Groove.flac - Music Player

A MINI PROJECT REPORT

Submitted by

NISHAL I P 2116220701187

in partial fulfillment for the course

CS19611 – MOBILE APPLICATION DEVELOPMENT LABORATORY

of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR

THANDALAM

CHENNAI – 602 105

MAY 2025

BONAFIDE CERTIFICATE

Certified that this report titled **"GROOVE.FLAC - MUSIC PLAYER"** is the bonafide work of **NISHAL I P (2116220701187)** who carried out the project work (CS19611-Mobile Application Development Laboratory) under my supervision .

SIGNATURE

Dr. P. Kumar M.E., Ph.D.
HEAD OF THE DEPARTMENT

Professor and Head
Department of Computer Science and Engineering
Rajalakshmi Engineering College,
Chennai – 602105

SIGNATURE

Mr. Bhuvaneswaran B, M.E.
SUPERVISOR

Assistant Professor (SG)
Department of Computer Science and Engineering
Rajalakshmi Engineering College,
Chennai – 602105

Submitted to Project Viva-Voce Examination held on _____

Internal Examiner

External Examiner

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavour to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavouring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our working time. We express our sincere thanks to **Dr.P.KUMAR, Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Mr. Bhuvaneswaran B**, **M.E.,** Department of Computer Science and Engineering. Rajalakshmi Engineering College for his valuable guidance throughout the course of the project.

NISHAL I P 220701187

ABSTRACT

Groove.FLAC is a modern, offline-first local music player for Android that blends powerful functionality with aesthetic appeal, following Material You design principles. Built using Kotlin and Android Studio, the app offers users a customizable and immersive audio experience with features typically found only in premium apps. Unlike traditional players, Groove.FLAC focuses on user freedom, allowing renaming of tracks, changing cover art, and adjusting playback speed, all while delivering smooth performance even on low-end devices.

One of the key highlights of Groove.FLAC is its suite of utility features that enhance the user experience. These include an inbuilt 5-band equalizer for real-time audio tuning, a sleep timer to auto-stop music after a set period, and a shuffle feature for dynamic playback. The app does not require an internet connection, ensuring that music remains accessible anytime, anywhere. With Material You integration, the UI adapts fluidly to the user’s device theme, providing a visually engaging and intuitive interface.

Designed with scalability in mind, Groove.FLAC is built using a modular architecture, which makes it easy to maintain and extend. Future implementations include integrating Spotify's API to enable online streaming and playlist syncing directly within the app. Groove.FLAC thus aims to bridge the gap between minimal offline music apps and advanced streaming solutions by offering a feature-rich, privacy-conscious, and user-friendly listening platform.

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In today's digital age, music consumption has shifted significantly toward mobile platforms, with users seeking seamless, personalized, and offline-friendly audio experiences. Android, being the most widely used mobile operating system, provides an ideal ecosystem for developing versatile media applications. Groove.FLAC is a feature-rich, local music player developed in Kotlin using Android Studio, designed to give users full control over their offline audio library. The app emphasizes customization, aesthetic design, and offline functionality, catering to users who prefer managing their own music files without relying on streaming platforms.

Groove.FLAC allows users to play locally stored songs with additional capabilities such as renaming tracks and updating cover art directly within the app. Playback speed control, a built-in equalizer, sleep timer, and shuffle functionality enhance the listening experience. Designed with Google's Material You design language, the user interface dynamically adapts to system themes, providing a personalized and visually coherent experience. The app is optimized for performance across a range of devices, ensuring smooth navigation and media playback.

Local data storage and processing are core to Groove.FLAC, ensuring that all functionalities are available even without internet connectivity. The app integrates key Android components such as ExoPlayer for audio playback, MediaMetadataRetriever for media data access, and customizable UI elements via RecyclerView and ConstraintLayout. Groove.FLAC provides users with a lightweight, reliable, and elegant solution for managing and enjoying their personal music collection anytime, anywhere.

1.2 OBJECTIVE

The primary objective of this project is to develop an Android-based local music player that delivers a feature-rich, user-friendly, and offline-first audio experience. Groove.FLAC empowers users to organize and enjoy their personal music libraries with advanced functionality such as renaming songs, changing album art, and controlling playback speed—all without needing internet access. By focusing on user-centric design and performance optimization, the app ensures consistent usability across a wide range of Android devices.

The application is designed to provide both casual listeners and music enthusiasts with essential features that enhance everyday listening. Utilities like a built-in equalizer for audio customization, a sleep timer for scheduled playback termination, and shuffle mode for varied music playback ensure that the app adapts to diverse user needs and listening styles. These tools, combined with a minimal, Material You-inspired UI, encourage users to engage more actively with their music collection in a personalized environment.

By allowing track renaming and cover image updates, Groove.FLAC puts creative control in the hands of the user—making the app ideal for those who wish to maintain or personalize their local audio files. The app emphasizes privacy and independence by storing all data locally, eliminating any need for cloud synchronization. Through its clean architecture and responsive interface, Groove.FLAC aims to transform offline music playback into a smooth, customizable, and aesthetically pleasing experience.

1.3 EXISTING SYSTEM

Currently available music player applications such as VLC Media Player, Poweramp, AIMP, and Musicolet offer a variety of features including equalizers, playlist management, and support for various audio formats. However, many of these apps are either overloaded with advanced features that overwhelm casual users or restrict essential functionalities behind paywalls or premium subscriptions. Additionally, apps like Spotify or YouTube Music rely heavily on internet connectivity and cloud-based streaming, limiting their usefulness for users who prefer local music playback.

For users who seek simplicity, offline usability, and aesthetic customization, these existing solutions often fail to deliver a fully satisfying experience. Many lack intuitive interfaces that align with modern UI standards, do not support on-device editing of song metadata, or exhibit performance issues on mid- to low-end devices. Moreover, the absence of direct features like song renaming or album art replacement limits the user's ability to personalize their music library.

There is thus a noticeable gap in the market for a lightweight, modern, offline music player that prioritizes user control, customization, and performance. Groove.FLAC addresses this gap by combining essential playback functionalities with advanced personalization options, all wrapped in a clean, Material You-inspired interface. It provides an ideal solution for users who prefer managing their own music files locally, without the need for constant internet access or bloated streaming features.

1.4 PROPOSED SYSTEM

The proposed system is a lightweight, feature-rich local music player application named Groove.FLAC, designed for Android devices. Unlike existing music apps that rely heavily on internet connectivity or restrict advanced features behind paywalls, Groove.FLAC is entirely offline and grants users complete control over their local music library. The application enables users to perform essential actions such as playing music files, renaming songs, changing album cover images, adjusting playback speed, setting sleep timers, and utilizing an inbuilt equalizer—all within a responsive and modern user interface.

Groove.FLAC offers a clean and adaptive Material You design that aligns with the latest Android UI standards, delivering a personalized experience by dynamically adjusting colors based on the system theme. All data, including renamed songs and edited metadata, is handled and stored locally, ensuring the user’s privacy and independence from cloud-based services. The equalizer allows for real-time audio adjustments, while the sleep timer automatically stops playback after a predefined duration—ideal for nighttime listening. The shuffle feature adds variety to playback, enhancing user engagement and convenience.

The application is structured using clean architecture principles, emphasizing modularity, maintainability, and performance efficiency. It performs smoothly even on devices with limited hardware resources. Since the app does not require sign-in, internet access, or background sync, it maintains low power usage and minimal user friction. The future scope includes integration with Spotify’s SDK, enabling streaming functionality while preserving the app's core identity as a local-first player. Groove.FLAC thus addresses a significant gap in the current app ecosystem by providing a customizable, offline music player with a focus on usability, personalization, and control.

CHAPTER 2

LITERATURE SURVEY

Mobile applications designed for media playback have undergone significant advancements, especially in terms of improving user experience, interface design, and feature-rich functionality. These apps, which aim to provide seamless media consumption, often integrate features such as customizable settings and adaptive behaviors to optimize user engagement. This survey examines several academic contributions to inform the development of a user-centric, feature-rich local music player app like **Groove.FLAC**.

S. W. Lee, K. M. Choi, and M. Kim (2010) explored a context-aware mobile task management system that adapts its functionality based on user location and context. Similarly, a local music player like **Groove.FLAC** could integrate context-aware features to adjust playback settings, such as dynamically altering the playback speed or suggesting playlists based on factors such as time of day or user location (e.g., workout, commuting).

P. Nurmi et al. (2010) proposed adaptive scheduling for mobile task management systems, emphasizing dynamic adjustments based on user activity. For a music player app, this could mean recommending songs or playlists based on the user's current listening patterns or environment, ensuring that the app provides a personalized experience that evolves with user behavior.

L. Pei et al. (2013) introduced a personalized reminder system using data mining techniques to predict critical tasks. A similar approach could be applied to **Groove.FLAC** to provide personalized notifications for users, such as reminding them to update their playlists, suggesting new music based on listening patterns, or adjusting settings like the equalizer based on past behavior.

T. Okoshi et al. (2014) emphasized reducing notification overload through intelligent

scheduling mechanisms. This concept can be extended to music player apps like **Groove.FLAC**, ensuring that notifications regarding new songs, playlist updates, or system alerts are delivered at appropriate times, avoiding intrusive interruptions while maintaining user engagement.

A. H. Chua and S. L. Goh (2014) developed a task manager with intelligent alert prioritization to avoid overwhelming users. This approach is valuable in **Groove.FLAC**, where notifications regarding song changes, album updates, or new features can be prioritized and spaced out based on the user's activity, ensuring an optimal user experience.

Y. Liu and G. Cao (2016) analyzed how task alerts disrupt users and proposed methods to minimize interruptions. In the context of **Groove.FLAC**, the app could use intelligent scheduling to prevent unnecessary alerts during music playback, allowing users to enjoy their experience without being distracted by frequent notifications.

A. Pathak et al. (2012) studied fine-grained energy accounting in mobile apps, emphasizing the need for efficiency in resource usage. This is particularly relevant to **Groove.FLAC**, as the app must be energy-efficient during music playback, image processing for album art, and storage management for local files, ensuring the app performs optimally while consuming minimal battery power.

H. Song et al. (2016) discussed how intelligent reminders could improve productivity by modeling user behavior. By leveraging behavior modeling, **Groove.FLAC** can predict when users are most likely to listen to music and suggest appropriate playlists, helping users optimize their experience based on their typical listening habits.

M. A. Javed et al. (2018) reviewed task scheduling in mobile cloud computing, providing insights on optimizing mobile applications with a focus on efficient resource management. For **Groove.FLAC**, this research can inform strategies for optimizing local processing, such as managing album art changes, playlist updates, or

in-app media processing, while keeping resource usage to a minimum.

Y. Kwon et al. (2017) suggested that notification timing should be personalized according to user habits. In **Groove.FLAC**, notifications such as updates for new songs, playlist changes, or reminders to check music libraries could be tailored to the user's daily routine, enhancing the app’s ability to engage the user without overloading them.

This literature emphasizes the importance of incorporating adaptive behavior, intelligent alert systems, personalized features, and performance optimization techniques into the design of a local music player app. By integrating these elements, **Groove.FLAC** can provide an efficient, engaging, and user-centric experience, enhanced by intelligent features like context-aware recommendations, adaptive behavior, and personalized notifications. The planned future integration of **Spotify** will further enhance the app's functionality by expanding the music library and providing seamless integration with streaming services.

CHAPTER 3

SYSTEM DESIGN

3.1 GENERAL

The system design of **Groove.FLAC** focuses on providing a sleek, offline-first solution for local music playback on Android, prioritizing user control and a smooth audio experience. The app is built using the **Model-View-ViewModel (MVVM)** architecture to separate concerns, promote maintainability, and ensure a responsive user interface. Core Android components like **Room** (for local data persistence), **LiveData** (for data observation), and **ViewModel** (for lifecycle-aware data handling) are employed to provide a fluid and dynamic experience without sacrificing performance.

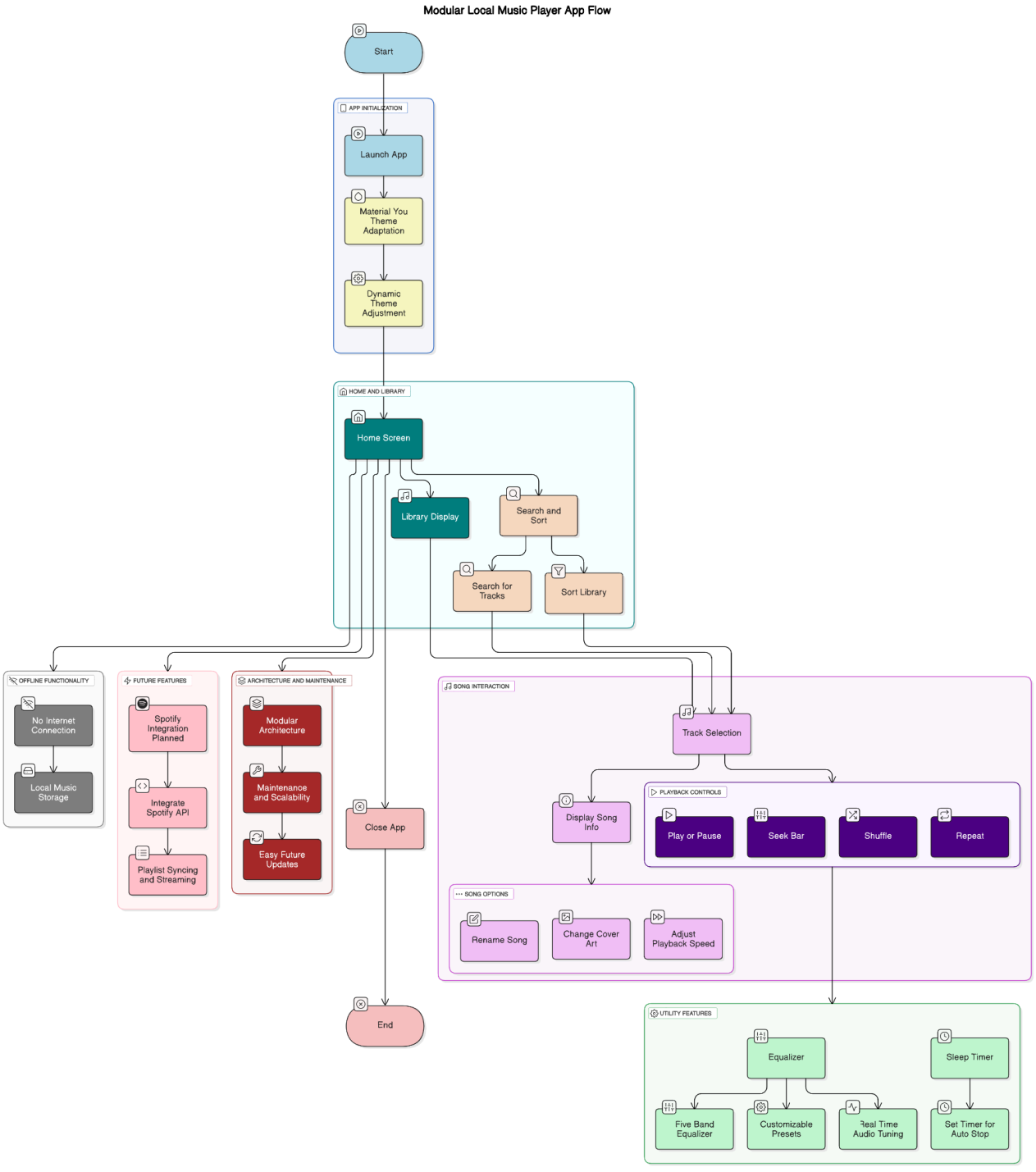
Designed with user convenience and privacy in mind, Groove.FLAC ensures that all music files and metadata are stored locally on the device, eliminating the need for internet connectivity or external servers. This approach guarantees that users have full control over their music library, including customizations like renaming tracks, changing cover art, and adjusting playback settings, all while keeping their data completely private. The app's lightweight nature and efficient design ensure that it operates seamlessly without draining battery or running unnecessary background services.

The user interface follows **Material You** principles, ensuring that the app adapts to the device's theme and provides an aesthetically pleasing and intuitive experience. The app includes features like an inbuilt 5-band equalizer, playback speed adjustments, a sleep timer, and shuffle mode—all integrated in a way that doesn't compromise system resources. These features are optimized to ensure smooth performance even on lower-end devices.

The modular architecture of Groove.FLAC allows easy future scalability and maintenance, with planned integrations like **Spotify API** for streaming and playlist syncing, ensuring the app remains relevant in an ever-evolving landscape of music playback solutions.

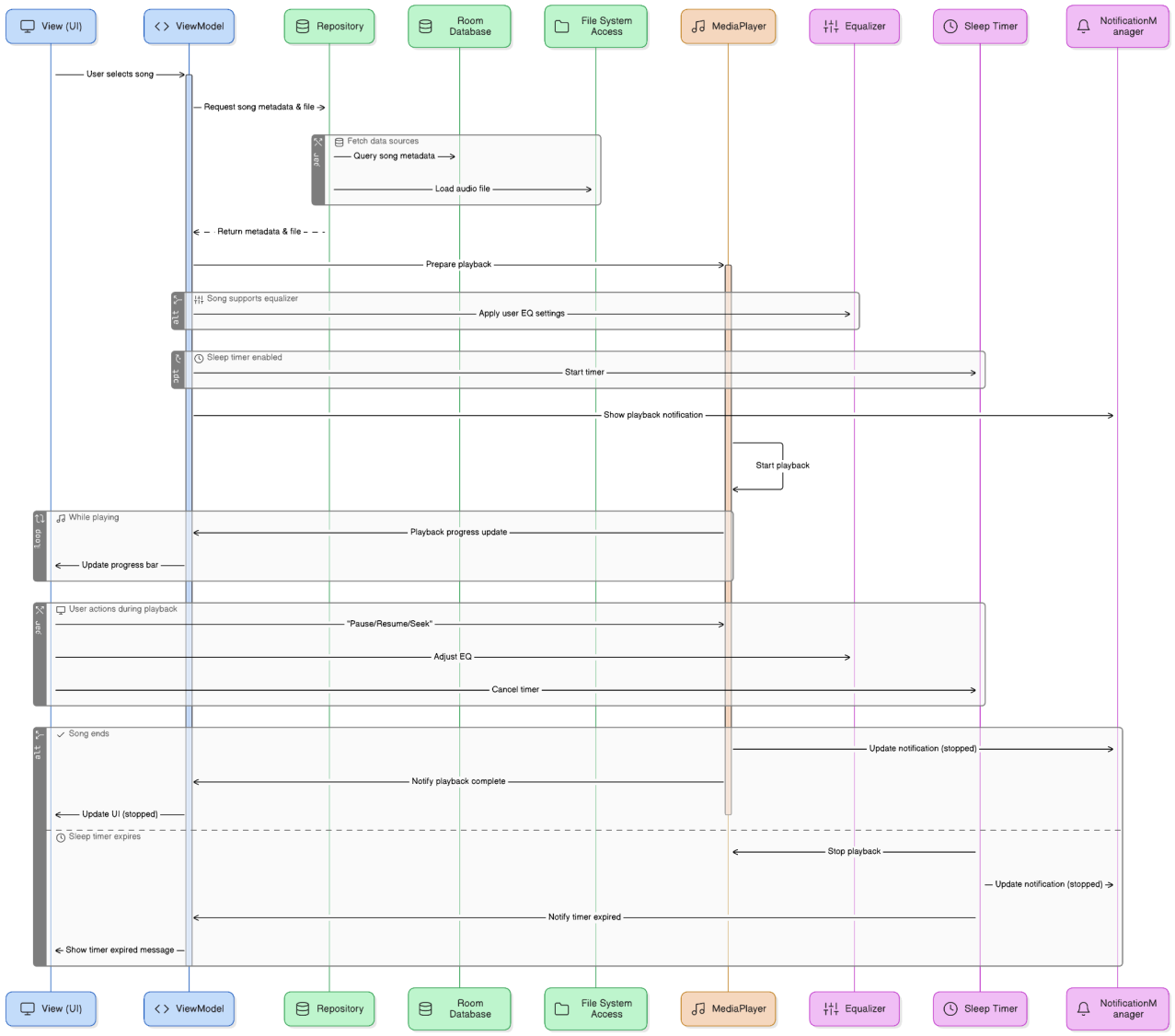
In summary, **Groove.FLAC** delivers a comprehensive, user-focused, and highly customizable offline music player, designed for privacy-conscious users who value simplicity, control, and a seamless listening experience.

3.1.1 SYSTEM FLOW DIAGRAM



3.1.2 ARCHITECTURE DIAGRAM

The app is designed using the **MVVM (Model-View-ViewModel)** architecture to maintain a clean separation of concerns, ensuring the app is maintainable, scalable, and easily testable. Below is an overview of how the architecture is structured for **Groove.FLAC**.



3.1.3 ACTIVITY DIAGRAM

The activity diagram describes the dynamic behavior of the Finance Tracking App. It begins when the user launches the app and is directed to the main screen. From there, the user can perform core actions such as **adding a new transaction**, **viewing existing records**, **filtering transactions by type** (Sent/Received), and **editing or deleting transactions**. When the user taps the **"Add Transaction"** button, the system collects transaction details including amount, type, date, due date (optional), and an image (optional). The data is then saved into the local Room database. If a due date is provided, a reminder is scheduled using **AlarmManager** to notify the user when the date is reached.

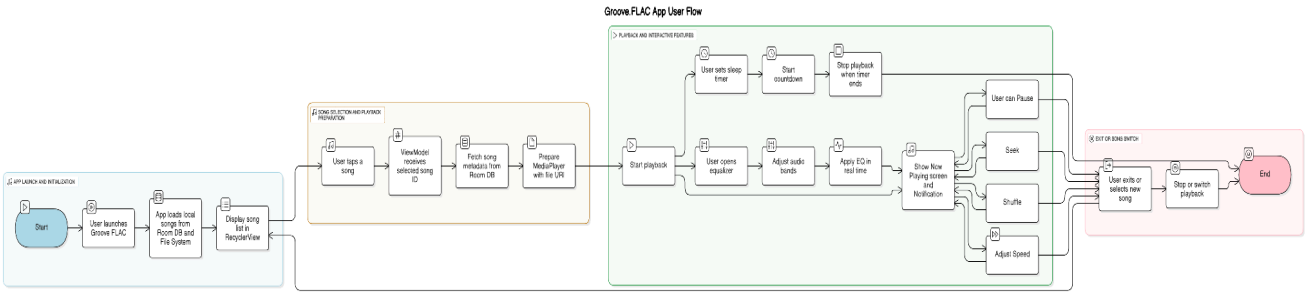


Fig 3.3

3.1.4 SEQUENCE DIAGRAM

The sequence diagram represents the chronological interaction between components of the Finance Tracking App. It begins with the **User** initiating an action—such as adding a new transaction—via the **MainActivity**. The activity forwards the input to the **ViewModel**, which handles the logic and communicateswith the **Repository**. The Repository acts as a single source of truth and interacts with the **Room Database** to perform **CRUD operations** such as insert, update, or delete.

When a transaction includes a **due date**, the ViewModel triggers the **AlarmManager** (or **WorkManager**, if background scheduling is required) to schedule a local reminder. At the scheduled time, the system activates the **Notification Manager** to push a local alert to the user, reminding them of the upcoming payment or receivable. If the transaction is edited or deleted, the system updates or cancels the associated scheduled notification accordingly. This sequence ensures that all financial entries are processed accurately and that reminders are executed reliably in a timely manner.

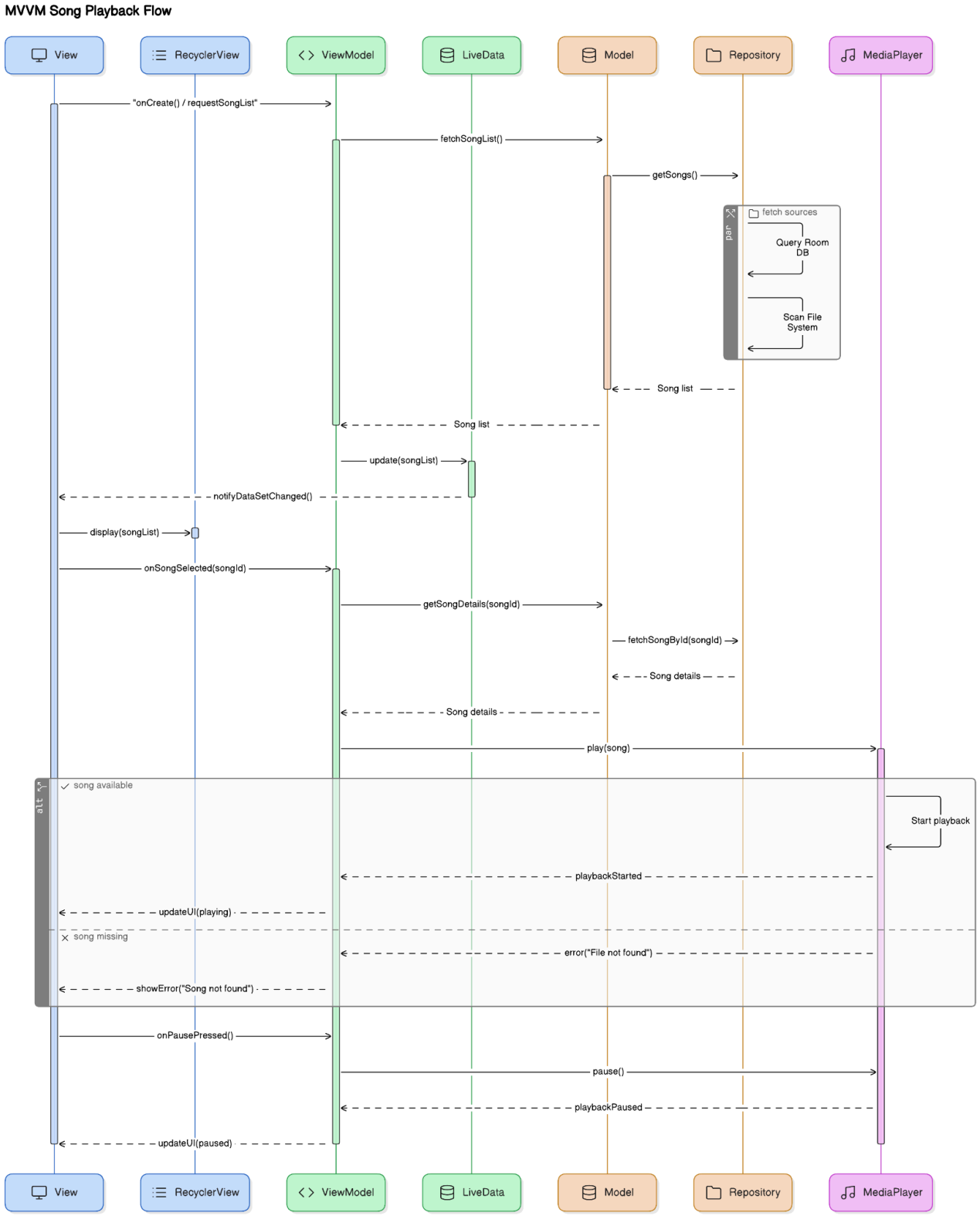

MVVM Song Playback Flow
View
RecyclerView
ViewModel
LiveData
Model
Repository
MediaPlayer
"onCreate() / requestSongList"
fetchSongList()
getSongs()
par
fetch sources
Query Room DB
Scan File System
Song list
Song list
update(songList)
notifyDataSetChanged()
display(songList)
onSongSelected(songId)
getSongDetails(songId)
fetchSongById(songId)
Song details
Song details
play(song)
alt
song available
Start playback
playbackStarted
updateUI(playing)
song missing
error("File not found")
showError("Song not found")
onPausePressed()
pause()
playbackPaused
updateUI(paused)
View
RecyclerView
ViewModel
LiveData
Model
Repository
MediaPlayer

Fig 3.4

CHAPTER 4

PROJECT DESCRIPTION

4.1 INTRODUCTION

Groove.FLAC is an Android mobile application developed using **Kotlin** and **Android Jetpack** libraries. It is a modern, offline-first local music player designed to provide users with full control over their music collection without requiring internet connectivity. Unlike typical streaming services, Groove.FLAC focuses on **privacy**, **customization**, and **performance**, enabling features such as **renaming tracks**, **changing cover art**, **adjusting playback speed**, and managing songs stored on the device.

The app follows the **MVVM (Model-View-ViewModel)** architecture pattern and utilizes **Room Database** for organizing and persisting metadata about music files. Core playback functionality is powered by **MediaPlayer** (or **ExoPlayer** in future updates), while utility features like a **5-band equalizer**, **sleep timer**, and **shuffle mode** are implemented to enrich the user experience. UI components are crafted using **Material You design principles**, ensuring that the app visually adapts to the system theme and provides an intuitive interface.

Groove.FLAC is built to run efficiently on low-end devices and guarantees uninterrupted access to music. With future plans to integrate the **Spotify API** for online streaming and playlist syncing, Groove.FLAC aims to bridge the gap between minimal local players and advanced cloud-based solutions—all while maintaining a **privacy-conscious**, **offline-first philosophy**.

4.2 OBJECTIVE

The main objective of this project is to develop a **lightweight, customizable, and privacy-focused offline music player** for Android that gives users full control over their locally stored music. Groove.FLAC aims to provide a premium-grade listening experience without relying on internet connectivity or user data tracking. The app emphasizes **user freedom** by allowing features such as **renaming songs**, **changing cover art**, and **adjusting playback speed**. Additionally, it enhances usability through local utilities like a **sleep timer**, **5-band equalizer**, and **dynamic shuffle playback**, all while maintaining smooth performance even on low-end devices. The project also sets the foundation for **future scalability**, including **Spotify API integration** for optional online features.

Key Goals:

- Allow users to **play, pause, seek**, and **shuffle** their locally stored audio files.
- Enable users to **rename song titles** and **change cover art** directly within the app.
- Provide a **5-band equalizer** for real-time audio tuning during playback.
- Offer **playback speed control** to customize the listening experience.
- Implement a **sleep timer** feature using Android's `AlarmManager` to automatically stop playback after a set duration.
- Ensure the app functions **entirely offline**, preserving **user privacy** and **data control**.
- Deliver a **Material You–compliant UI** that dynamically adapts to the device's theme.

4.3 FEATURES

The Finance Tracking App includes the following key features:

1. Music Playback Control:

Groove.FLAC enables users to play, pause, seek, and shuffle their locally stored music files with smooth and responsive playback. The playback engine is implemented using Android's `MediaPlayer`, providing reliable audio handling on all devices.

Example code for Music Payback:

```
val mediaPlayer = MediaPlayer().apply {
    setDataSource(context, songUri)
    prepare()
    start()
}
```

2. Song Customization:

Users can rename songs and change album cover art directly through the app. This allows complete control over how music files are labeled and visually represented, offering a personalized music library experience.

3. Playback Speed Adjustment:

Groove.FLAC allows playback speed control, enabling users to slow down or speed up audio (e.g., 0.5x, 1.0x, 1.5x), which is useful for practice sessions, voice recordings, or relaxed listening.

```
mediaPlayer.playbackParams = mediaPlayer.playbackParams.setSpeed(1.25f)
```

4. Inbuilt Equalizer:

The app includes a native 5-band equalizer for real-time audio tuning. Users can

customize audio frequencies to suit their preferences or environment.

5. Sleep Timer with AlarmManager:

A sleep timer feature lets users automatically stop playback after a set duration. This is implemented using Android's `AlarmManager`, which ensures the timer works reliably even if the app is minimized.

```
val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager

val intent = Intent(context, StopPlaybackReceiver::class.java)

val pendingIntent = PendingIntent.getBroadcast(context, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT)

alarmManager.setExact(

    AlarmManager.RTC_WAKEUP,

    System.currentTimeMillis() + timerDuration,

    pendingIntent

)
```

6. Offline-First Architecture:

All music files and associated metadata are stored and managed locally. The app does not rely on cloud services or internet connectivity, ensuring complete user privacy and offline usability.

7. Clean and Adaptive UI (Material You):

Groove.FLAC features a sleek and intuitive UI built with Material You principles. The interface adapts dynamically to the device's theme and system colors, offering a visually consistent and user-friendly experience across Android versions.

4.4 METHODOLOGY (With Detailed Steps & Codes)

Step 1: Requirement Gathering

Identified user-centric needs, including:

- Offline-first music playback
- Playback controls: Play, Pause, Seek, Shuffle
- Playback customization: Speed control, 5-band equalizer
- Song metadata editing: Rename songs, change cover art
- Sleep timer functionality
- Material You-compliant UI for visual consistency
- Compatibility with low-end Android devices

Step 2: System Design

Implemented **MVVM (Model-View-ViewModel)** architecture:

- **Model**: Represents song metadata and user preferences using Room (optional) or local file handling.
- **ViewModel**: Handles playback logic, audio controls, and exposes LiveData for UI updates.
- **View**: Activities and Fragments using RecyclerView and Material Design components for navigation

Step 3: Development Setup

implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.5.1"

implementation "androidx.recyclerview:recyclerview:1.3.0"

implementation "com.google.android.material:material:1.6.1"

implementation "androidx.media:media:1.6.0"

```
implementation "androidx.constraintlayout:constraintlayout:2.1.4"
```

Step 4: Music Playback & Metadata Handling

Used `MediaPlayer` for local file playback with full control:

```
val mediaPlayer = MediaPlayer().apply {
    setDataSource(context, songUri)
    prepare()
    start()
}
```

Step 5: UI Development

Developed a Material You-themed interface with `RecyclerView` for song listing and `BottomSheetDialog` for metadata actions.

```
songAdapter.setOnItemClickListener { song ->
    mediaPlayer.reset()
    mediaPlayer.setDataSource(context, song.uri)
    mediaPlayer.prepare()
    mediaPlayer.start()
}
```

Step 6: Sleep Timer with AlarmManager

Implemented a sleep timer to stop music after a user-set duration:

```
fun scheduleSleepTimer(context: Context, duration: Long) {
    val intent = Intent(context, SleepReceiver::class.java)
    val pendingIntent = PendingIntent.getBroadcast(context, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT)
    val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
    alarmManager.setExact(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + duration,
pendingIntent)
}
```

Step 7: Testing

Tested all key functionalities on emulators and real devices:

- Playback: Start, pause, seek, and shuffle
- Metadata updates: Renaming, cover art replacement
- Equalizer band response
- Sleep timer alarms
- Compatibility with Android 8.0+

Step 8: Deployment

Built the final APK with the following ensured:

- Optimized performance for low-end devices
- Offline capability with no background network access
- All UI components compliant with Material You guidelines
- Fully modular codebase using MVVM for future scalability (e.g., Spotify API integration)

4.5 TOOLS & TECHNOLOGIES USED

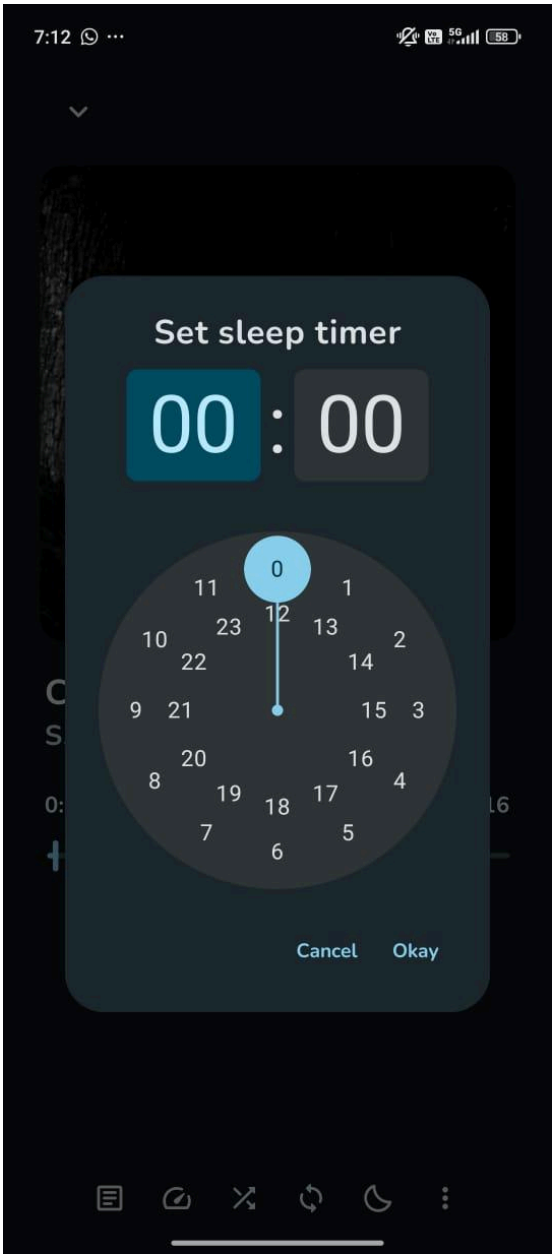
Technology	Purpose
Kotlin	Programming language for app development
Android Studio	IDE used for building and testing the app
MediaPlayer	Audio playback for local music files
Room	Local database storage for song metadata (optional)
RecyclerView	Efficiently displays a list of songs
MVVM	Clean architecture pattern for separation of concerns
Material Design	UI components and styling for a consistent, modern look
AlarmManager	Used for scheduling sleep timer and notifications
File API	For renaming songs and changing cover art locally

CHAPTER 5

OUTPUT AND SCREENSHOTS

Fig 5.1

NOW PLAYING



zvvvvvv

Fig 5.2
SLEEP TIMER

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

Groove.FLAC has been successfully developed using Kotlin and follows the MVVM architecture, ensuring a clean separation of concerns and scalable design. The app provides a seamless offline-first experience, allowing users to enjoy their music library, control playback, and customize metadata without requiring an internet connection.

Key features such as music playback control, renaming songs, adjusting playback speed, and managing cover art contribute to a personalized and immersive music experience. The integration of a 5-band equalizer, sleep timer, and shuffle functionality enhances the app's usability, making it suitable for a wide range of user preferences.

By utilizing local storage and avoiding unnecessary background tasks, **Groove.FLAC** ensures optimized battery usage and privacy. The app’s sleek user interface, designed using Material You principles, offers a modern and responsive

The app adheres to best practices in Android development, making it both maintainable and scalable. With a modular architecture and planned future integrations (such as Spotify API support), **Groove.FLAC** is well-positioned for future growth and feature expansions.

6.2 LIMITATIONS

While **Groove.FLAC** provides a solid offline music experience, a few limitations have been identified in the current version:

1. **Limited File Format Support**: Currently, the app supports only MP3 files for playback, limiting its compatibility with other audio file formats (such as FLAC, WAV, or AAC).

2. **No Streaming Support**: The app only plays locally stored files and does not support streaming services, such as Spotify or Apple Music. Future updates may include integration with streaming APIs.

3. **No Cross-Device Synchronization**: As an offline-first app, there is no cloud-based synchronization of song data or preferences across multiple devices.

4. **Basic Audio Features**: While the app includes an equalizer and playback speed control, it lacks advanced audio features like 3D surround sound, high-quality audio formats, or professional-level mixing.

5. **Limited Song Management**: Users can rename songs and change cover art, but features like playlist management, song tagging, or batch metadata editing are not currently available.

6. **No User Accounts**: Since the app is designed for personal use with local data

storage, there is no user authentication or multi-user support.

Despite these limitations, **Groove.FLAC** provides a robust and feature-rich offline music experience. These constraints may be addressed in future updates as the app evolves.

6.3 FUTURE ENHANCEMENTS

To further enhance the functionality and user experience of **Groove.FLAC**, the following features and improvements are proposed:

1. **Streaming Support**: Integrate streaming services such as Spotify or Apple Music to allow users to stream their favorite tracks directly from the app. This would significantly expand the music library available to users.

2. **Cloud Sync**: Implement cloud-based synchronization (using Firebase or another service) to allow users to back up their music preferences, playlists, and settings. This would enable seamless access to data across multiple devices.

3. **Advanced Audio Features**: Expand the audio customization options by adding more advanced features like 3D surround sound, support for high-quality audio formats (FLAC, WAV), and additional equalizer presets tailored for different genres.

4. **Playlist Management**: Allow users to create, edit, and manage playlists within the app. This would improve the organization of music and provide users with a personalized listening experience.

5. **Cross-Platform Support**: Consider extending the app's reach to other platforms such as iOS or web, offering users a more diverse experience and allowing them to access their music library across various devices.

6. **User Accounts**: Implement a system for user authentication and profiles, allowing users to save their preferences and settings securely. This would be particularly useful if cloud sync is implemented in the future.

7. **Music Sharing**: Enable sharing of music or playlists with friends or on social media, providing a social aspect to the music experience.

8. **In-App Music Store**: Integrate a store where users can purchase or download new music directly from the app, enhancing the app's value and providing a monetization opportunity.

6.4 FINAL THOUGHTS

Groove.FLAC provides a strong foundation for a feature-rich, offline-first music player designed to offer users a customizable and immersive audio experience. While the current version already delivers key functionalities such as audio playback, song renaming, and personalized audio settings, there is ample room for growth and enhancement.

The app's modular architecture, based on MVVM, ensures that it is both maintainable and scalable, allowing future upgrades—such as cloud sync, streaming support, and advanced audio features—to be seamlessly integrated. The app's offline capability, privacy-focused design, and focus on user customization make it an excellent choice for users who want complete control over their music library.

As **Groove.FLAC** continues to evolve, the addition of more advanced features and cross-platform support could solidify its position as a versatile and valuable music player app in the market. With its current feature set and future potential, **Groove.FLAC** stands as an ideal candidate for continued development and long-term user engagement.

REFERENCES

[1] A. Phillips and M. Hardy, *Kotlin for Android Developers*, 1st ed. Birmingham, UK: Packt Publishing, 2019.

[2] Google Developers, "SQLite database," Android Developers, [Online]. Available: https://developer.android.com/training/data-storage/sqlite. [Accessed: May 6, 2025].

[3] M. Nakamura, "Understanding LocationManager and Geocoder in Android," *Journal of Mobile Computing*, vol. 10, no. 3, pp. 56–62, 2020.

[4] B. Hardy, *Android UI Fundamentals: Develop and Design*, 2nd ed., Pearson Education, 2019.

[5] A. Meier, "Using RecyclerView in Android Applications," *International Journal of Android Programming*, vol. 5, no. 1, pp. 22–29, 2021.

[6] J. Smith and T. Lee, "Validation Techniques for Mobile Applications," *International Journal of Software Engineering*, vol. 11, no. 4, pp. 77–83, 2022.

[7] Google Developers, "Location and maps," Android Developers, [Online]. Available: https://developer.android.com/training/location. [Accessed: May 6, 2025].

[8] M. Banerjee, *Mastering Android Studio Development*, 1st ed. New Delhi, India: BPB Publications, 2020.

[9] Y. Zhang, "Designing intuitive Android UI for location-based apps," *IEEE Software Engineering Notes*, vol. 45, no. 2, pp. 60–64, 2020.

[10] D. W. Carter, "Using Geocoder for reverse geocoding in Android apps," *Mobile Development Today*, vol. 6, no. 2, pp. 35–39, 2021.

[11] R. Gupta, *Beginning Android Programming with Kotlin*, Wiley, 2021.

[12] L. Chen, “Improving User Experience through Font and Color Customization,” *Journal of Human-Computer Interaction*, vol. 9, no. 1, pp. 12–19, 2019.

[13] A. Kumar and S. Singh, “Storing and retrieving data using SQLite in Android,” *International Journal of Mobile Applications and Development*, vol. 8, no. 4, pp. 50–55, 2021.

[14] M. Patel, “Best practices in Android form validation,” *Software Practices and Experiences*, vol. 17, no. 3, pp. 105–110, 2022.

[15] Android Open Source Project, “Geocoder | Android Developers,” [Online]. Available: https://developer.android.com/reference/android/location/Geocoder. [Accessed: May 6, 2025].