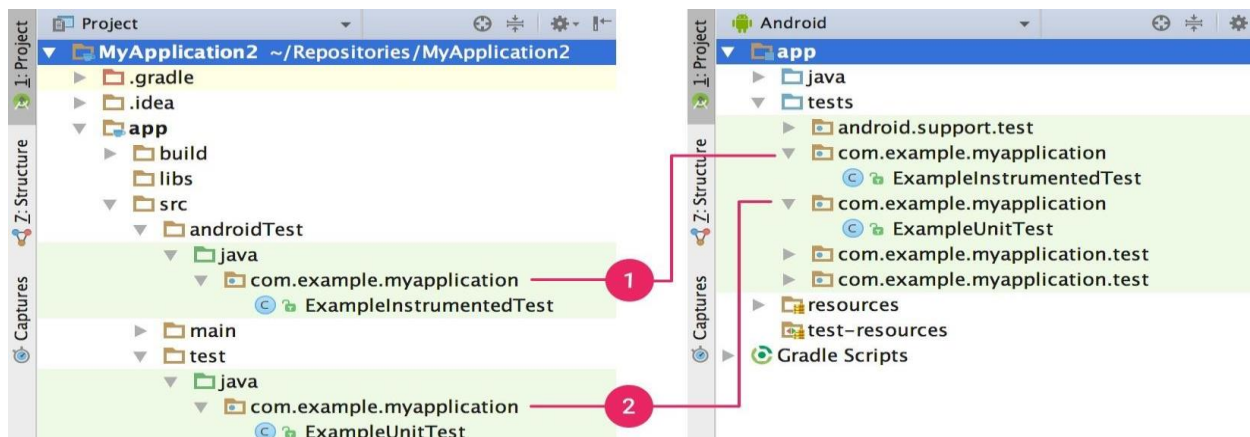


Figure 4.2.1. Your project's (1) instrumented tests and (2) local JVM tests are visible in either the Project view (left) or Android view (right)

Add a new test


To create either a local unit test or an instrumented test, you can create a new test for a specific class or method by following these steps:

1. Open the Java file containing the code you want to test.
2. Click the class or method you want to test, then press **Ctrl+Shift+T** (**⌘+T**).
3. In the menu that appears, click **Create New Test**.
4. In the **Create Test** dialog, edit any fields and select any methods to generate, and then click **OK**.

Alternatively, you can create a generic Java file in the appropriate test source set as follows:

1. In the Project window on the left, click the drop-down menu and select the **Project** view.
2. Expand the appropriate module folder and the nested **src** folder. To add a local unit test, expand the **test** folder and the nested **java** folder; to add an instrumented test, expand the **androidTest** folder and the nested **java** folder.
3. Right-click on the Java package directory and select **New > Java Class**.
4. Name the file and then click **OK**. Run a test

To run a test, proceed as follows:

- Be sure your project is synchronized with Gradle by clicking **Sync Project** in the toolbar.
- Run your test in one of the following ways:
 - In the Project window, right-click a test and click **Run**.
 - To run all tests, right-click on the test directory and click **Run tests**.
- In the Code Editor, right-click a class or method in the test file and click **Run**  to test all methods in the class.

By default, your test runs using Android Studio's default run configuration. If you'd like to change some run settings such as the instrumentation runner and deployment options, you can edit the run configuration in the

XML Code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.maxfour.music">

    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.BROADCAST_STICKY" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />

    <application
        android:name=".App"
        android:allowBackup="true"
        android:icon="@mipmap/ic_music"
        android:label="@string/app_name"
        android:requestLegacyExternalStorage="true"
        android:resizeableActivity="true"
        android:supportsRtl="true"
        android:theme="@style/Theme.Music.Light"
        tools:ignore="UnusedAttribute">
        <activity
            android:name=".ui.activities.MainActivity"
            android:label="@string/main_activity_name"
            android:theme="@style/SplashTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <action android:name="android.intent.action.MUSIC_PLAYER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```

    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.category.APP_MUSIC" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
<intent-filter>
    <action android:name="android.media.action.MEDIA_PLAY_FROM_SEARCH" />

    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />

    <category android:name="android.intent.category.DEFAULT" />

    <data android:scheme="content" />
    <data android:mimeType="audio/*" />
    <data android:mimeType="application/ogg" />
    <data android:mimeType="application/x-ogg" />
    <data android:mimeType="application/itunes" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />

    <category android:name="android.intent.category.DEFAULT" />

    <data android:scheme="file" />
    <data android:mimeType="audio/*" />
    <data android:mimeType="application/ogg" />
    <data android:mimeType="application/x-ogg" />
    <data android:mimeType="application/itunes" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />

    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />

```



```

<data android:scheme="http" />
<data android:mimeType="audio/*" />
<data android:mimeType="application/ogg" />
<data android:mimeType="application/x-ogg" />
<data android:mimeType="application/itunes" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.VIEW" />

    <category android:name="android.intent.category.DEFAULT" />

    <data android:mimeType="vnd.android.cursor.dir/playlist" />
    <data android:mimeType="vnd.android.cursor.dir/albums" />
    <data android:mimeType="vnd.android.cursor.dir/artists" />
</intent-filter>
<intent-filter>
    <action android:name="com.cyanogenmod.eleven.AUDIO_PLAYER" />

    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
<intent-filter>
    <action android:name="android.intent.action.PICK" />

    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.OPENABLE" />

    <data android:mimeType="vnd.android.cursor.dir/audio" />
</intent-filter>
</activity>
<activity android:name=".ui.activities.AlbumDetailActivity" />
<activity android:name=".ui.activities.ArtistDetailActivity" />
<activity android:name=".ui.activities.GenreDetailActivity" />
<activity android:name=".ui.activities.PlaylistDetailActivity" />
<activity
    android:name=".ui.activities.tageditor.SongTagEditorActivity"
    android:windowSoftInputMode="adjustResize" />
<activity

```

```

        android:name=".ui.activities.tageditor.AlbumTagEditorActivity"
        android:windowSoftInputMode="adjustResize" />
<activity android:name=".ui.activities.SearchActivity" />
<activity
    android:name=".ui.activities.SettingsActivity"
    android:label="@string/action_settings" />
<activity
    android:name=".ui.activities.AboutActivity"
    android:label="@string/action_about"
    android:theme="@style/AboutActivityTheme" />
<activity
    android:name=".ui.activities.intro.AppIntroActivity"
    android:label="@string/intro_label"
    android:theme="@style/Theme.Intro" />
<activity
    android:name=".appshortcuts.AppShortcutLauncherActivity"
    android:launchMode="singleInstance"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />

<service
    android:name=".service.MusicService"
    android:enabled="true" />

<receiver android:name=".service.MediaButtonIntentReceiver">
    <intent-filter>
        <action android:name="android.intent.action.MEDIA_BUTTON" />
    </intent-filter>
</receiver>

<meta-data
    android:name="android.max_aspect"
    android:value="2.1" />
<meta-data
    android:name="com.lge.support.SPLIT_WINDOW"
    android:value="true" />
<meta-data
    android:name="com.maxfour.music.glide.MusicGlideModule"

```

```

        android:value="GlideModule" />
<meta-data
    android:name="com.bumptech.glide.integration.okhttp3.OkHttpGlideModule"
    android:value="GlideModule" />

<!-- Widgets -->
<receiver android:name=".appwidgets.BootReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <action android:name="android.intent.action.QUICKBOOT_POWERON" />
    </intent-filter>
</receiver>
<receiver
    android:name=".appwidgets.AppWidgetBig"
    android:exported="false"
    android:label="@string/app_widget_big_name">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/app_widget_big_info" />
</receiver>
<receiver
    android:name=".appwidgets.AppWidgetClassic"
    android:exported="false"
    android:label="@string/app_widget_classic_name">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/app_widget_classic_info" />
</receiver>
<receiver

```

```

        android:name=".appwidgets.AppWidgetClassicDark"
        android:exported="false"
        android:label="@string/app_widget_classic_dark_name">
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/app_widget_classic_dark_info" />
    </receiver>
    <receiver
        android:name=".appwidgets.AppWidgetSmall"
        android:exported="false"
        android:label="@string/app_widget_small_name">
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/app_widget_small_info" />
    </receiver>
    <receiver
        android:name=".appwidgets.AppWidgetSmallDark"
        android:exported="false"
        android:label="@string/app_widget_small_dark_name">
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/app_widget_small_dark_info" />
    </receiver>
    <receiver
        android:name=".appwidgets.AppWidgetCard"

```

```

        android:exported="false"
        android:label="@string/app_widget_card_name">
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/app_widget_card_info" />
    </receiver>
    <receiver
        android:name=".appwidgets.AppWidgetCardDark"
        android:exported="false"
        android:label="@string/app_widget_card_dark_name">
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>

        <meta-data
            android:name="android.appwidget.provider"
            android:resource="@xml/app_widget_card_dark_info" />
    </receiver>

    <provider
        android:name="androidx.core.content.FileProvider"
        android:authorities="${applicationId}"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/provider_paths" />
    </provider>
</application>

</manifest>

```

Java Code:

```

package com.maxfour.music;

import android.util.SparseArray;
import android.util.SparseIntArray;
import android.view.View;
import androidx.databinding.DataBinderMapper;
import androidx.databinding.DataBindingComponent;
import androidx.databinding.ViewDataBinding;
import java.lang.Integer;
import java.lang.Object;
import java.lang.Override;
import java.lang.RuntimeException;
import java.lang.String;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class DataBinderMapperImpl extends DataBinderMapper {
    private static final SparseIntArray INTERNAL_LAYOUT_ID_LOOKUP = new SparseIntArray(0);

    @Override
    public ViewDataBinding getDataBinder(DataBindingComponent component, View view, int layoutId) {
        int localizedLayoutId = INTERNAL_LAYOUT_ID_LOOKUP.get(layoutId);
        if(localizedLayoutId > 0) {
            final Object tag = view.getTag();
            if(tag == null) {
                throw new RuntimeException("view must have a tag");
            }
        }
        return null;
    }

    @Override
    public ViewDataBinding getDataBinder(DataBindingComponent component, View[] views, int
layoutId) {
        if(views == null || views.length == 0) {
            return null;
        }
        int localizedLayoutId = INTERNAL_LAYOUT_ID_LOOKUP.get(layoutId);
        if(localizedLayoutId > 0) {

```

```

final Object tag = views[0].getTag();
if(tag == null) {
    throw new RuntimeException("view must have a tag");
}
switch(localizedLayoutId) {
}
}
return null;
}

```

```

@Override
public int getLayoutId(String tag) {
    if (tag == null) {
        return 0;
    }
    Integer tmpVal = InnerLayoutIdLookup.sKeys.get(tag);
    return tmpVal == null ? 0 : tmpVal;
}

```

```

@Override
public String convertBrIdToString(int localId) {
    String tmpVal = InnerBrLookup.sKeys.get(localId);
    return tmpVal;
}

```

```

@Override
public List<DataBinderMapper> collectDependencies() {
    ArrayList<DataBinderMapper> result = new ArrayList<DataBinderMapper>(1);
    result.add(new androidx.databinding.library.baseAdapters.DataBinderMapperImpl());
    return result;
}

```

```

private static class InnerBrLookup {
    static final SparseArray<String> sKeys = new SparseArray<String>(1);

    static {
        sKeys.put(0, "_all");
    }
}

```

```
private static class InnerLayoutIdLookup {  
    static final HashMap<String, Integer> sKeys = new HashMap<String, Integer>(0);  
}  
}
```


Chapter 5

RESULTS AND DISCUSSIONS

5.1Screenshots:

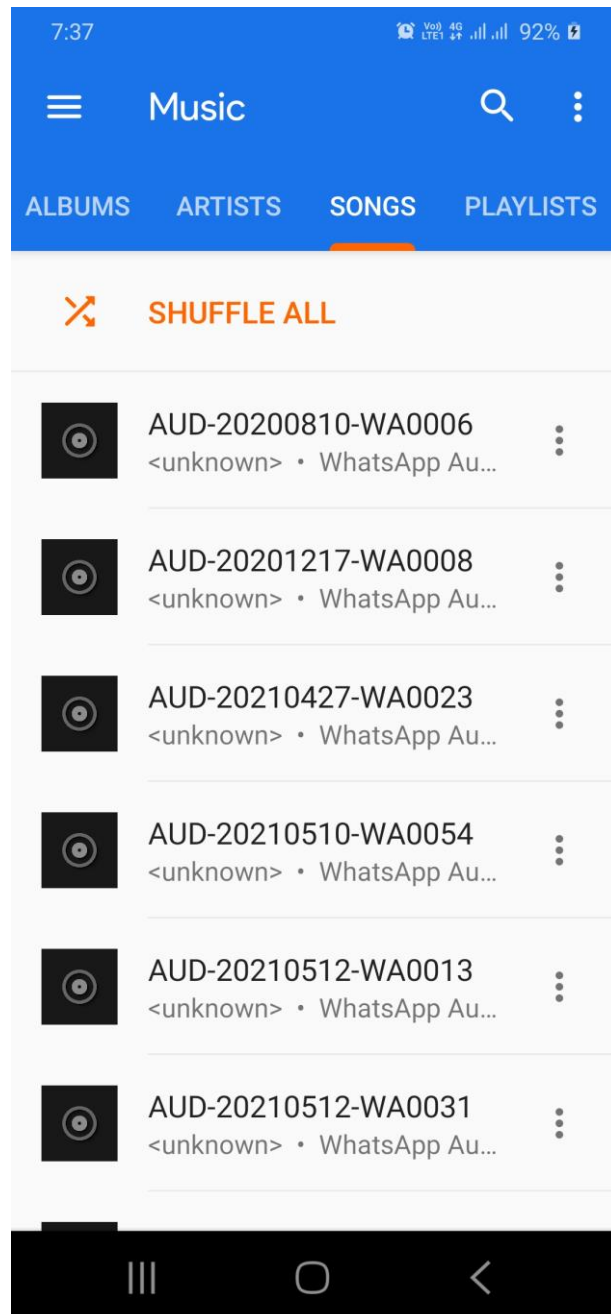


Figure 5.1.1This displays ‘THE SONGS PAGE OF APP’

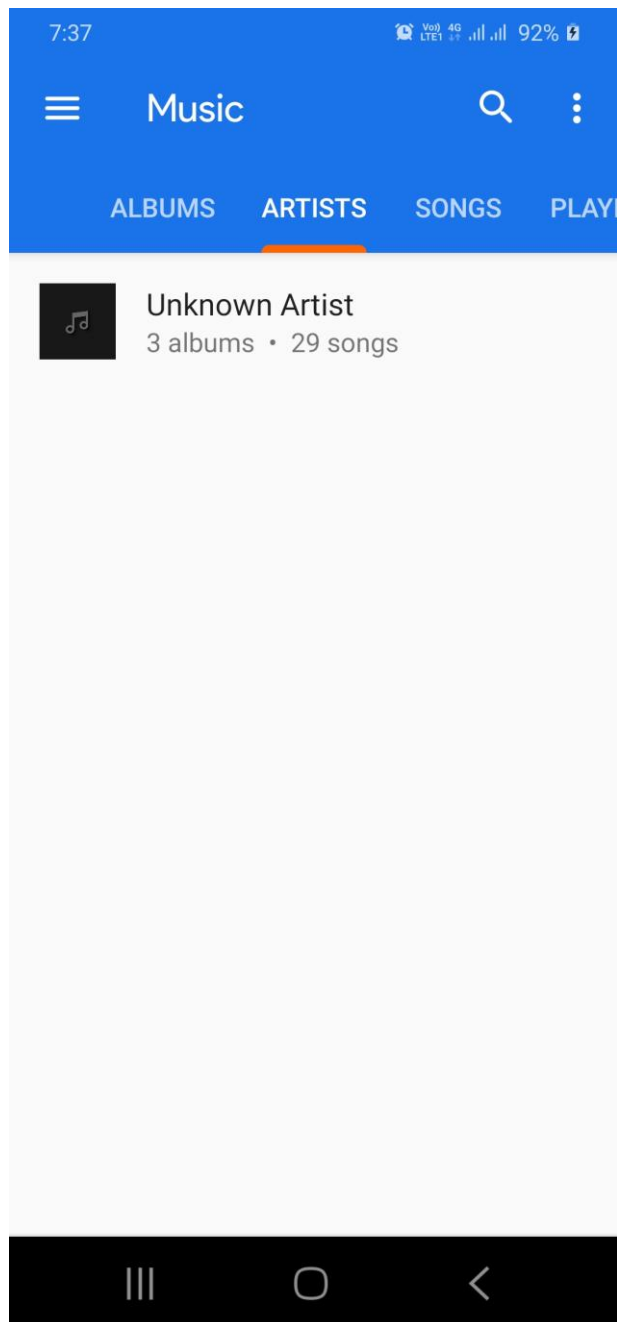


Figure 5.1.2 This displays 'THE ARTIST PAGE OF APP'

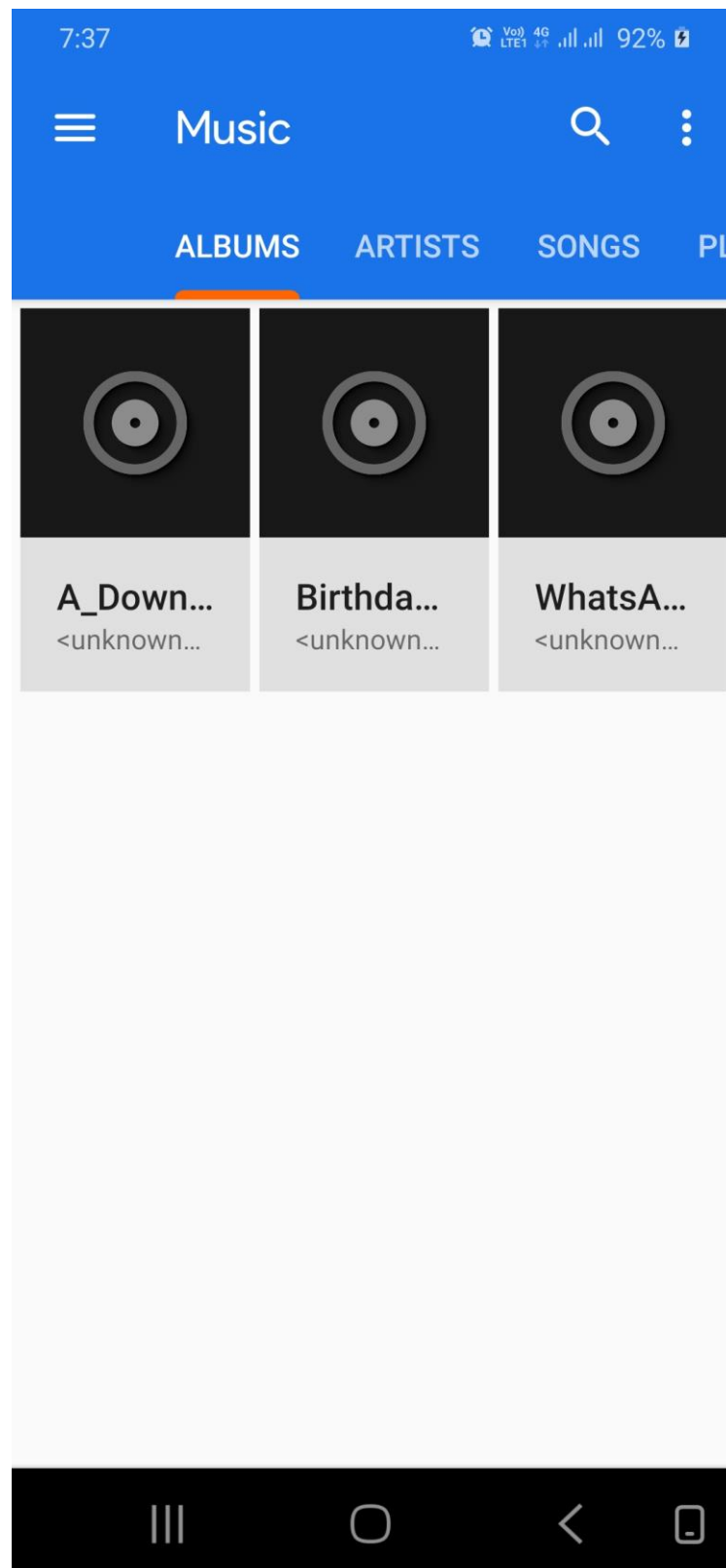


Figure 5.1.3 This displays 'THE ALBUMS PAGE OF APP'

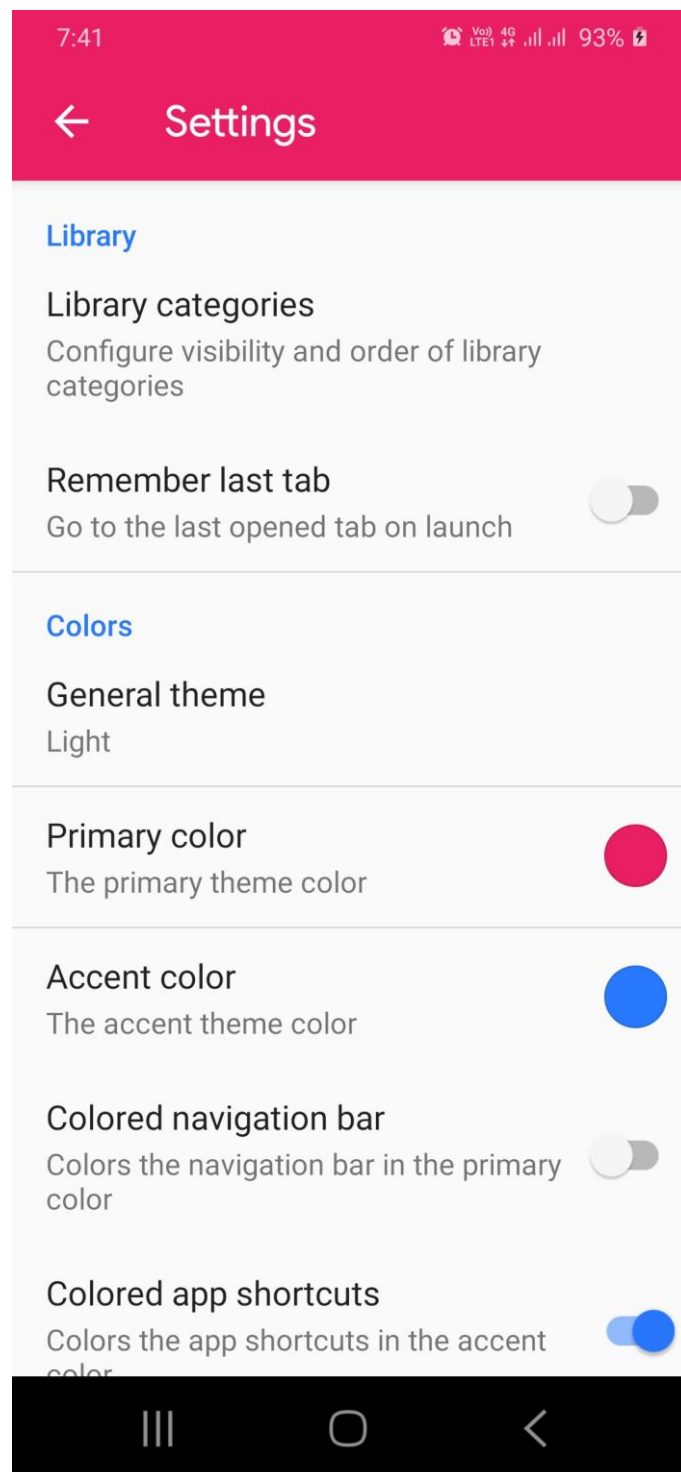


Figure 5.1.4 This displays 'THE SETTINGS PAGE OF APP'

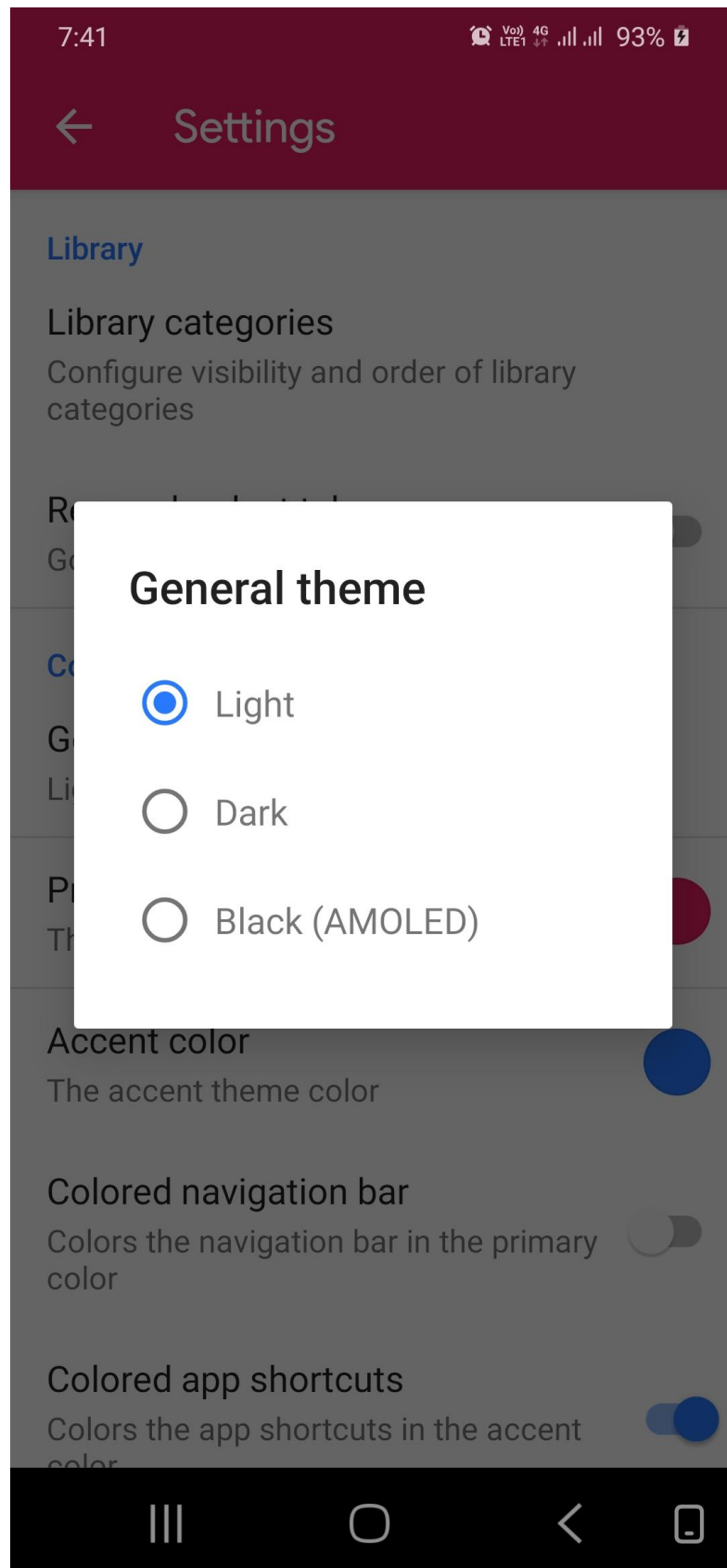


Figure 5.1.5 This displays 'THE SETTING DARK MODE PAGE OF APP'

Chapter6

FUTURE WORK

- I. Enhanced interactivity, allowing users to open song playlist when they swipe up from the music playing interface
- II. Refactoring code, rebuild the coding structure to make the coding look cleaner, easier to understand and perform efficiently
- III. Cross-platform, running the app on IOS, not only Android information

Conclusion

Through the development of music player on Android platform, we get a clear understanding of overall process of the system. The core part of the music player is mainly composed of main interface, file browsing and song listing, Grasping the development of the music player has had the preliminary scale small features. Music player system realized the basic function of player: play, pause, rewind and fast forward a, volume adjustment is performed through the Android System Itself, play mode, song search, seek bar. This development implicated the popular mobile terminal development technology. This design of music player based on Android system requires elaborate design of the music player framework, by adopting ANDROID STUDIO 3.1.2 + Java language as technical support of this system, with the Android plug-in tools, and combination of Latest Android SDK version lead to the comprehensive and smoothly design and development of the mobile terminal

Bibilography

1. Google Developer Training, “Android Developer Fundamentals Course – Concept Reference, Google Developer Training Team, 2017. <https://www.gitbook.com/book/google-developer-training/android-developer-fundamentals-course-concepts/details> (Download pdf file from the above link)
2. Erik Hellman, “Android Programming – Pushing the Limits”, 1 st Edition, Wiley India Pvt Ltd, 2014. ISBN-13: 978-8126547197
3. Dawn Griffiths and David Griffiths, “Head First Android Development”, 1st Edition, O’Reilly SPD Publishers, 2015. ISBN-13: 978-9352131341
4. Bill Phillips, Chris Stewart and Kristin Marsicano, “Android Programming: The Big Nerd Ranch Guide”, 3 rd Edition, Big Nerd Ranch Guides, 2017. ISBN-13: 978-0134706054
5. <https://developer.android.com>
6. <https://www.github.com>