Android Music Player Application

Project Report

Arjun Mehta (K036)
Dakshita Galhotra (K021)
Jal Bafana (K005)

February 2023

Contents

Abstract

This project developed an Android-based music player application focusing on providing a seamless and intuitive music playback experience. The application leverages the Android framework to deliver core functionalities including audio playback, playlist management, favorites organization, and a modern Material Design user interface. Notably, the app successfully combines local media access with an intuitive user experience, addressing the need for a lightweight yet feature-rich music player. Initial testing shows the application performs effectively across various Android devices, demonstrating its potential as a practical solution for everyday music consumption.

1 Introduction

1.1 Background

Mobile devices have become the primary method for consuming digital music, creating a demand for efficient, user-friendly music applications. While numerous commercial music players exist, many are bloated with unnecessary features or require subscription services, highlighting the need for a streamlined, offline music player.

1.2 Need for the Application

This music player application was developed to address several needs:

- Provide a lightweight alternative to resource-intensive commercial players
- Enable offline music access without requiring continuous internet connectivity
- Offer simple playlist management without account requirements
- Deliver an intuitive interface that follows modern design principles

1.3 Objectives

The primary objectives of this project were to:

1. Develop a fully functional Android music player for local media
2. Implement comprehensive playlist and favorites management
3. Create an intuitive, responsive user interface following Material Design guidelines
4. Ensure efficient media handling and seamless playback experience
5. Build a modular, maintainable application architecture

2 Literature Review

2.1 Existing Solutions

Several popular music player applications were examined during the development process:

- **Spotify:** Industry leader with streaming capabilities and algorithm-based recommendations
- **Google Play Music/YouTube Music:** Integrated with Google's ecosystem with both streaming and local playback
- **VLC Media Player:** Open-source with broad format support but limited playlist management
- **BlackPlayer:** Android-focused with extensive customization but more complex UI

These applications offer valuable insights but also demonstrated limitations that our application addresses, particularly in terms of simplicity and offline usability.

2.2 Technologies and Frameworks

The project leverages several key Android technologies:

- **Android SDK:** Provides the foundation for application development
- **MediaPlayer API:** Core component for audio playback handling
- **SQLite Database:** Lightweight storage solution for playlists and favorites
- **Material Design Components:** UI framework for consistent visual elements
- **Fragment-based Architecture:** Enables modular, responsive layout design

3 Methodology

3.1 Development Approach

The application was developed using an iterative approach, with distinct phases:

1. **Research and Planning:** Analyzing requirements and existing solutions
2. **Core Architecture Design:** Establishing the foundational structure
3. **UI Implementation:** Developing the interface following Material Design principles
4. **Feature Implementation:** Building core functionality incrementally
5. **Testing and Refinement:** Improving performance and user experience

3.2 Tools and Technologies Used

- **Android Studio:** Primary IDE for development
- **Java:** Primary programming language
- **SQLite:** Database for persistent storage
- **Git:** Version control system
- **Gradle:** Build automation tool

3.3 Development Process

Development followed a component-based approach, focusing on individual modules:

1. Media access and playback engine
2. UI design and implementation
3. Database design and integration
4. Playlist management system
5. Integration and system testing

4 System Design

4.1 Architecture

The application follows the Model-View-Controller (MVC) architecture pattern:

- **Model:** Data classes (SongsList, Playlist) and database operations
- **View:** XML layouts and UI components
- **Controller:** Activity and Fragment classes that handle user interactions

4.2 User Interface Design

The UI implements Material Design principles with:

- Navigation drawer for main menu access
- Tab-based interface for content organization
- Player controls with intuitive iconography
- Responsive layouts for different screen sizes
- Consistent color scheme and typography

4.3 Database Schema

The application uses SQLite with three primary tables:

4.3.1 Favorites Table

Stores user's favorite songs

```
CREATE TABLE favorites (
  songID INTEGER,
  title TEXT,
  subtitle TEXT,
  songpath TEXT PRIMARY KEY
)
```

4.3.2 Playlists Table

Stores playlist metadata

```
CREATE TABLE playlists (
  playlistId INTEGER PRIMARY KEY AUTOINCREMENT,
  playlistName TEXT UNIQUE
)
```

4.3.3 Playlist Songs Table

Maps songs to playlists

```
CREATE TABLE playlist_songs (
  playlistId INTEGER,
  songId INTEGER,
  title TEXT,
  subtitle TEXT,
  songpath TEXT,
  PRIMARY KEY (playlistId, songpath),
  FOREIGN KEY (playlistId) REFERENCES playlists(playlistId) ON DELETE CASCADE
)
```

4.4 Component Integration

The application integrates several key components:

- **MediaPlayer:** Handles audio playback and streaming
- **ContentResolver:** Accesses device media storage
- **DrawerLayout and NavigationView:** Provides navigation structure
- **ViewPager and TabLayout:** Organizes content into swiping tabs
- **RecyclerView with custom adapters:** Displays song lists efficiently

5 Implementation

5.1 Core Components

The implementation includes several key components:

5.1.1 Activity Classes

- **MainActivity:** Central hub handling playback controls, navigation, and fragment management
- **SplashActivity:** Entry point with initialization logic

5.1.2 Fragment Classes

- **AllSongFragment:** Displays all available songs
- **FavSongFragment:** Shows user's favorite songs
- **CurrentSongFragment:** Displays current playlist
- **PlaylistFragment:** Manages playlist creation and viewing

5.1.3 Database Handlers

- **FavoritesDBHandler:** Schema definition for favorites
- **FavoritesOperations:** CRUD operations for favorites
- **PlaylistDBHandler:** Schema definition for playlists
- **PlaylistOperations:** CRUD operations for playlists

5.1.4 Model Classes

- **SongsList:** Represents individual songs with metadata
- **Playlist:** Encapsulates playlist information

5.1.5 Adapters

- **SongAdapter:** Binds song data to list views
- **PlaylistAdapter:** Handles playlist display
- **ViewPagerAdapter:** Manages fragment navigation

5.2 Challenges and Solutions

Several implementation challenges were addressed:

5.2.1 Media Permission Handling

- **Challenge:** Android's permission model restricts media access
- **Solution:** Implemented runtime permission requests with fallback handling

5.2.2 Playback Continuity

- **Challenge:** Maintaining playback across navigation
- **Solution:** Centralized MediaPlayer in MainActivity with state management

5.2.3 Efficient Media Scanning

- **Challenge:** Loading large music libraries efficiently
- **Solution:** Implemented background scanning with UI feedback

5.2.4 Database Integration

- **Challenge:** Ensuring consistent database operations
- **Solution:** Created centralized database operations classes with transaction support

6 Testing

6.1 Testing Methodology

The application was tested using multiple approaches:

- **Unit Testing:** Individual components tested in isolation
- **Integration Testing:** Verifying component interactions
- **Manual Testing:** Real-world usage scenarios
- **Performance Testing:** Measuring responsiveness and resource usage

6.2 Test Results

Testing revealed mostly positive results:

- Core functionality performed as expected across different Android versions
- Database operations maintained data integrity
- UI responsiveness met requirements even with large music libraries
- Memory usage remained within acceptable parameters

6.3 Issues and Resolutions

Some issues were identified and addressed:

- MediaPlayer state inconsistencies resolved with improved lifecycle management
- Database query optimization improved loading times for large playlists
- UI layout issues on various screen sizes corrected with constraint-based layouts
- Permission handling edge cases addressed with more robust error handling

7 Results and Discussion

7.1 Achievements

The project successfully delivered:

- A fully functional music player with core playback controls
- Comprehensive playlist management capabilities
- Favorites system for quick access to preferred songs
- Search functionality for finding specific tracks
- Intuitive Material Design interface
- Offline functionality without internet requirements

7.2 Objective Fulfillment

All defined objectives were successfully met:

1. The application provides complete functionality for local music playback
2. The playlist system allows for flexible organization of music
3. The user interface follows Material Design guidelines for intuitive use
4. Media handling is efficient with smooth playback transition
5. The architecture is modular and maintainable for future enhancement

7.3 Comparative Analysis

Compared to existing solutions, this application offers:

- Lower resource utilization than commercial streaming apps
- More streamlined interface than feature-heavy alternatives
- Better playlist management than basic media players
- Complete offline functionality without account requirements

8 Conclusion

8.1 Summary

The Android Music Player project successfully delivers a feature-rich yet lightweight music playback solution. By focusing on core functionality and user experience, the application provides practical value while maintaining simplicity. The implementation demonstrates effective use of Android framework components and follows best practices in mobile application development.

8.2 Key Insights

Throughout development, several valuable insights were gained:

- The importance of efficient media handling for responsive user experience
- The value of consistent error handling for robust application behavior
- The significance of user-centered design in music consumption applications
- The effectiveness of modular architecture for maintainable development

8.3 Future Enhancements

Potential future improvements include:

- Implementing audio visualization features
- Adding basic audio editing capabilities
- Introducing cloud backup for playlists
- Implementing audio equalizer functionality
- Supporting additional media formats
- Adding theme customization options

9 References

1. Android Developers. (2022). MediaPlayer Overview. https://developer.android.com/guide/topics/media/mediaplayer
2. Android Developers. (2022). Fragments. https://developer.android.com/guide/fragments
3. Android Developers. (2022). Save data in a local database using Room. https://developer.android.com/training/data-storage/room
4. Material Design. (2022). Material Design Guidelines. https://material.io/design
5. SQLite. (2022). Documentation. https://www.sqlite.org/docs.html

10 Appendices

10.1 Technical Documentation

The project architecture follows standard Android application patterns with specific implementation details described in the implementation section.

10.2 Code Repository

The application source code is maintained in a version control system with organized package structure:

- `com.example.soc_macmini_15.musicplayer.Activity`: Core activities
- `com.example.soc_macmini_15.musicplayer.Fragments`: UI fragments
- `com.example.soc_macmini_15.musicplayer.Adapter`: Data binding adapters
- `com.example.soc_macmini_15.musicplayer.Model`: Data models
- `com.example.soc_macmini_15.musicplayer.DB`: Database operations

10.3 User Manual

The application is designed to be intuitive, with a tab-based interface for navigation:

- **All Songs:** View and play all available music
- **Favorites:** Access marked favorite songs
- **Playlists:** Create and manage custom playlists
- **Control Panel:** Provides playback controls and current song information
- **Navigation Drawer:** Access additional options and information

The music player requires storage permission to access local media files and provides standard playback controls including play/pause, next, previous, shuffle, and repeat options.