May 2025

ANDROID MUSIC APPLICATION

Submitted by

GROUP ID : BCA22I003

Comprising of

Roll Number	Registration Number	Student Code	Student Name
22010301413	22013002851 of 2022-2023	BWU/BCA/22/485	Rohit Vishwakarma
22010301414	22013002852 of 2022-2023	BWU/BCA/22/486	Priyangshu Debnath
22010301415	22013002853 of 2022-2023	BWU/BCA/22/487	Himanish Karmakar
22010301416	22013002854 of 2022-2023	BWU/BCA/22/488	Akashdeep Ghosh
22010301439	22013002886 of 2022-2023	BWU/BCA/22/521	Golam Ahmed Mortaza
22010301453	22013002900 of 2022-2023	BWU/BCA/22/535	Mrinmoy Choudhury
22010301593	22013003030 of 2022-2023	BWU/BCA/22/484	Richik Giri

in partial fulfillment for award of the degree of

Bachelor of Computer Applications

in

Department of Computational Sciences

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

Department of Computational Sciences

BONAFIDE CERTIFICATE

Certified that this project report "**NOCTUNE ANDROID MUSIC APPLICATION**" is the bonafide work of group **Rohit Vishwakarma, Priyangshu Debnath, Himanish Karmakar, Akashdeep Ghosh, Golam Ahmed Mortaza, Mrinmoy Choudhury & Richik Giri** who carried out the project work under my supervision.

SIGNATURE

DR. Jayanta Aich

HEAD OF THE DEPARTMENT

Department of Computational Sciences

University Building 6, Room. 305

Brainware University, Barasat, West Bengal, India

SIGNATURE

Shyamashree Singha

SUPERVISOR

Asst. Professor

Department of Computational Sciences

ACKNOWLEDGEMENT

Project Title: ANDROID MUSIC APPLICATION
Project Group ID: BCA22I003

We take this opportunity to sincerely thank everyone who played a role in the successful development and completion of our project. Our deepest appreciation goes to Shyamashree Singha, our project guide, whose expert advice, continuous encouragement, and helpful feedback consistently guided us in the right direction. Her support helped us stay focused and committed throughout the project journey. We are also thankful to Dr. Jayanta Aich, Head of the Department of Computational Sciences, Brainware University, for creating a supportive academic atmosphere and for giving us the platform to explore and implement our ideas. A warm thanks goes to our friends and batchmates who stood by us with valuable suggestions, motivation, and a sense of camaraderie that made even the toughest phases manageable. Lastly, we owe heartfelt thanks to our families for their patience, emotional support, and constant faith in our abilities. Their belief in us inspired us to give our best at every step. This project is not just a product of our effort but a reflection of the encouragement, help, and cooperation of all those mentioned above. We are truly grateful.

Project Members:

1. BWU/BCA/22/485 Rohit Vishwakarma
2. BWU/BCA/22/486 Priyangshu Debnath
3. BWU/BCA/22/487 Himanish Karmakar
4. BWU/BCA/22/488 Akashdeep Ghosh
5. BWU/BCA/22/521 Golam Ahmed Mortaza
6. BWU/BCA/22/535 Mrinmoy Choudhury
7. BWU/BCA/22/484 Richik Giri

Date:

Department of Computational Sciences
Brainware University

ABSTRACT

This project presents the development of a feature-rich music application designed to deliver a seamless and efficient offline music streaming experience. With the increasing demand for media consumption in areas with limited or unstable internet connectivity, the core objective of the application is to enable users to enjoy their favorite music without relying on continuous network access. The application allows users to download and organize tracks for offline playback while maintaining essential functionalities such as playlist creation, artist and genre-based categorization, and an intuitive user interface. Core components include local storage management, metadata handling, dynamic search and playback controls optimized for offline usage. Emphasis has been placed on performance optimization, low resource consumption, and a user-friendly design that adheres to modern UI/UX standards. This project aims to bridge the gap between accessibility and functionality, offering a reliable solution for music lovers who prefer or require offline streaming capabilities without compromising on quality or user experience.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The Noctune Android Music Application is a minimalist music player designed to cater to users who prefer simplicity and efficiency in their music listening experience. This app focuses exclusively on playing locally stored audio files, eliminating complex features like online streaming, search functionality, or a homepage. By concentrating on core playback features, the app ensures a seamless and distraction-free user experience.

Built using XML for the frontend and Java for backend functionality, the application incorporates essential music controls such as play, pause, forward, rewind, and track selection. The user interface, crafted in Figma, combines a clean design with dynamic elements to provide a visually appealing and intuitive experience. The app's lightweight design and streamlined features make it suitable for a wide range of Android devices.

This project not only highlights the technical capabilities of the development team but also serves as a demonstration of their ability to create user-centric applications. With a focus on usability, performance, and aesthetics, the Android Music Application stands as a testament to the potential of effective design and programming integration.

CHAPTER 2

OBJECTIVE

The primary objective of this project is to design and develop an efficient Android Music Application that focuses on simplicity and functionality. By prioritizing the playback of locally stored songs, the app eliminates unnecessary complexities and ensures a user-centric experience. The project emphasizes delivering a lightweight and aesthetically pleasing application that caters to music enthusiasts looking for straightforward playback solutions.

Key Objectives

1. **Seamless Playback**: Provide uninterrupted music playback with smooth transitions between tracks.
2. **Core Music Controls**: Integrate essential features like play, pause, fast-forward, rewind, and track skipping.
3. **Minimalistic UI Design**: Develop a clean and interactive interface using Figma to enhance usability.
4. **Local File Accessibility**: Ensure the app efficiently accesses and manages songs stored on the device.
5. **Notification Integration**: Allow users to control playback directly from the notification panel for added convenience.
6. **Device Compatibility**: Optimize the app to perform consistently across a range of Android devices.
7. **Performance Focus**: Prioritize quick loading times and responsive interactions for a seamless user experience.
8. **User-Centric Approach**: Eliminate complex features to maintain a straightforward and distraction-free application.

CHAPTER 3

PLANNING

Planning

The development of the Android Music Application followed a structured and systematic approach to ensure efficient execution and timely completion. Each phase was designed to address specific project needs, from initial concept to final implementation.

Phase 1: Conceptualization

The team began by outlining the scope of the application, focusing on local audio playback and core functionalities like play, pause, forward, and rewind. Simplifying the user experience by excluding complex features such as online streaming or search was a key decision to keep the app lightweight.

Phase 2: Design and Prototyping

Using Figma, the team crafted a clean and user-friendly interface with minimalistic aesthetics and interactive elements. Prototypes were refined over multiple iterations to finalize a layout that balanced functionality and visual appeal.

Phase 3: Technology Stack Selection

XML and Java were chosen for frontend and backend development, respectively, while Android Studio served as the development platform. This combination ensured compatibility, stability, and ease of maintenance.

Phase 4: Development and Testing

Frontend and backend development progressed in parallel, integrating essential playback features and UI elements. Regular testing was conducted to address bugs, optimize performance, and ensure the app worked seamlessly across devices.

This phased planning approach ensured the project was executed effectively, meeting both functional and aesthetic goals.

CHAPTER 4

REQUIREMENT ANALYSIS

The Android Music Application was designed with a focus on addressing both functional and non-functional requirements to deliver a streamlined user experience.

Functional Requirements

1. **Playback Controls**: Provide essential features like play, pause, rewind, fast-forward, next, and previous track.
2. **Local File Integration**: Seamlessly access and play locally stored songs.
3. **Dynamic Song List**: Display all available tracks in a clean, organized list view.
4. **Notification Controls**: Enable playback management directly from the notification panel.
5. **Permission Handling**: Request and manage permissions for media and notification access effectively.

Non-Functional Requirements

1. **User Interface**: Maintain a clean, minimalistic design with dynamic elements like progress bars.
2. **Performance**: Ensure fast loading, efficient file handling, and smooth track transitions.
3. **Device Compatibility**: Support a wide range of Android devices and OS versions.
4. **Error Handling**: Provide clear messages for issues like denied permissions or unsupported formats.

Technical Requirements

1. **Development Tools**: Figma for UI design and Android Studio for coding.
2. **Languages**: XML for the frontend and Java for backend logic.
3. **Frameworks**: Android SDK for system integration and app functionality.

CHAPTER 5

SYSTEM FLOW (DATA FLOW DIAGRAM LEVEL - 0)

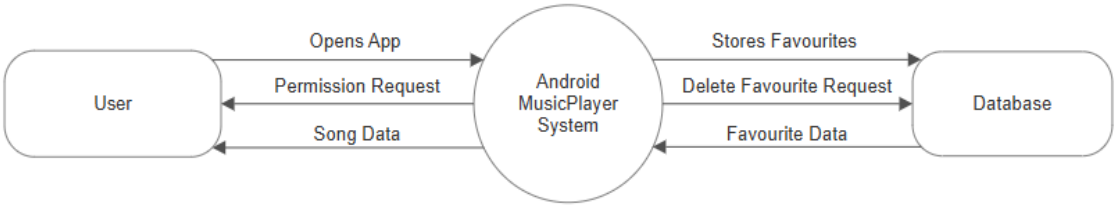


IMAGE 1.0

SYSTEM FLOW (DATA FLOW DIAGRAM LEVEL - 1)

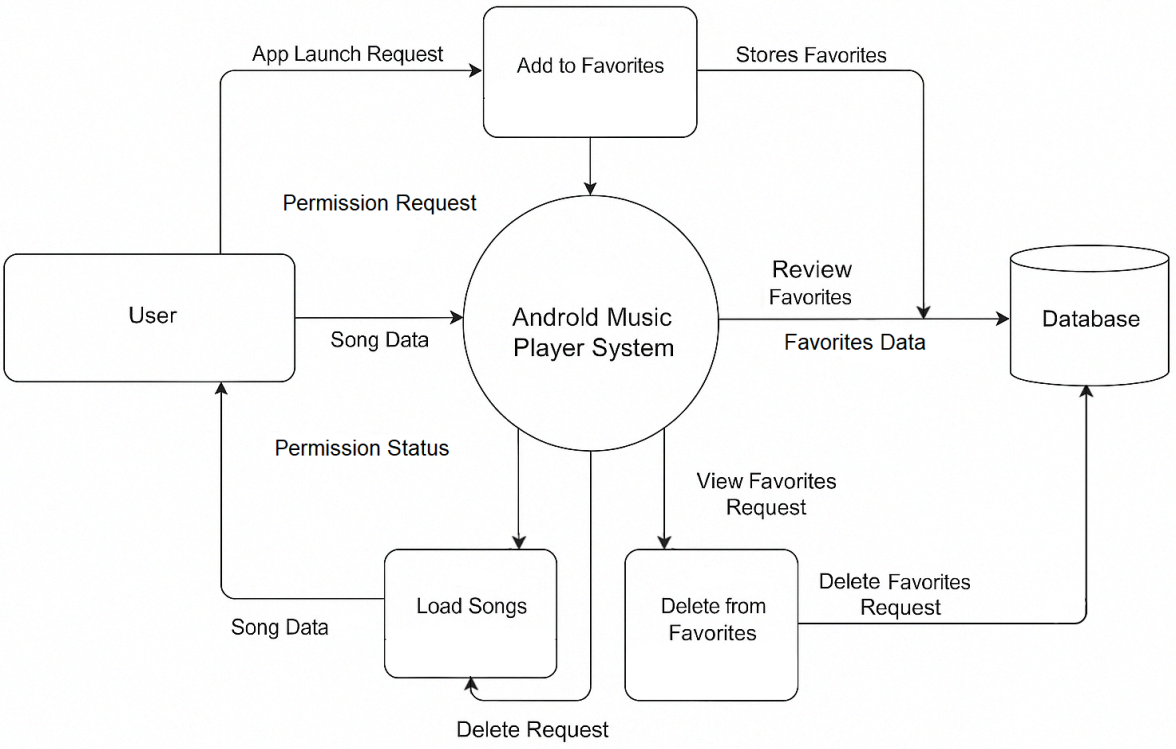
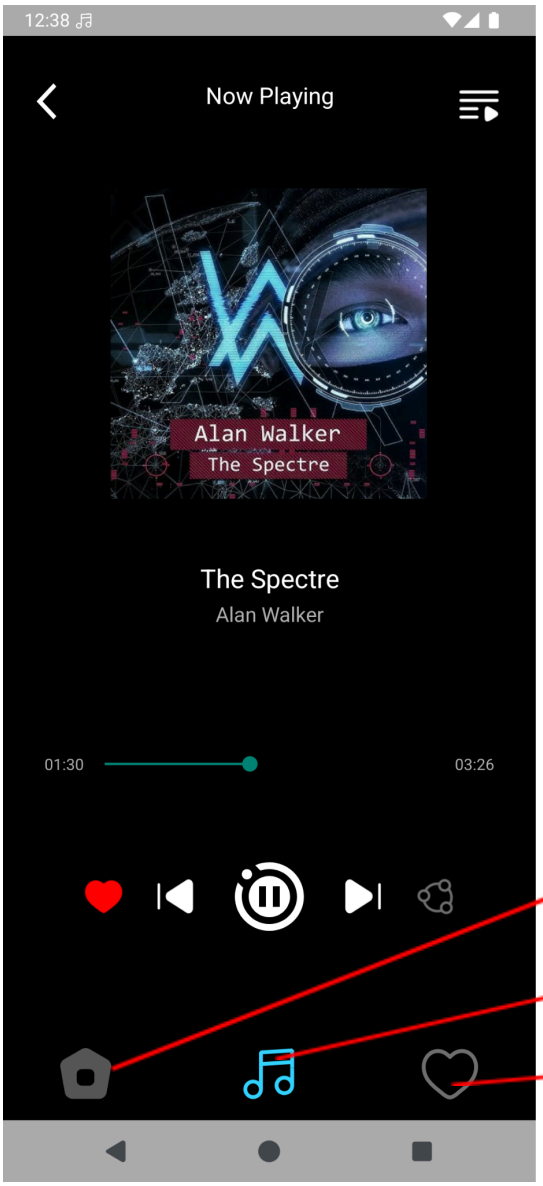


IMAGE 1.1

CHAPTER 6

PROPOSED DESIGN

Music Player



The given Design is considered as the initial design for our app
"Noctune" or Our Music player APP which was our initial name.
This design is solely focusing upon playing music with some simple function, like play, pause, fast forward, add to favorite etc.

1. User Interface(UI)
   I. Clean and minimalist layout with interactive elements.
   II. Dynamic Animations for transitions for better user experience.

2. User Experience(UX)
   I. Simple Button for quick access, from the app and from the notification itself.
   II. Visual progress bar including the current duration of songs.

IMAGE 1.2

So, this page was for playing the songs, but while finalizing this layout we discovered that the app keeps crashing whenever a user tries to click on the given interactive buttons. We face the technical problems which were,

PROBLEMS

1. It was discovered that the format of the buttons were in .png (Portable Network Graphics)
2. And secondly, we initially implemented the buttons ImageView instead of ViewImageButton.

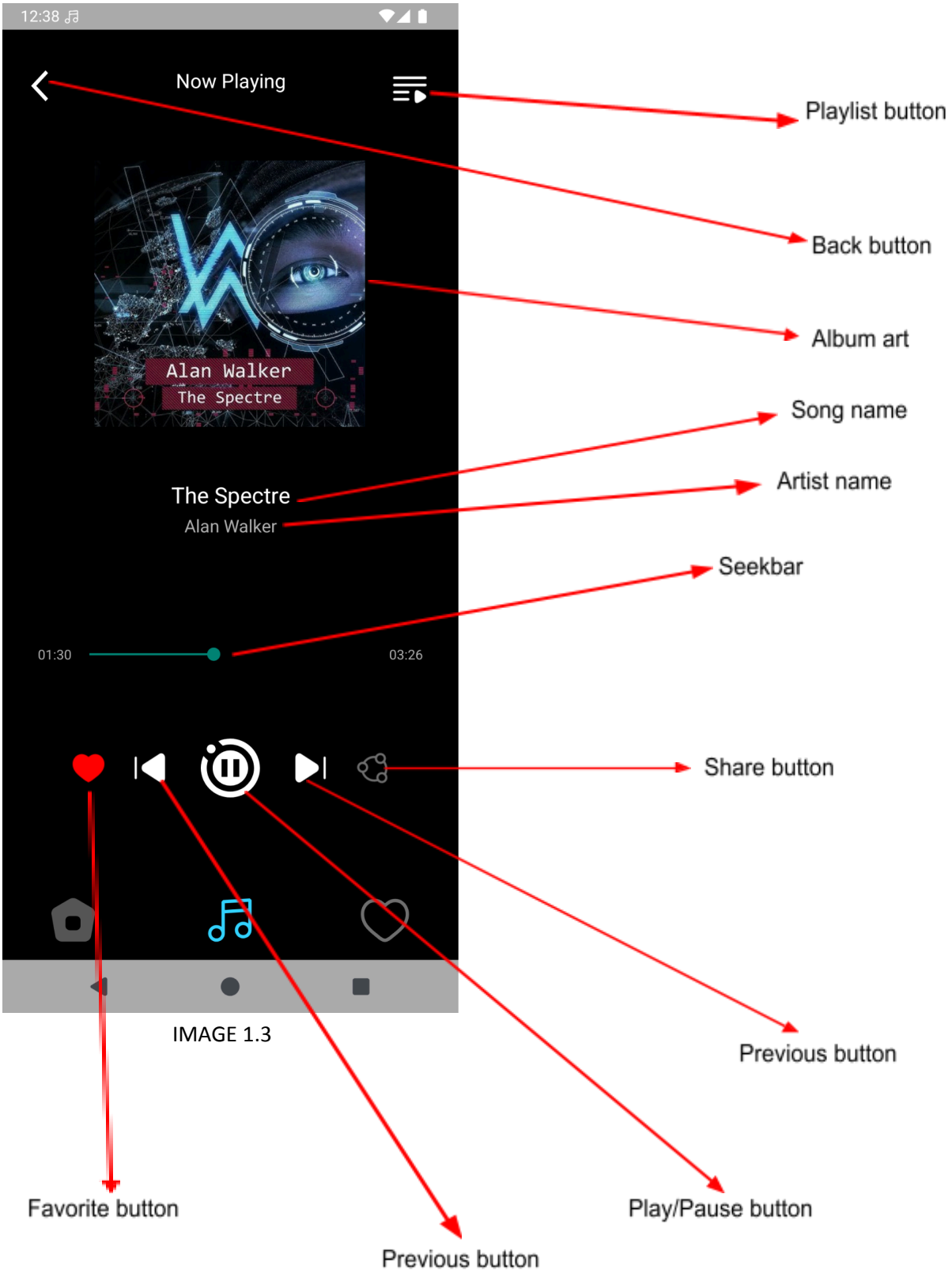

12:38
Now Playing
Alan Walker
The Spectre
The Spectre
Alan Walker
01:30
03:26
Playlist button
Back button
Album art
Song name
Artist name
Seekbar
Share button
Previous button
Play/Pause button
Previous button
Favorite button

IMAGE 1.3

1. Here is the Xml for the music layout which was finalized for the project.

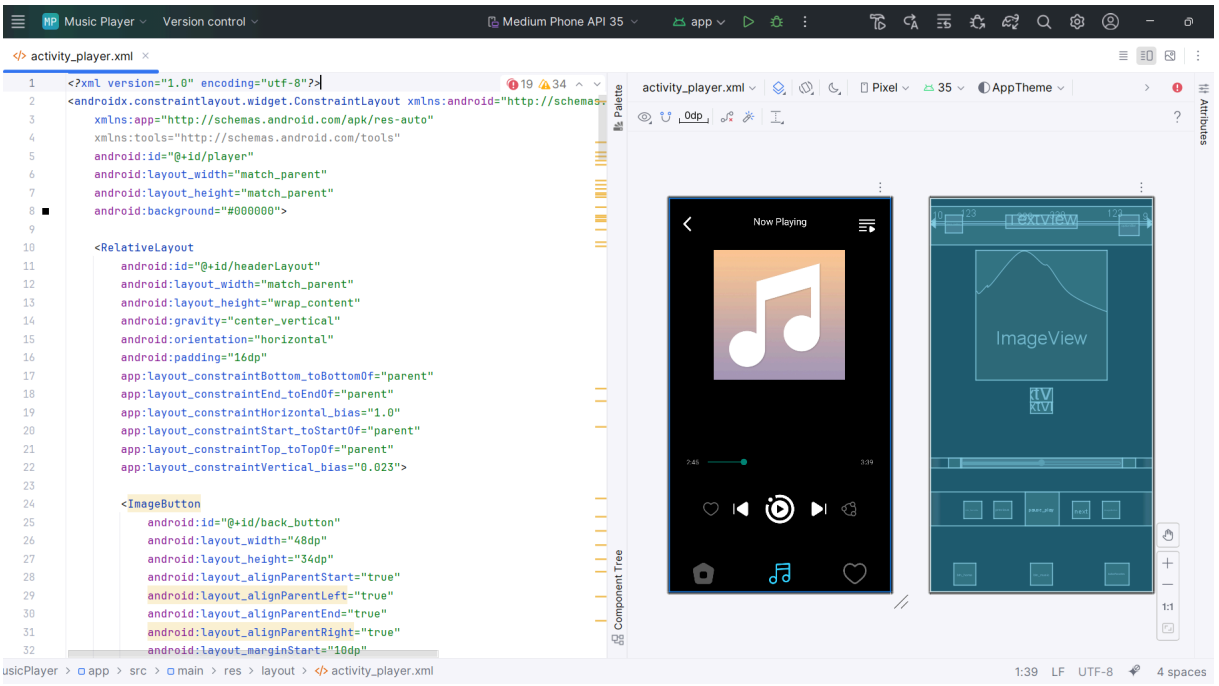


IMAGE 1.4

In the above image(IMAGE 1.4) we gave the elements for the basic functionality of the playing the music, were we can PLAY, PAUSE, CHANGE TO NEXT AND PREVIOUS SONGS with some personalised functions like ADD TO FAVORITES, SHARING OF THE SONG AND A PLAY LIST BUTTON for easier access.

2. The xml for the two codes are given below;

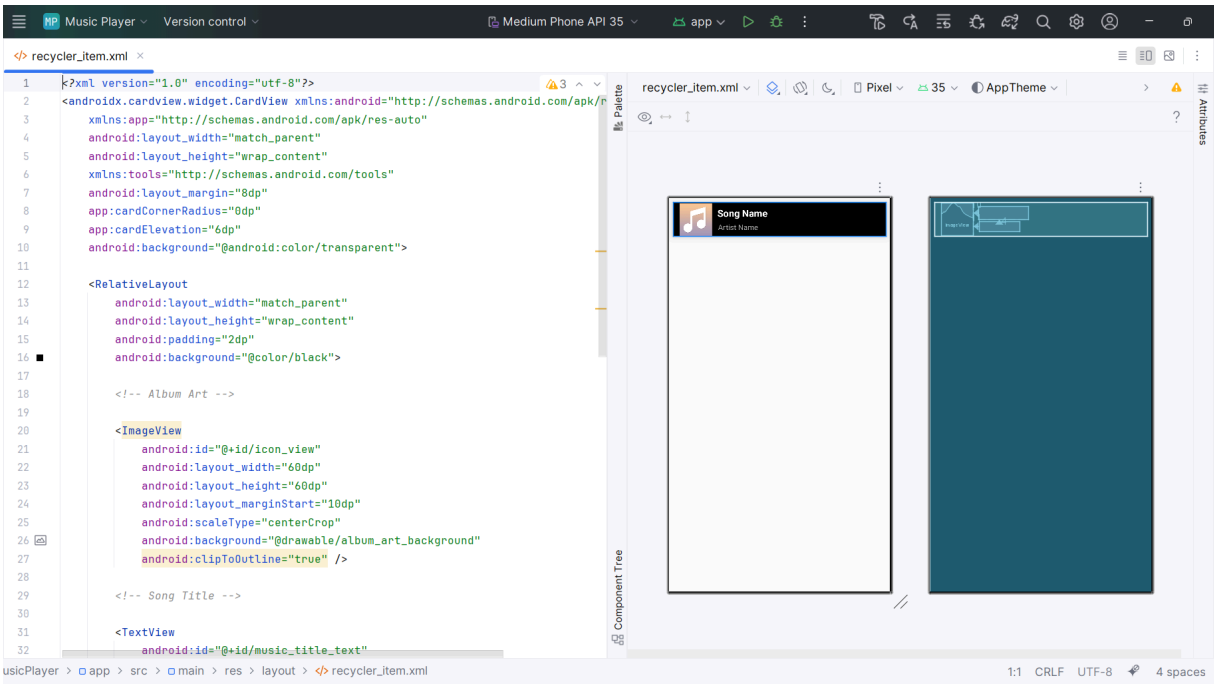


Image 1.5

In this XML the individual songs have the dedicated Song name and the artist name with the album art if they have any.

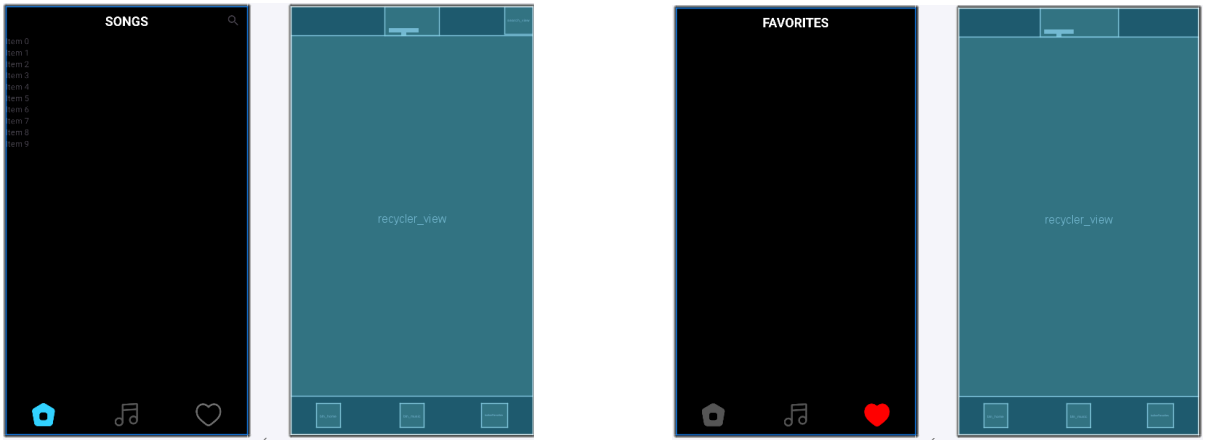


IMAGE 1.6

In 1.6 the songs will be shown in the form of a list. The users can just selected the preferable song's they wanted they want to listen too.

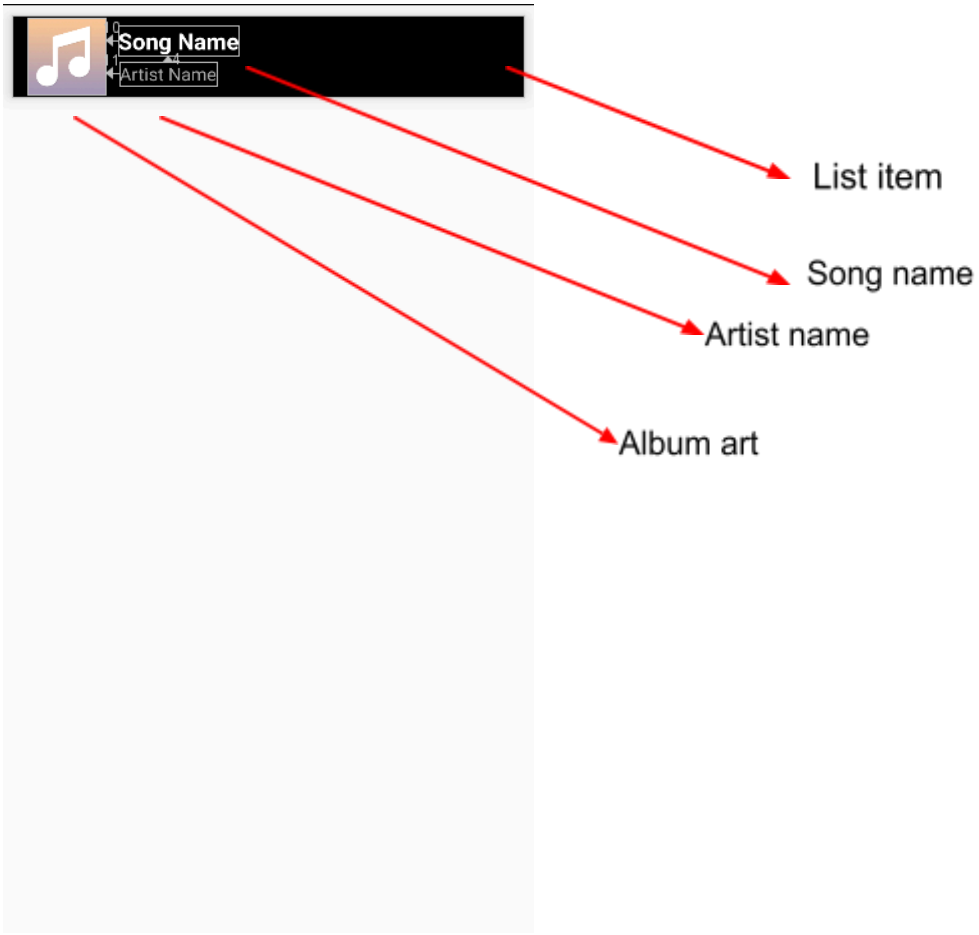


IMAGE 1.7

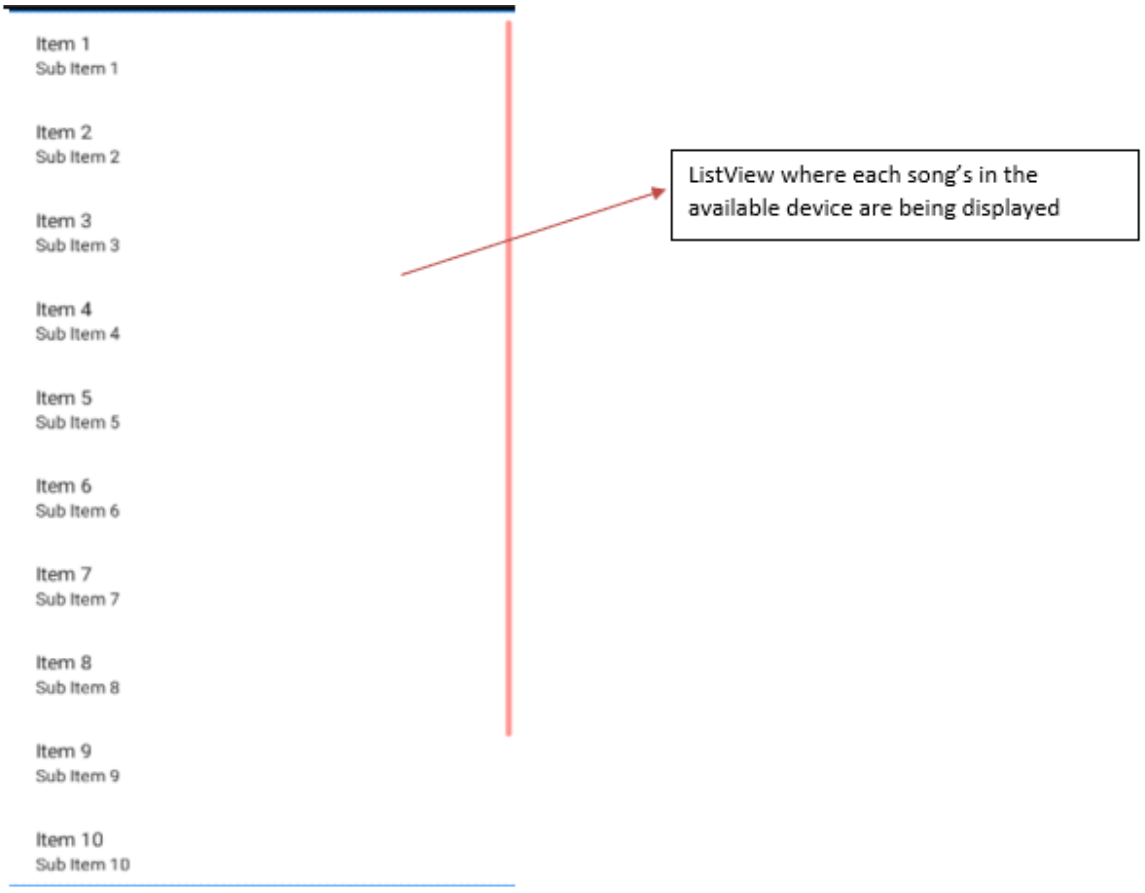
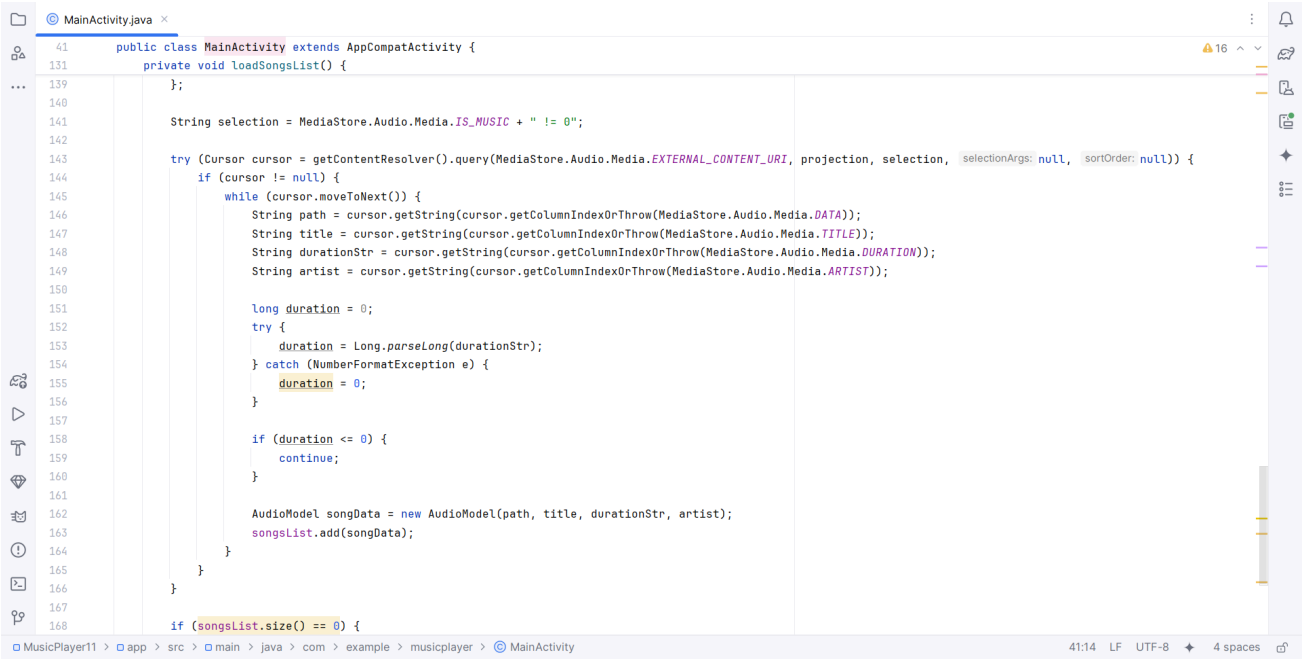


IMAGE 1.8

CHAPTER 7

SAMPLE CORE CODE

MainActivity.java

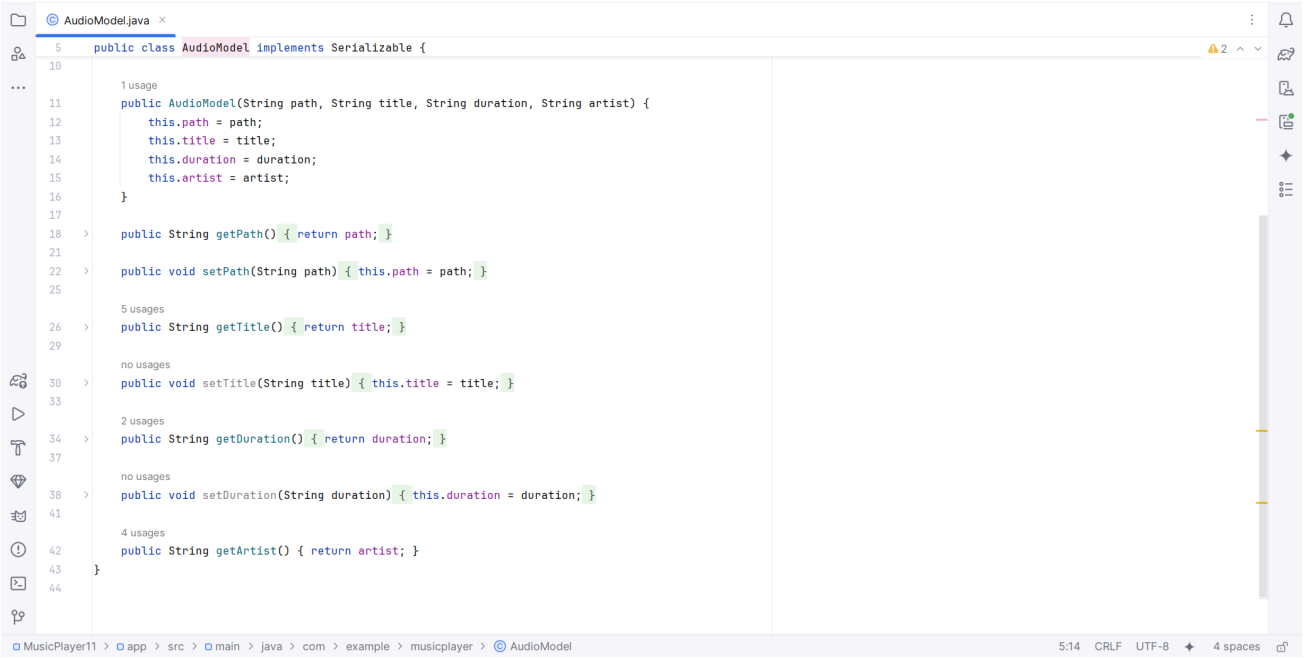


```
public class MainActivity extends AppCompatActivity {
    private void loadSongsList() {
        };

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

        try (Cursor cursor = getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection, selectionArgs: null, sortOrder: null)) {
            if (cursor != null) {
                while (cursor.moveToNext()) {
                    String path = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DATA));
                    String title = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.TITLE));
                    String durationStr = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DURATION));
                    String artist = cursor.getString(cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.ARTIST));

                    long duration = 0;
                    try {
                        duration = Long.parseLong(durationStr);
                    } catch (NumberFormatException e) {
                        duration = 0;
                    }

                    if (duration <= 0) {
                        continue;
                    }

                    AudioModel songData = new AudioModel(path, title, durationStr, artist);
                    songsList.add(songData);
                }
            }
        }

        if (songsList.size() == 0) {
```

IMAGE 1.9

- **Purpose**: Serves as the main entry point, displaying all available songs and providing search and navigation features.
- **Key Functionality**: Loads songs from device storage, handles runtime permissions, filters songs via SearchView, and navigates to MusicPlayerActivity or FavoriteListActivity.

AudioModel.java



```
public class AudioModel implements Serializable {

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

    public String getPath() { return path; }

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

    5 usages
    public String getTitle() { return title; }

    no usages
    public void setTitle(String title) { this.title = title; }

    2 usages
    public String getDuration() { return duration; }

    no usages
    public void setDuration(String duration) { this.duration = duration; }

    4 usages
    public String getArtist() { return artist; }
}
```

IMAGE 2.0

- **Purpose**: Represents a music file's metadata as a serializable data model.
- **Key Functionality**: Stores and provides access to song attributes like path, title, duration, and artist.

MyMediaPlayer.java

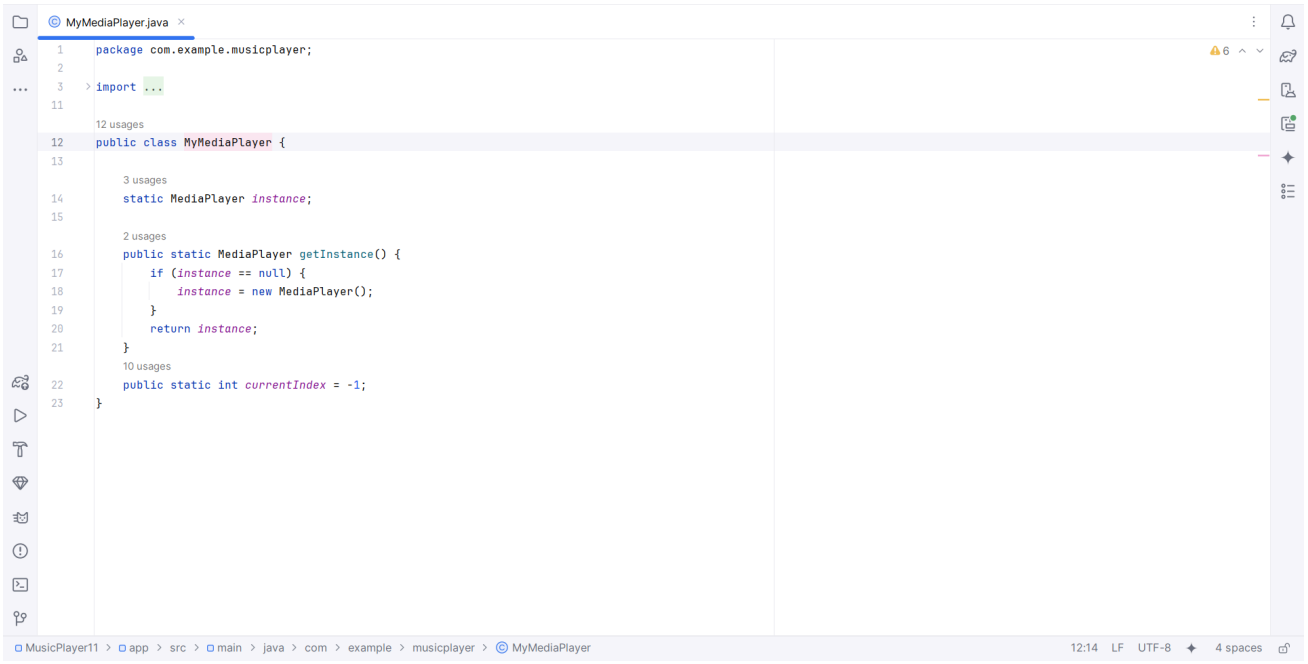


IMAGE 2.1

- **Purpose**: Manages a singleton MediaPlayer instance for audio playback.
- **Key Functionality**: Maintains a single MediaPlayer instance and tracks the current song index for playback control across activities.

MusicPlayerActivity.java

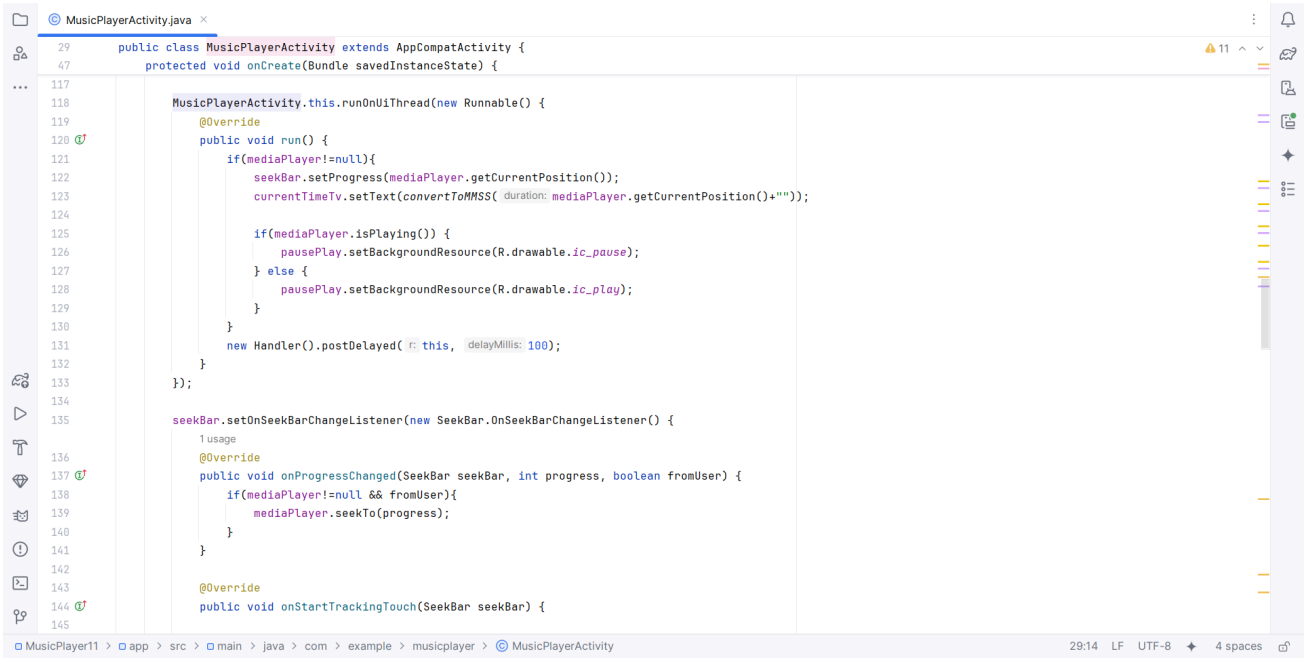


IMAGE 2.2

- **Purpose**: Handles music playback and provides an UI for controlling the currently playing song.
- **Key Functionality**: Plays, pauses, skips songs, displays song details and album art, updates seek bar progress, and manages favorite song status.

MusicListAdapter.java

IMAGE 2.3

- **Purpose**: Manages the display of the music list in a RecyclerView.
- **Key Functionality**: Binds song data (title, artist, album art) to views, handles click events to play songs, caches album art for performance, and supports search filtering.

AppDatabase.java

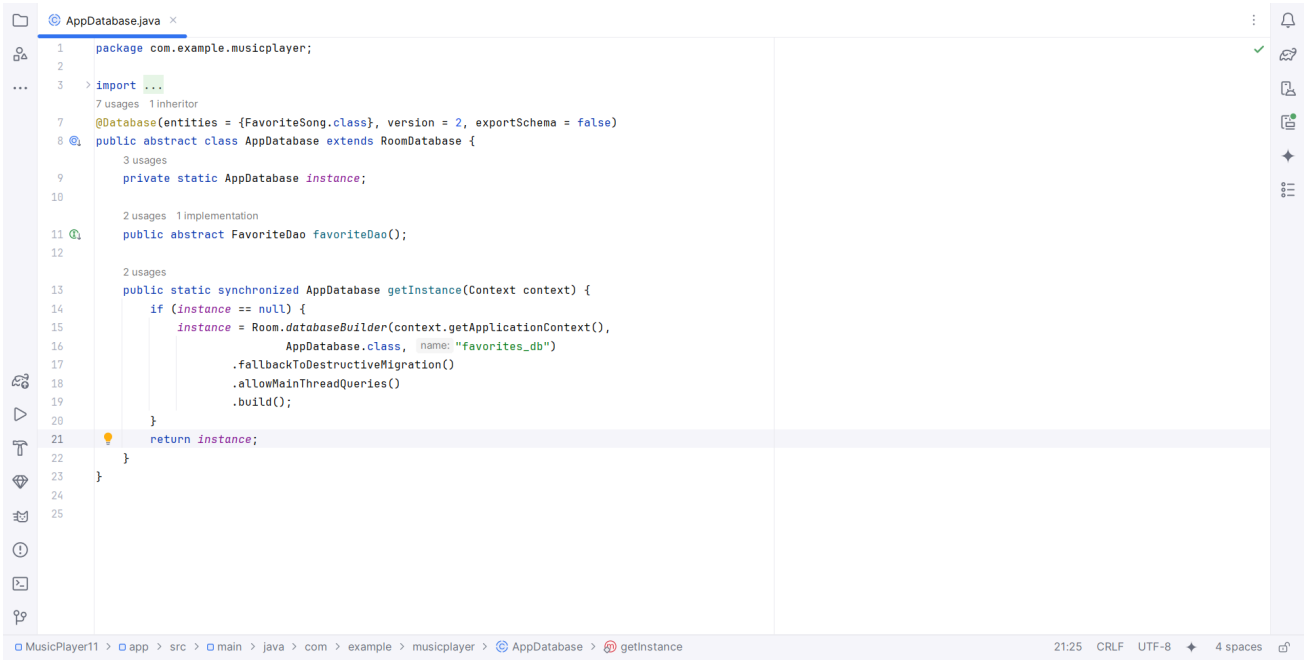
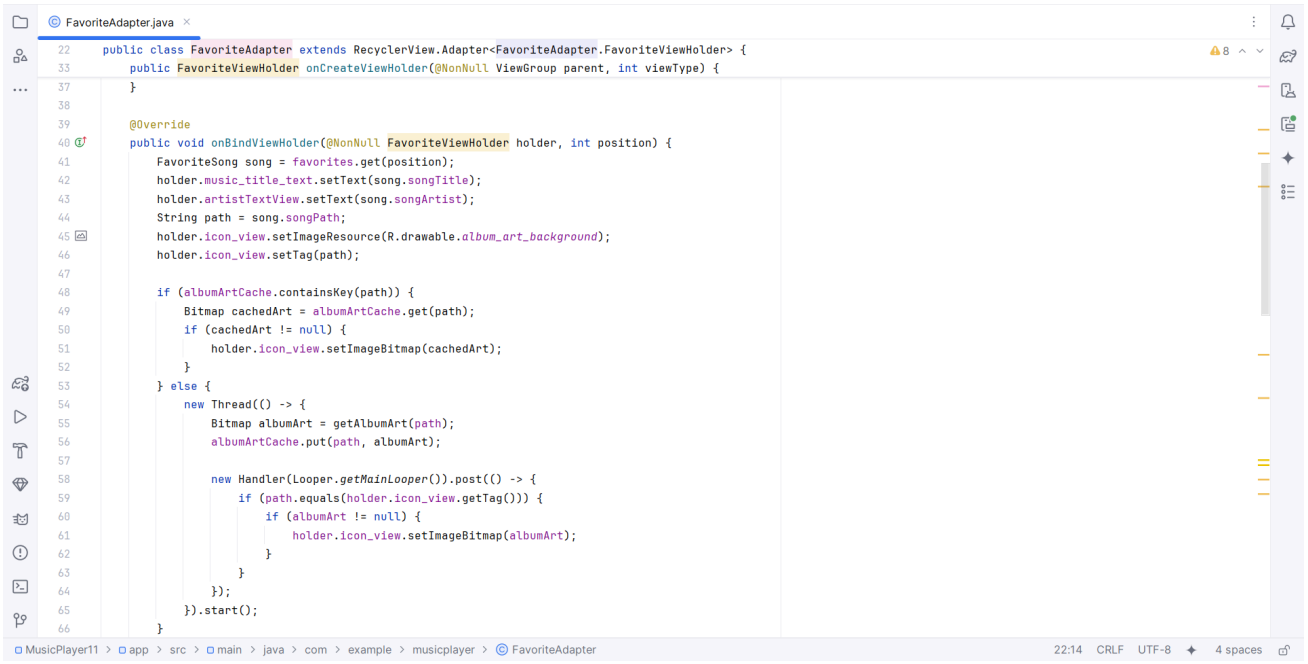


IMAGE 2.4

- **Purpose**: Defines the Room database for storing favorite songs.
- **Key Functionality**: Provides a singleton instance of the database and access to the FavoriteDao for CRUD operations on favorite songs.

FavoriteAdapter.java

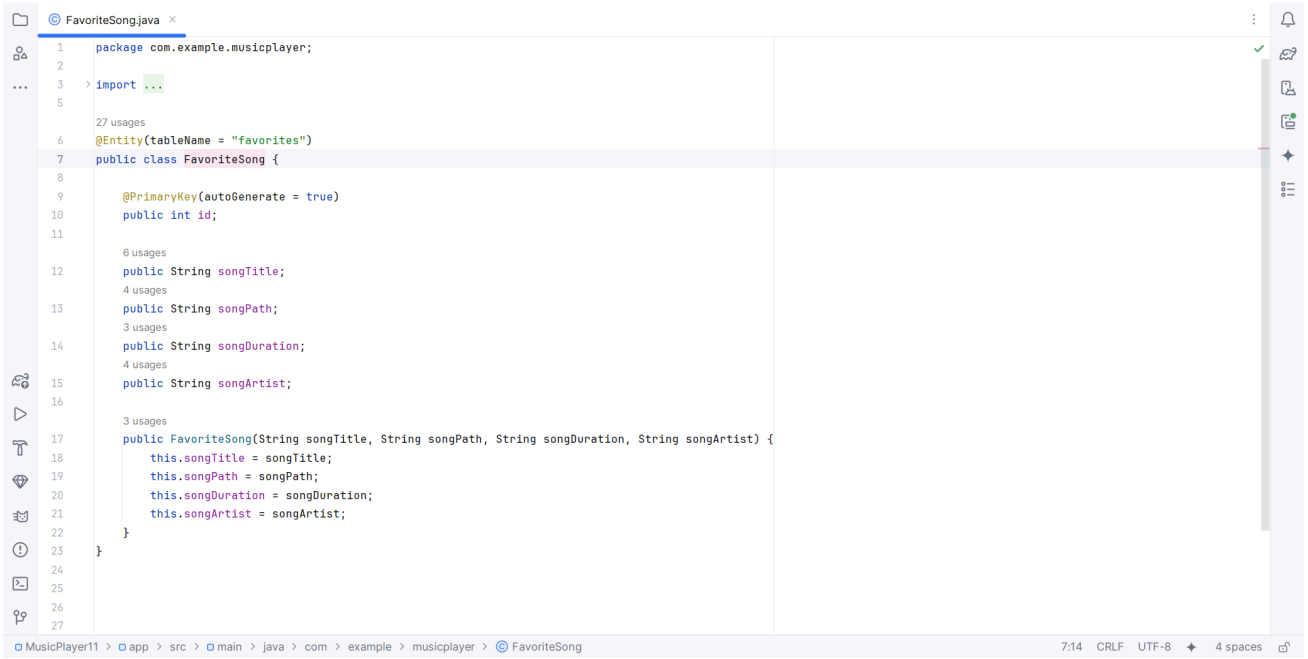


```java
public class FavoriteAdapter extends RecyclerView.Adapter<FavoriteAdapter.FavoriteViewHolder> {
    public FavoriteViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    }

    @Override
    public void onBindViewHolder(@NonNull FavoriteViewHolder holder, int position) {
        FavoriteSong song = favorites.get(position);
        holder.music_title_text.setText(song.songTitle);
        holder.artistTextView.setText(song.songArtist);
        String path = song.songPath;
        holder.icon_view.setImageResource(R.drawable.album_art_background);
        holder.icon_view.setTag(path);

        if (albumArtCache.containsKey(path)) {
            Bitmap cachedArt = albumArtCache.get(path);
            if (cachedArt != null) {
                holder.icon_view.setImageBitmap(cachedArt);
            }
        } else {
            new Thread(() -> {
                Bitmap albumArt = getAlbumArt(path);
                albumArtCache.put(path, albumArt);

                new Handler(Looper.getMainLooper()).post(() -> {
                    if (path.equals(holder.icon_view.getTag())) {
                        if (albumArt != null) {
                            holder.icon_view.setImageBitmap(albumArt);
                        }
                    }
                });
            }).start();
        }
```

IMAGE 2.5

- **Purpose**: Manages the display of favorite songs in a RecyclerView within FavoriteListActivity.
- **Key Functionality**: Binds favorite song data (title, artist, album art) to views, caches album art for efficiency, and updates the UI when the favorites list changes.

FavoriteSong.java



```java
package com.example.musicplayer;

import ...

27 usages
@Entity(tableName = "favorites")
public class FavoriteSong {

    @PrimaryKey(autoGenerate = true)
    public int id;

    6 usages
    public String songTitle;
    4 usages
    public String songPath;
    3 usages
    public String songDuration;
    4 usages
    public String songArtist;

    3 usages
    public FavoriteSong(String songTitle, String songPath, String songDuration, String songArtist) {
        this.songTitle = songTitle;
        this.songPath = songPath;
        this.songDuration = songDuration;
        this.songArtist = songArtist;
    }
}
```

IMAGE 2.6

- **Purpose**: Defines the entity for favorite songs stored in the Room database.
- **Key Functionality**: Represents a favorite song with attributes like ID, title, path, duration, and artist, used for persistence in the favorites table.

FavoriteListActivity.java

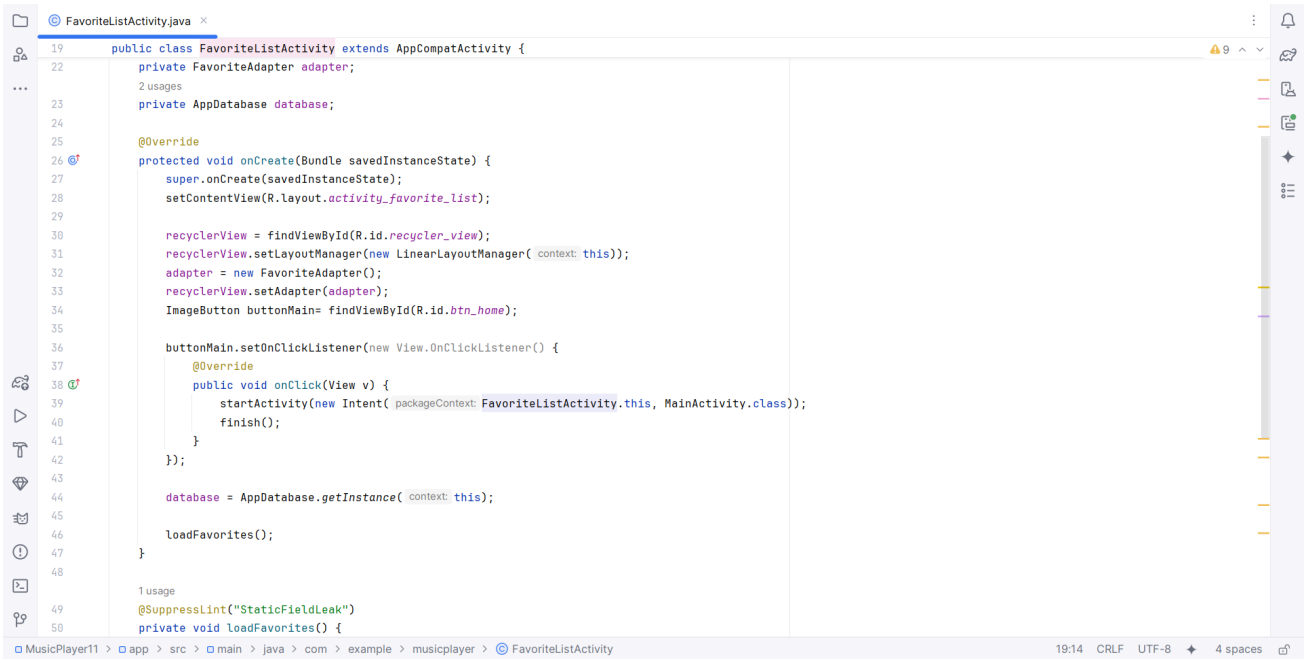


IMAGE 2.7

- **Purpose**: Displays a list of favorite songs stored in the app's database.
- **Key Functionality**: Uses a RecyclerView to show favorite songs, retrieves data from Room database via AsyncTask, and allows navigation back to MainActivity.

All these classes combined together make the core of our Music Player app.

Searchview Functionality:

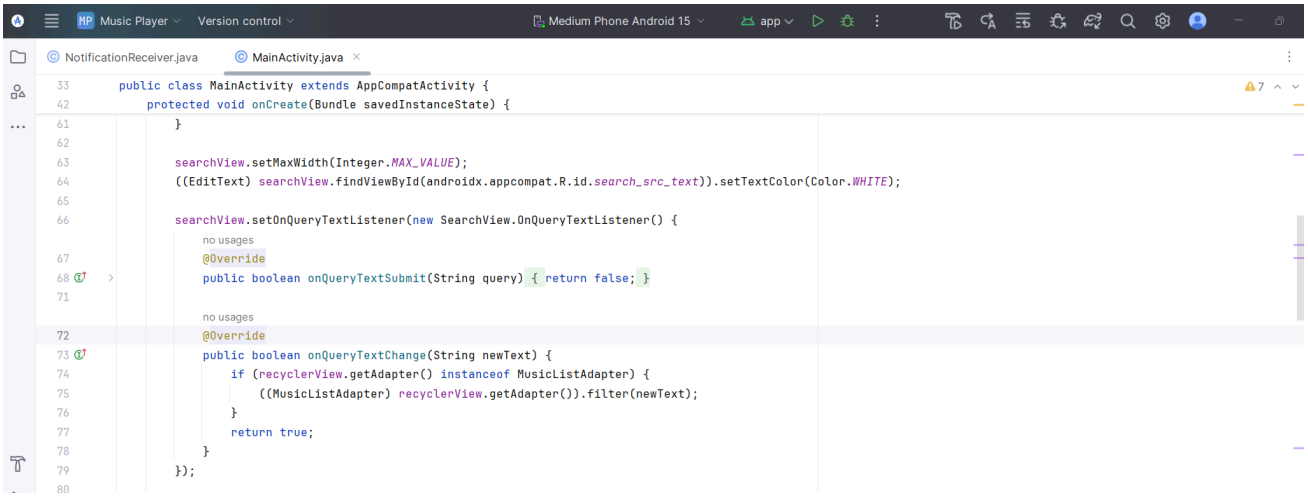


Image:2.8

- **Purpose:**It helps users to filter the song list and quick access of desired songs.
- **Key Functionality:**This code configures a SearchView to allow users to perform real-time filtering of items displayed in a RecyclerView. It is primarily used to improve user experience in applications where quick access to specific list items needed through live search input.

activity_main.xml

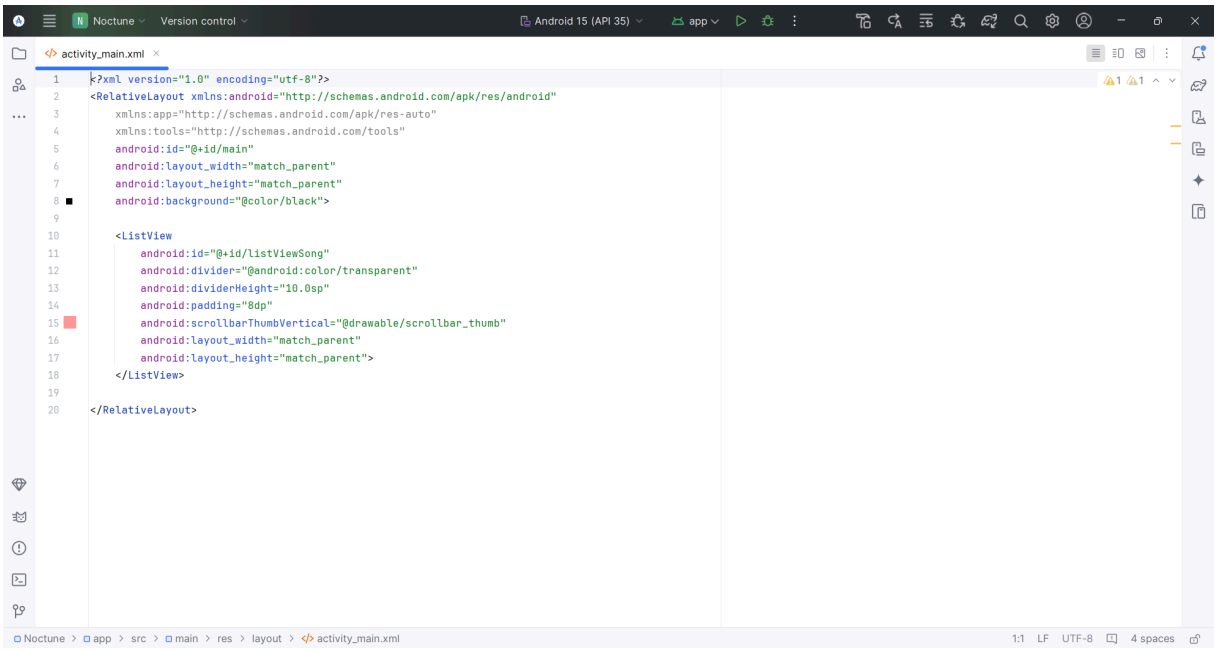


IMAGE 2.9

So, here is the main xml activity were the background color set to black "@color/back" and a List view is added with the id listviewsongs "@id/listviewsongs".

Were the given outputs are,

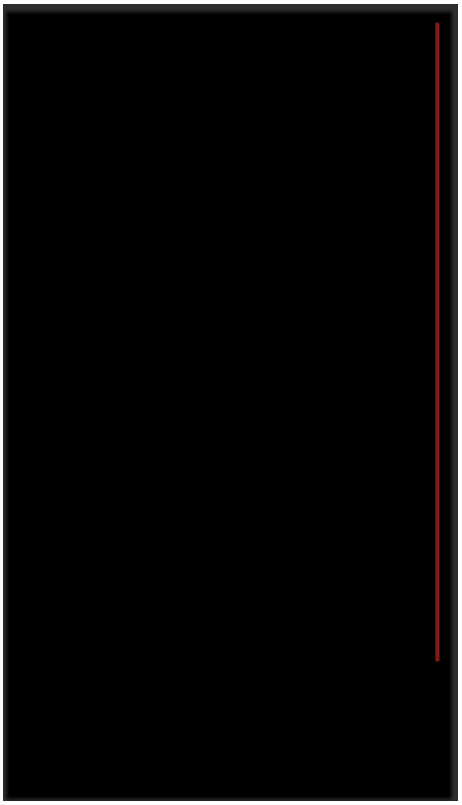


IMAGE 3.0

"@color/back" from xml

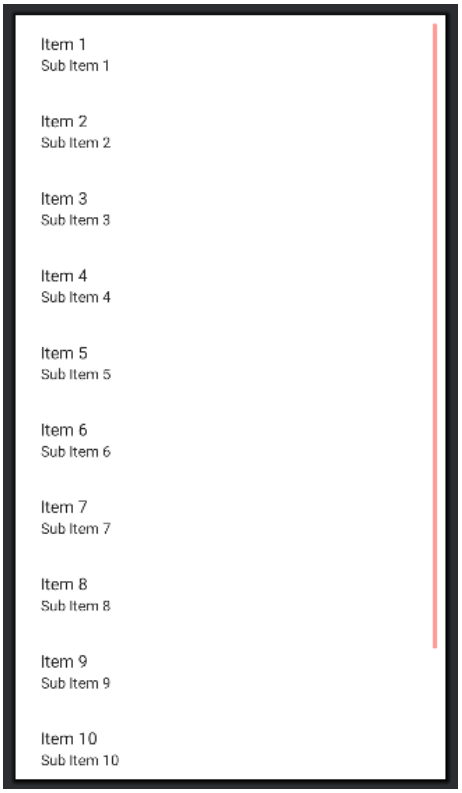


IMAGE 3.1

"@id/listviewsongs" from xml

activity_player.xml

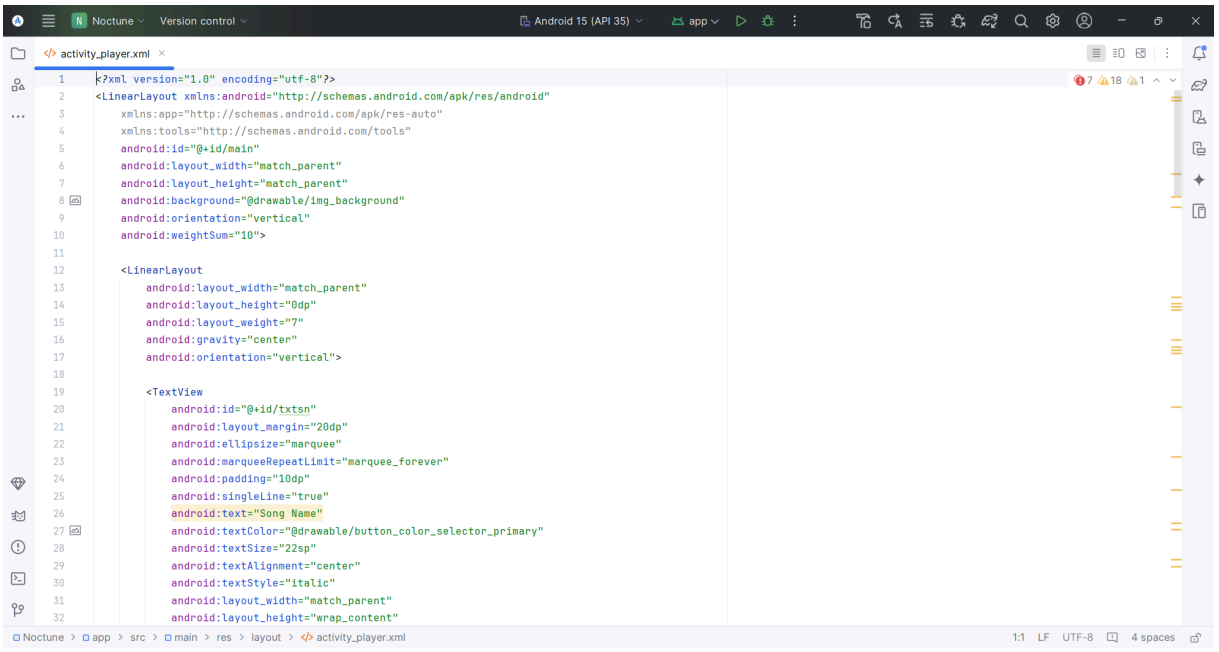


IMAGE 3.2

In the activity_player.xml it shows the main media player for the music app. The whole layout is under a linear Layout with sub layouts which are into the main layout.

The activity are divided into 2 linearLayouts for button implementation which are for the music access, second for the image view and another is for the seekbar with text view for the start duration and end duration of each song.

Below is a quick xml code for the implementation of the seek bar. It includes the following things

1.The interactable seekbar.

2.Start and end time for the music.

IMAGE 3.3 The view for the seek bar

IMAGE 3.4 The overall seek Bar for user

EXPERIMENTAL RESULT

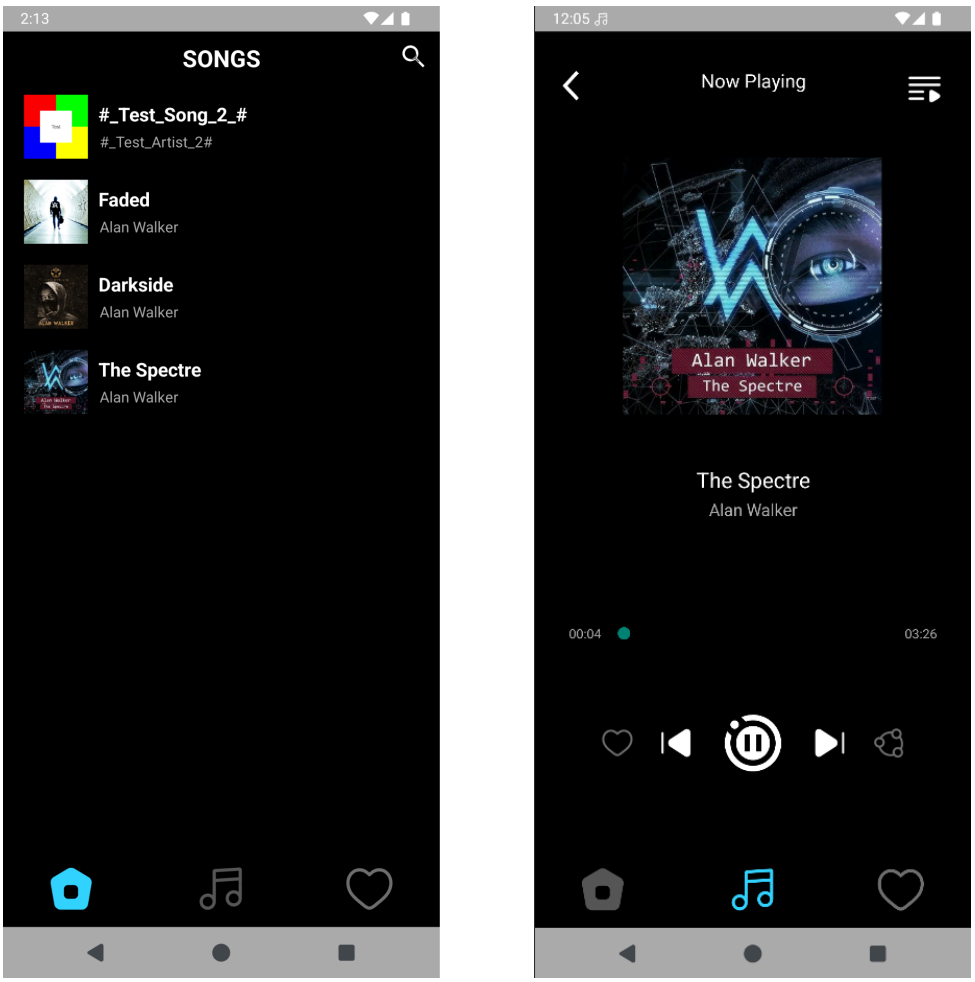


IMAGE 3.5
(Local song is displayed and Played)

CHAPTER 8

FUTURE SCOPE

The Android Music Application has significant potential for growth and enhancement. Future developments will depend on the app's evolving requirements and user feedback. Below are some potential areas for improvement and new feature integration:

1. **Search Implementation**
    - Introduce a search feature to help users quickly find specific songs, artists, or albums from their local library.
    - Enhance the search functionality with filters, such as sorting by genre, artist, or recently added tracks.
2. **Sign-In and Sign-Up Functionality**
    - Add user authentication to enable personalized experiences.
    - Allow users to create accounts and sync preferences, playlists, and favorites across devices.
3. **Profile Page**
    - Implement a profile page where users can manage their personal information, preferences, and app settings.
    - Display statistics like total listening time or most-played tracks to enhance engagement.
4. **Online Streaming**
    - Expand the app’s functionality to support online music streaming from popular platforms or custom APIs.
    - Include features like downloading tracks for offline listening and curated playlists based on user preferences.
5. **Favorite Section**
    - Create a dedicated section for users to save and manage their favorite songs, offering quick access to frequently played tracks.
6. **Home Page**
    - Design a dynamic home page to showcase recently played songs, featured playlists, and personalized recommendations.
7. **UI Overhaul**
    - Revamp the app's design to adopt modern trends, with options for themes and user customization.
8. **Playlist Management**
    - Allow users to create, edit, and organize custom playlists for better music organization.

These future scopes are flexible and will be implemented based on situational needs, user demand, and technical feasibility. This adaptability ensures that the app evolves in line with both technological advancements and user expectations.

CHAPTER 9

CONCLUSION

The Android Music Application successfully demonstrates the integration of simplicity, functionality, and aesthetic design in a music player tailored for local audio playback. By focusing on essential features such as seamless playback, core music controls, and a clean user interface, the app provides a user-centric experience free from unnecessary complexities.

Developed using XML and Java, and designed with Figma, the project showcases the technical and creative capabilities of the development team. The app's lightweight design and efficient performance ensure compatibility with a wide range of Android devices, making it accessible to a broad audience.

This project not only serves as a foundation for understanding mobile application development but also opens the door to numerous enhancements in the future. With planned features like search functionality, user authentication, and an improved user interface, the application holds significant potential for growth and wider adoption.

In conclusion, the Android Music Application is a testament to the team’s commitment to delivering a functional, user-friendly, and visually appealing product, laying the groundwork for further innovations in music app development.

REFERENCES

Books

1. Schildt, H. (2021). *Java: The Complete Reference.* McGraw-Hill Education.
2. Banerjee, S. (2020). *Developing Music Player Apps for Android.* Apress.
3. Norman, D. (2013). *The Design of Everyday Things.* Basic Books.

YouTube Videos

1. "Android Development for Beginners - Full Course" by FreeCodeCamp.org
   (Accessed: 15th May 2025)
   Link: https://www.youtube.com/watch?v=fis26HvvDII

2. "How to Build a Music Player App in Android Studio" by Simplified Coding
   (Accessed: 10th May 2025)
   Link: https://www.youtube.com/watch?v=VR6T3sZiy_8