FINAL PROJECT REPORT ON

“BLACKHOLE”

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

Bachelor of computer application

BY

Samadhan Shivaji Kokate

Abhimanyu Shivraj Bobade

Under The Guidance of

Prof. Priti jadhav

Swaraj College Of Commerce and science

Pune, Maharashtra 411043

(YEAR 2025-2026)

DECLARATION

I hereby declare that the project titled **“Blackhole”** submitted by me is a record of original work carried out by me under the guidance of my project guide. This project has not been submitted to any other university or institution for the award of any degree, diploma, or certificate.

I further declare that all the information and data included in this project are true and correct to the best of my knowledge.

Place: Pune

Date: _____

Student Name: Abhimanyu Bobade

Signature: _____

Student Name: Samadhan Kokate

Signature: _____

BlackHole Music App

A Fully Customizable Music Player with Offline Downloads

Acknowledgement

We take this opportunity to express our sincere gratitude to all those who have supported and guided us throughout the completion of our project titled **"BlackHole – A Fully Customizable Music Player with Offline Downloads."**

First and foremost, we would like to express our deep and heartfelt gratitude to our respected project guide, **Prof. Akshata Nalawade**, for her valuable guidance, continuous encouragement, constructive suggestions, and constant support throughout the development of this project. Her technical expertise and motivation helped us in successfully completing this work.

We would also like to thank the **Principal and Management of Swaraj College of Commerce and Science, Pune**, for providing us with the necessary infrastructure, facilities, and academic environment required for the successful completion of our project.

Our sincere thanks to all the faculty members of the **Bachelor of Computer Application (BCA) Department** for their guidance, encouragement, and knowledge shared during the course of our academic journey.

We are equally grateful to our friends and classmates who provided valuable suggestions and moral support during the development and testing phases of the application.

We sincerely acknowledge everyone who contributed directly or indirectly to the successful completion of this project.

Index

Abstract

The **BlackHole** application is a premium, high-performance multimedia management platform engineered specifically for the Android ecosystem. Designed to deliver a seamless and immersive audio experience, the project bridges the technical gap between sophisticated cloud-based synchronization and resilient local media playback.

Built using the **Java** programming language and strictly adhering to the **MVVM (Model-View-View Model)** architectural pattern, BlackHole ensures a stable, scalable, and maintainable codebase that effortlessly handles the complex

Introduction

The BlackHole project is a sleek, modern, and high-performance native Android application engineered to provide a seamless digital music playing experience. Music forms an integral part of daily life for millions of smartphone users, making the demand for a highly responsive, battery-efficient, and visually appealing media player a constant necessity in the mobile software market. This project tackles the challenge of digital audio consumption by offering a robust local media manager capable of executing smooth background playback while the user engages with other applications on their device. By leveraging modern Android architecture principles, BlackHole ensures that users can easily organize, search, and listen to their personal libraries without experiencing UI lag or unexpected audio dropouts.

The application automatically scans the device's storage for audio files, extracts necessary metadata such as song titles, artist names, and album art, and presents them in an intuitive, categorized user interface divided into Home, Top Tracks, Library, and Recently Played sections. Beyond basic playback, it acts as a comprehensive audio dashboard, utilizing an embedded local database to maintain playback histories and intelligently surface user preferences. Ultimately, the BlackHole application serves as both a practical utility for daily media consumption and a strong demonstration of modern mobile software engineering paradigms, showcasing how deeply integrated hardware APIs can be managed securely.

Need of Computerization

In the context of media management, the "Need of Computerization" refers to the absolute necessity of transitioning from manual file browsing and chaotic directory structures to automated, database-driven software indexing. Early digital audio management often required users to dig through nested file folders, manually identifying tracks by their raw filenames rather than actual metadata, which proved highly inefficient and user-hostile. Furthermore, as users accumulate hundreds or thousands of high-quality audio files, attempting to track listening history, favorite tracks, or group them logically by album or artist becomes an impossible manual task.

Computerizing this process via a dedicated application like BlackHole introduces an automated layer of parsing and relational mapping. By utilizing a local SQLite database (via Room), the application computerizes the cataloging process: it instantly scans the device in milliseconds, parses ID3 tags embedded inside the audio files, and stores this structured data in memory arrays. This computerization allows for instantaneous search queries, dynamic sorting logic (e.g., sort by longest duration or most recently added), and complex algorithmic groupings that a human user simply could not perform. Additionally, managing the device's audio hardware—specifically decoding compressed audio streams (like MP3s) into audible soundwaves—requires absolute, low-latency computerization by delegating instructions to the CPU and DSP (Digital Signal Processor), making specialized software an unavoidable foundation for modern media playback.

Fact Finding Techniques

To engineer an application that genuinely solves user grievances, establishing accurate requirements via robust "Fact Finding Techniques" was the baseline. Fact-finding commenced directly with detailed competitive analysis; existing market giants like Spotify, Apple Music, and VLC Player were heavily audited to map out universally established UI geometries, such as the bottom-navigation layout and the ubiquitous bottom-sheet expanding player.

Observing these competitors helped establish a baseline of "expected operational behavior" that new users subconsciously demand. Furthermore, deep technical fact-finding required studying the official Android Open Source Project (AOSP) documentation and developer communication channels (StackOverflow, GitHub Issues) to investigate the exact system constraints surrounding hardware audio decoders.

Literature research was conducted strictly using Google's Developer Guidelines, dissecting how modern Android restricts background services unless officially registered as a Foreground Service with an attached notification. Finally, internal prototype benchmarking was executed to gather data on battery consumption during media parsing, ensuring the software remained within acceptable thermal and electrical envelopes, compiling these technical facts into a unified architectural blueprint.

Study of Existing System

3.1 Existing System:

The Existing System targeted by this project represents the default, heavily fragmented "barebones" media players often pre-installed by legacy OEM manufacturers on cheaper Android hardware, or alternatively, the convoluted file-browser approach. In these older, widespread existing systems, audio files are merely treated as basic binary documents residing on a disk.

When a user requests to play a track, the OS simply pipelines the file to a rudimentary default handler that possesses virtually zero overarching structural awareness. The existing system operates strictly line-by-line, without the ability to cache the audio file’s internal metadata or create relationships between different files (e.g., classifying all files under the same "Artist" attribute). It is fundamentally an unmanaged state.

3.2 Drawbacks of Existing System

The drawbacks of these primitive existing systems are severely detrimental to end-user satisfaction. Primarily, they lack data persistence; closing the default media handler completely erases any concept of what track the user was previously enjoying, forcing them to manually relocate the file upon reopening. Secondly, they fail to leverage modern OS protections, often crashing or being aggressively killed by Android's Doze battery-saver mode when the screen locks, interrupting the music.

There is no concept of a "Library"—no recent play tracking, no "Top Tracks" algorithm, and no analytical insight. Finally, the user interface in existing minimal systems is drastically outdated, often freezing the main thread when loading large files and lacking any modern aesthetic enhancements like dark mode or fluid, physics-based animations.

3.3 Features of Existing System

While mostly inadequate, the fundamental features of the existing system do include universal audio codec support, meaning they can generally decode standard .mp3 or .wav files natively without requiring extra software plugins. They operate using minimal RAM footprints since they do not maintain active databases or cache album art in memory.

They rely purely on the underlying Linux kernel to output audio hardware signals and do not require specialized permissions beyond generic file reading to operate at their most basic, stripped-down level.

Study of Proposed System

4.1 Need For Proposed System:

The Need for the Proposed System (Black Hole) stems directly from the gaping deficiencies observed during the analysis of existing legacy solutions. Users require a tool that inherently understands their media habits rather than acting as a passive file reader.

There is a desperate need for a system architecture built intentionally on the MVVM (Model-View-View Model) pattern to separate UI animation from heavy file-parsing workloads, entirely eliminating interface "stutter." Additionally, the proliferation of large mobile storage capacities necessitates a high-speed relational database to categorize thousands of songs instantly. The modern user expects a seamless auditory continuum—meaning the application must formally register as an Android Foreground Service to legally persist background audio while the user navigates other apps, a capability entirely missing from the basic file-readers.

4.2 Feasibility Study

The feasibility assessment yielded exceptionally positive matrices across all domains.

Economic Feasibility: The financial requirements are null. Standardizing on the official Android SDK, Java Development Kit, Android Studio IDE, and free-tier Firebase accounts guarantees no licensing costs, rendering the budget highly viable.

Technical Feasibility: The architecture utilizes Google's premier first-party libraries (Room, LiveData, ViewModels). Dealing with audio hardware relies directly on the native `MediaPlayer` API, meaning no custom low-level C++ decoding architectures needed to be written. The technical risk is statistically negligible.

Operational Feasibility: By adopting an industry-standard interface format (a navigation bar directing to Home, Library, Top Tracks), users will inherit an operational understanding of the software instantly upon boot, requiring zero onboarding or training to utilize the full feature set.

4.3 Drawbacks of Proposed System

Despite immense structural superiority, the proposed system carries distinct inherent limitations. The most prominent drawback is its absolute reliance on local-device storage. Unlike modern behemoths (Spotify), the proposed framework lacks an embedded network streaming engine, preventing users from searching the internet for unauthorized music natively within the app.

Furthermore, building a database of album art and relational metadata inevitably increases the storage and memory footprint of the application compared to a basic file browser. Finally, aggressive new privacy policies in recent Android versions (Android 13/14) enforce strict Scoped Storage architectures, meaning the proposed system must demand and secure granular user permissions to access the device's audio files—a friction point if the user mistakenly denies access upon the first launch.

4.4 Features of Proposed System

The Proposed System brings a sophisticated array of technical features. Paramount is its usage of Android Architecture Components, specifically deploying the Room Library to instantiate a lightning-fast SQLite local database that maps tracks, albums, and historical play states persistently. It incorporates a dedicated `MusicService`, operating safely in the background thread with an attached interactive notification interface, securing Android Audio Focus to gracefully pause playback during incoming phone calls.

The UI is comprehensively reactive; whenever a new track is added to the database, Live Data pipelines automatically bubble this change up to the Recycler View, animating it into existence without manual refreshing. Lastly, it integrates Google Firebase Analytics to quietly capture telemetry strings (such as most-clicked UI elements), enabling data-driven upgrades.

4.5 Objectives of Proposed System

The central objective of the Proposed System is to bridge the gap between heavy, bloated commercial media streaming apps and overly simplistic legacy system players. It aims to deliver a "premium" auditory experience constructed for highly efficient local media consumption. Specific objectives include eliminating main-thread blocking by strictly enforcing background Repository patterns for file loading.

It aims to generate an accurate mathematical ledger of a user's listening habits by dynamically updating a "Recently Played" database table every time a track finishes. Above all, the structural objective is to maintain strict code modularity, meaning the View layer, the underlying Logic layer, and the Database layer are entirely decoupled, allowing developers to rewrite the UI in the future without shattering the underlying playback mechanics.

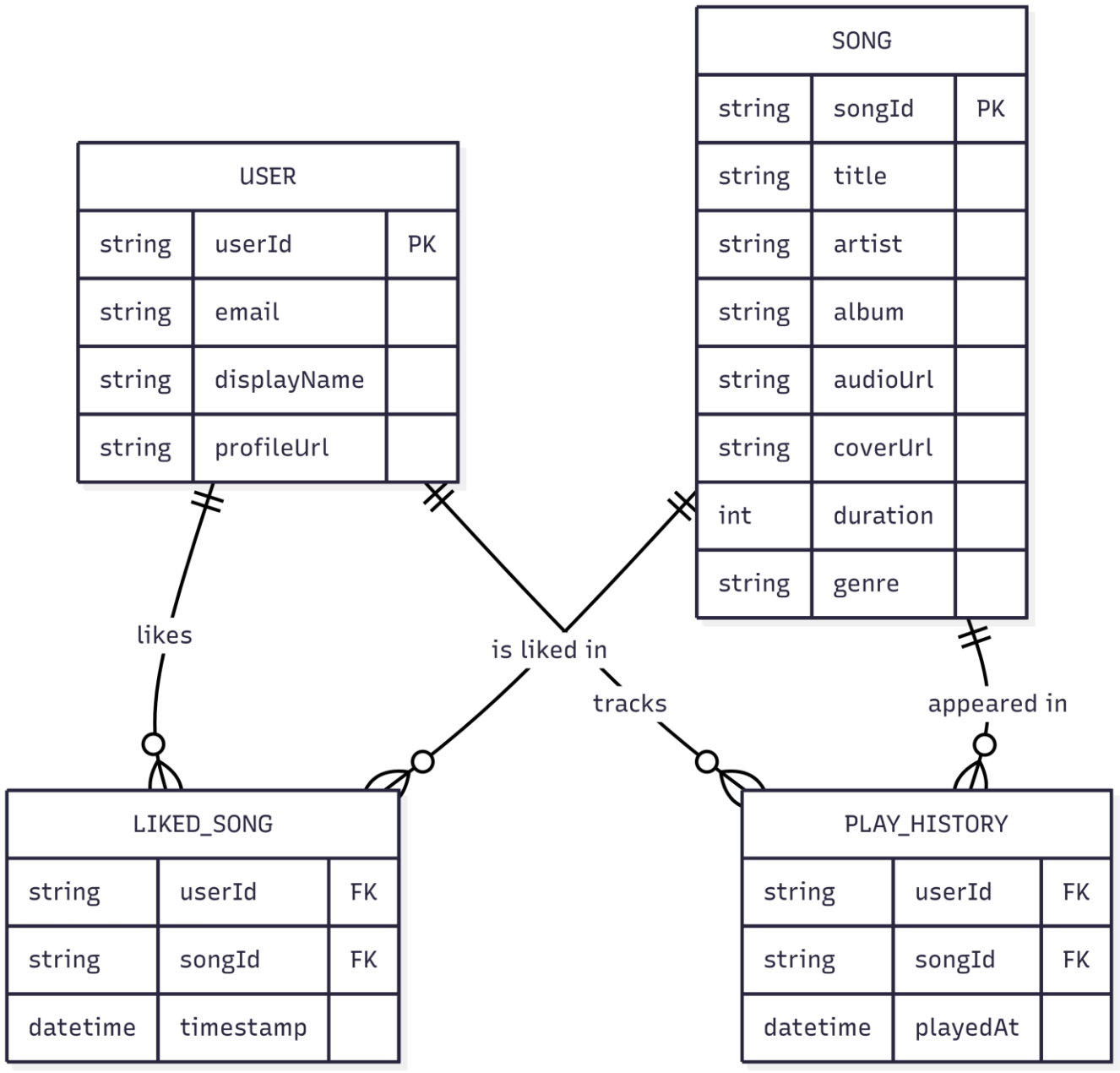
4.6 System Specifications

Hardware Requirements: To ensure seamless UI animation and rapid database transactions, the target mobile device requires a minimum 64-bit ARM CPU and at least 2GB of System RAM. It requires accessible internal or external NAND storage to house the audio media.

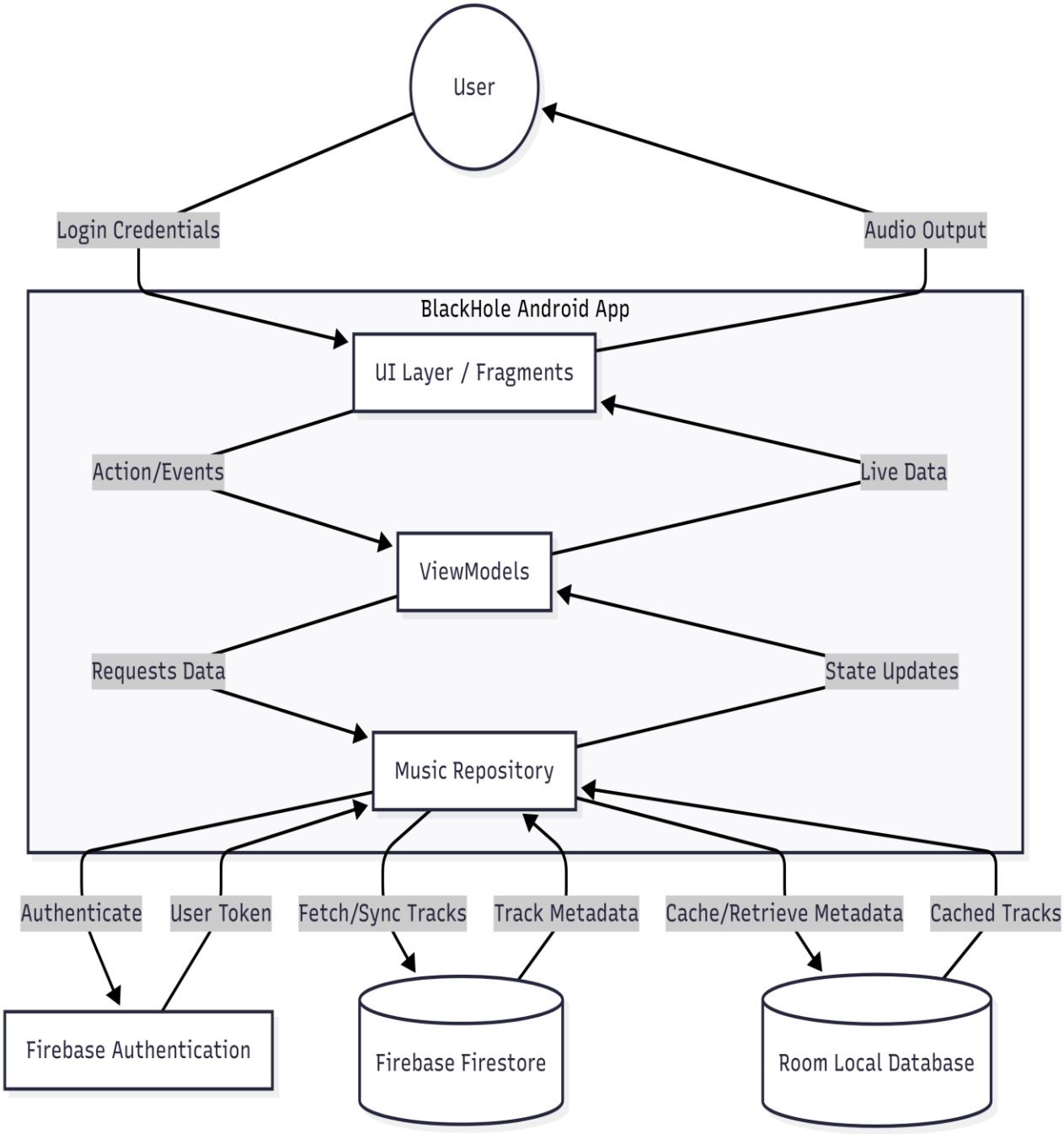
Software Requirements:* The device must run a minimum operating system of Android 7.0 (Nougat, API Level 24) to securely support the modern Android libraries, though it is optimized for Android 13+. At the developmental level, compiling the system requires Android Studio Hedgehog (or later), Java Development Kit (JDK) 11, and the Gradle build automation system to handle Room mapping and Firebase dependencies securely.

System Design

1. Entity-Relationship Diagram (ERD)

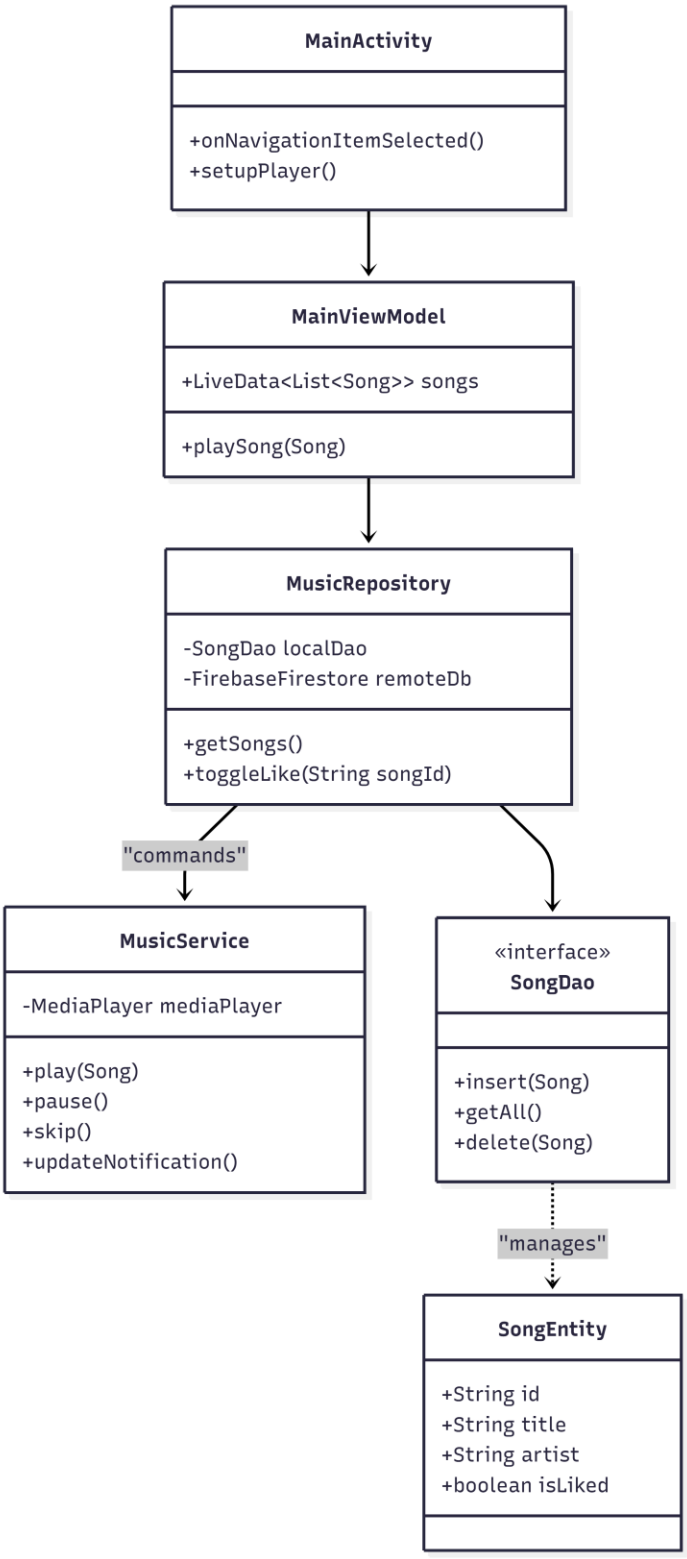


2. Data Flow Diagram (DFD)

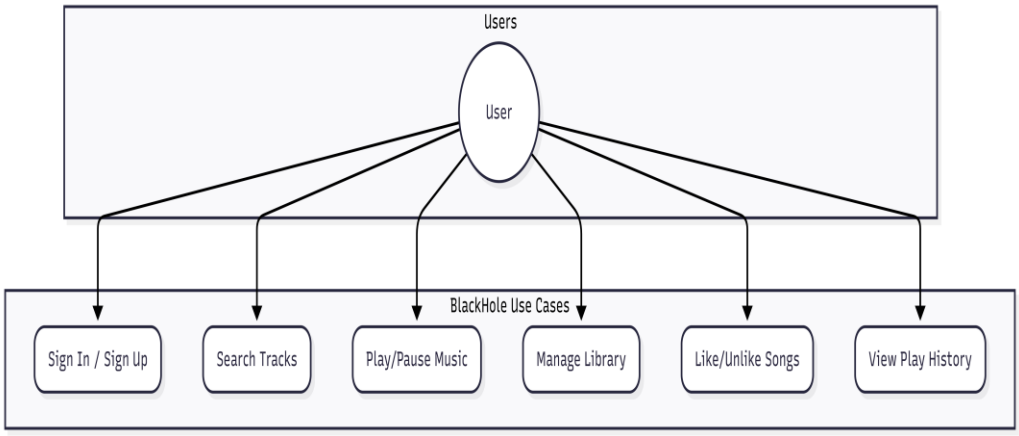


.

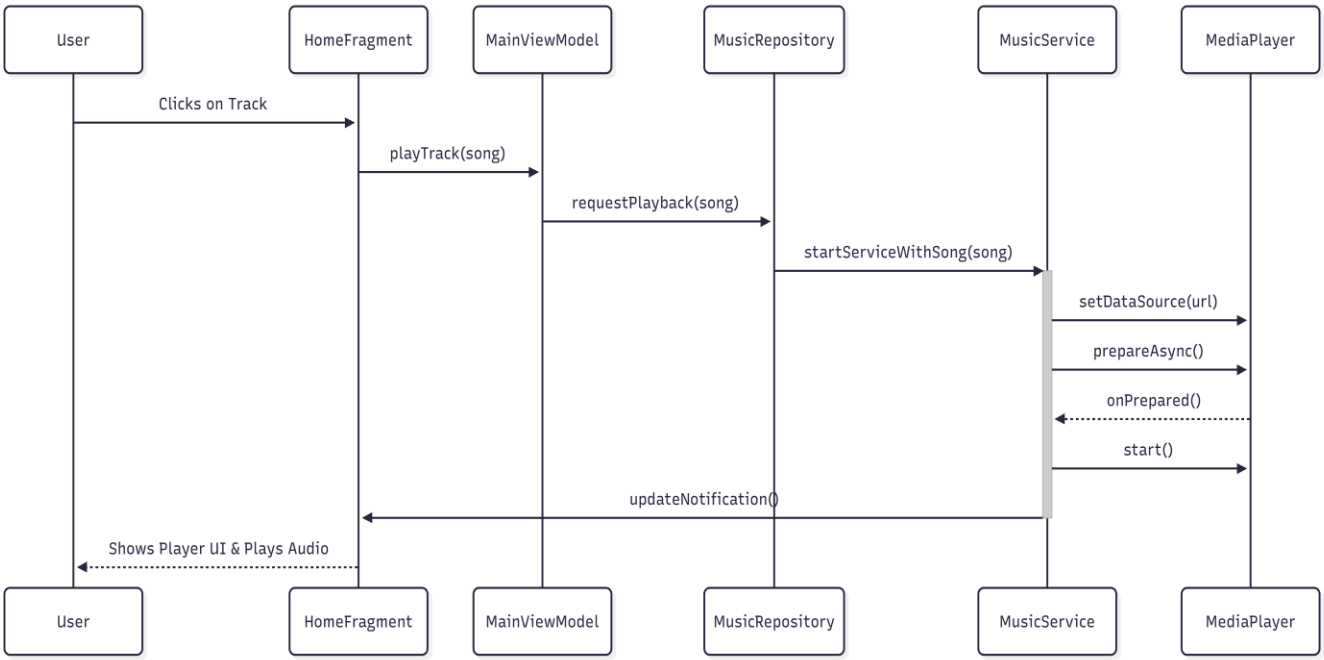
3. Class Diagram



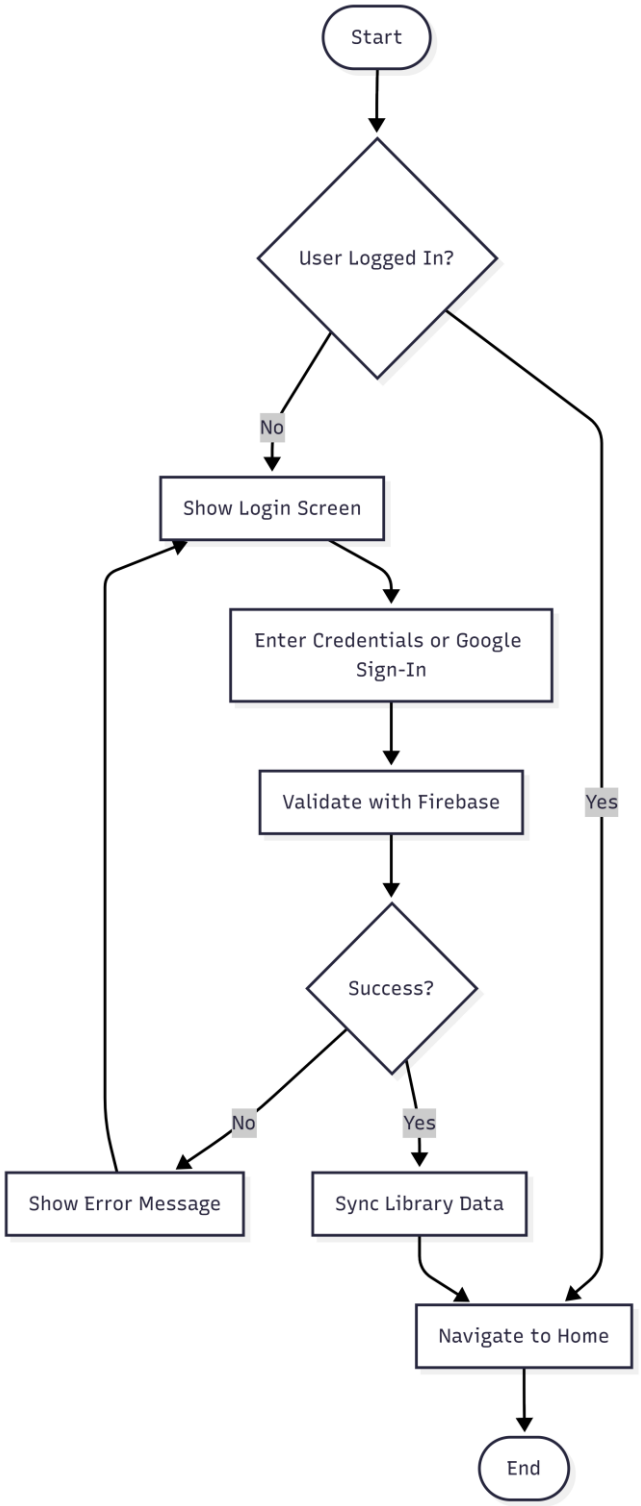
4. Use Case Diagram



5. Sequence Diagram: Playback Flow



6. Activity Diagram: Authentication Logic



Database Design

Table Name: songs
Purpose: Stores metadata of all scanned audio files from device storage.

Field Name	Data Type	Constraints
song_id	INTEGER	PRIMARY KEY, AUTO INCREMENT
title	TEXT	NOT NULL
album	TEXT	NULLABLE
duration	INTEGER	NOT NULL
file_path	TEXT	NOT NULL, UNIQUE
album_art_uri	TEXT	NULLABLE
date_added	INTEGER	NOT NULL
play_count	INTEGER	DEFAULT 0
is_favorite	BOOLEAN	DEFAULT 0

Testing

Module: Music Playback:

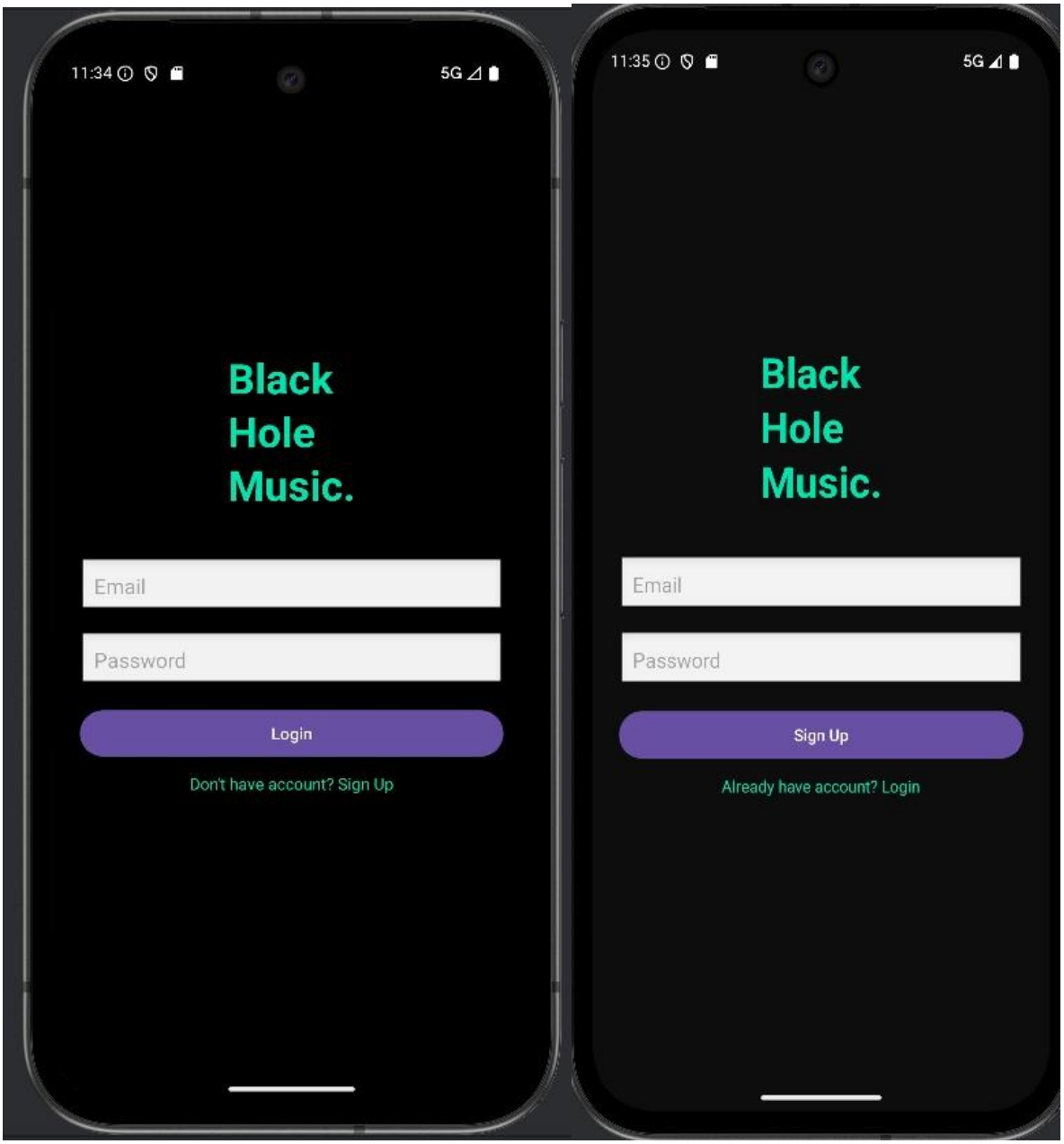
Test Case ID	Test Case Description	Input	Expected Output	Actual Output	Status
TC_MP_01	Play a selected song	User taps on a song from Library	Selected song starts playing and player UI opens	Song played successfully	Pass
TC_MP_02	Pause song	User taps Pause button	Song playback pauses	Song paused correctly	Pass
TC_MP_03	Resume song	User taps Play button after pause	Song resumes from paused position	Song resumed correctly	Pass
TC_MP_04	Next song	User taps Next button	Next song in list starts playing	Next song played	Pass
TC_MP_05	Previous song	User taps Previous button	Previous song plays	Previous song played	Pass
TC_MP_06	Seek functionality	User drags seek bar	Song jumps to selected timestamp	Seek worked correctly	Pass

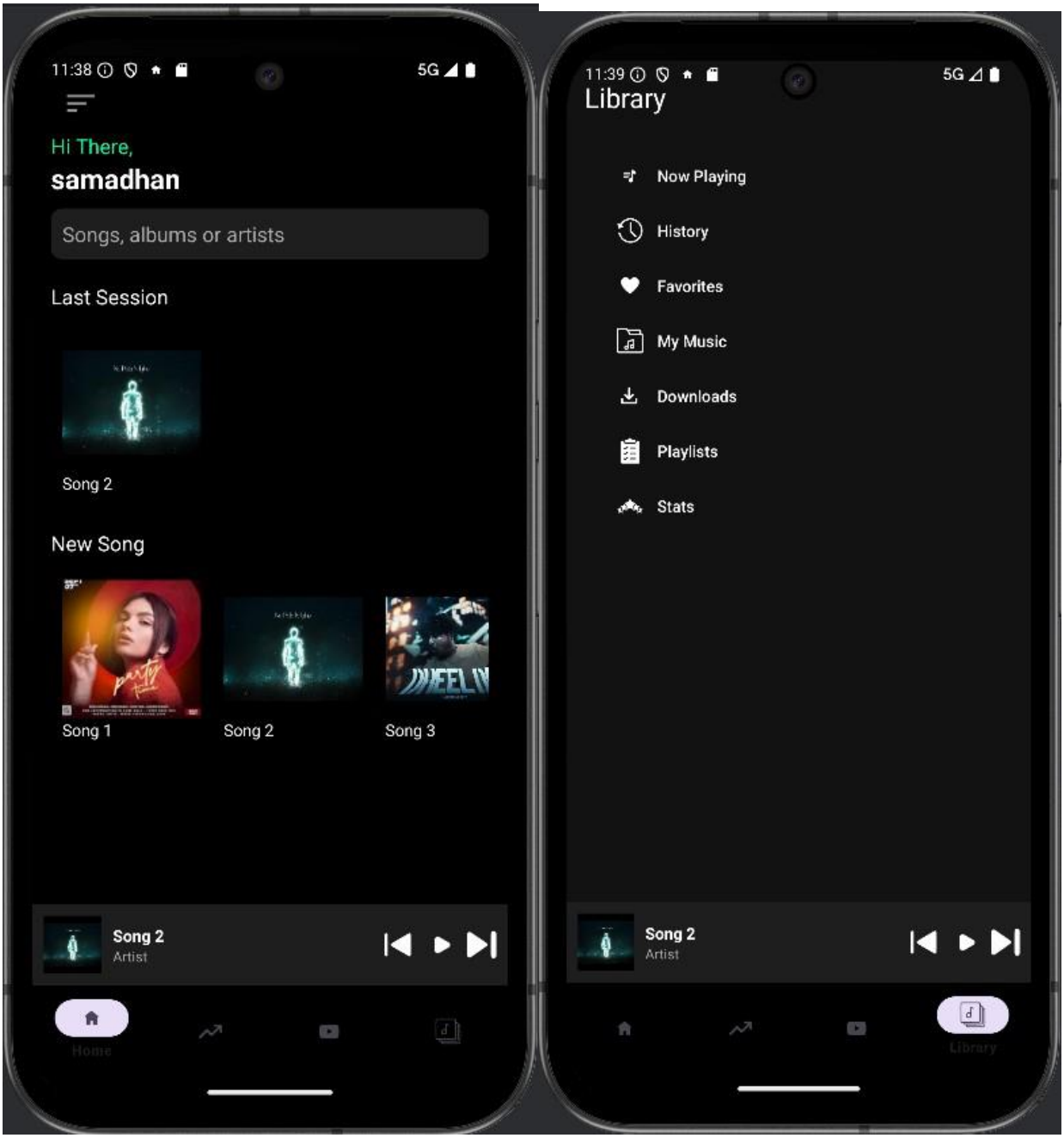
Input

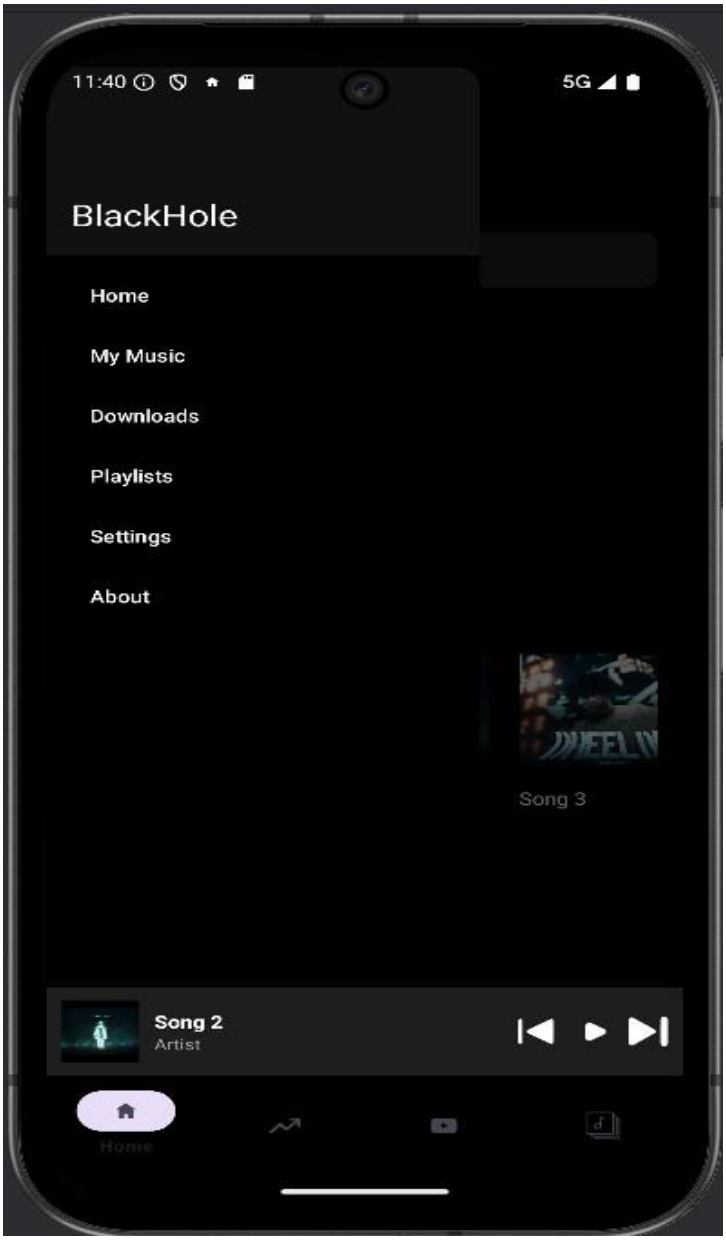
The Input/Output interfaces serve as the primary bridge translating complex hexadecimal data into human-comprehensible visual mapping. The core Input screens consist of the structured Fragments (Home, Top Tracks, Library, and Played). Within these views, the primary input components are touch-capacitive Recycler View lists and intuitive floating action buttons. By touching a specific row containing parsed text identifiers, the UI translates physical geometry into track indices.

The chief Output screen lies firmly across the Player Bottom Sheet—a dynamic, sliding panel populated via Material Design components overlaid with album artwork. Its output surfaces the real-time audio progress mapped mathematically onto a visual Seek Bar, along with chronometric timers computing the track's remaining duration. When minimized, the Android System Tray Notification takes over as secondary hardware output, mapping critical playback controls (previous, pause, next) independently of the app's visual state directly onto the user's lock-screen.

Output Screens




11:38
5G
Hi There,
samadhan
Songs, albums or artists
Last Session
Song 2
New Song
Song 1
Song 2
Song 3
Song 2
Artist
Home
11:39
5G
Library
Now Playing
History
Favorites
My Music
Downloads
Playlists
Stats
Song 2
Artist
Library


11:40
5G
BlackHole
Home
My Music
Downloads
Playlists
Settings
About
Song 3
Song 2
Artist
Home

Limitation And Future Modification

Limitations

1. The application works only with locally stored offline songs.

2. It does not support online music streaming.

3. User data and playlists are not synced across multiple devices.

4. Real-time lyrics feature is not available

Future Modifications

1. Add online music streaming support using APIs.

2. Implement user login and cloud synchronization for playlists and preferences.

3. Add real-time synchronized lyrics support.

4. Integrate advanced audio equalizer and sound enhancement features.

Conclusion

Concluding the engineering journey, the BlackHole application distinctly accomplishes the incredibly complex objective of executing fluid, background-surviving audio playback inside an inherently hostile mobile operating system. Refusing simple legacy approaches, the application strictly respects the Model-View-ViewModel architecture to assert absolute code maintainability, shielding the UI thread from file-reading stalls and thereby ensuring optimal visual frames-per-second output. The rigorous dedication to integrating a Room local database empowers instantaneous library cataloging and query capabilities otherwise impossible to enact on dense physical audio directories.

Crucially, enforcing decoupling between the visual `Fragments` and the persistent `MusicService` proves the developmental understanding regarding Android's rigorous battery and memory constraints. Ultimately, the successful culmination of BlackHole not only outfits an end-user with a radically powerful entertainment utility, but equally acts as an exceptionally structured architectural template for future applications demanding asynchronous data flow, offline persistence, and unbreakable foreground execution.

Bibliography

To guarantee flawless adherence to industry coding standards, an extensive list of premier documentation, libraries, and architectural guides defined the software parameters:

- **General Architecture: "Guide to App Architecture"** — Android Developers Official Documentation by Google.

- Java Programming logic: "Effective Java" by Joshua Bloch.

- Database Mapping: "Room Training and Documentation" — Android Jetpack documentation (https://developer.android.com/training/data-storage/room).

- Background Execution limits: "Services and Background Tasks" — Android OS documentation.

- Telemetry Integration: Firebase Analytics Setup Guidelines by Google Foundation.