GENZIFY-Music Streaming Application

K. Anandan[1], K. Jegathish[2]

[1]Assistant Professor, Department of Computer Applications,

Nehru college of Management, Coimbatore, Tamil Nadu, India

[2]Student of II MCA, Department of Computer Applications,

Nehru college of Management, Coimbatore, Tamil Nadu, India

1.ABSTRACT:

The Genzify Music Streaming Application is a mobile application designed to provide seamless music playback while leveraging modern technologies such as Firebase, Hive, and Bloc for efficient state management. The app enables users to stream songs uploaded by an admin, manage playlists, and cache music for offline playback. Firebase Firestore stores metadata, Firebase Storage hosts audio files, and Hive facilitates local caching. The application integrates authentication, background playback, and a user-friendly interface. This journal outlines the development process, including project structure, database design, state management, and future enhancements to ensure a scalable and efficient music streaming experience.

Keywords: Flutter, Firebase, Bloc, Music

2.INTRODUCTION:

Music streaming has become an integral part of digital entertainment, allowing users to access vast libraries of songs anytime and anywhere. With the increasing demand for personalized and high-quality music experiences, developing a robust and scalable music streaming application has become essential. The Genzify Music Streaming Application is designed to provide a seamless and immersive listening experience using modern technologies.

This application is developed using Flutter, an open-source UI framework, along with Dart for programming. It leverages Firebase for backend services, including Firestore for song metadata, Firebase Storage for hosting audio files, and Firebase Authentication for secure user management. Additionally, Hive is used for local storage, allowing offline playback and caching of song metadata. To efficiently manage the app's state, Bloc is utilized, ensuring a structured and maintainable architecture.

The app features user authentication, song streaming, playlist management, background playback, and search functionalities. Admins have the ability to upload and manage songs, ensuring a well-organized content library. Users can create and manage playlists, mark favorite songs, and listen to music even when offline. The JustAudio package is integrated for high-quality audio playback, providing features such as play, pause, skip, and seek.

3.METHODOLOGY:

3.1 Requirement Analysis:

- Identifying user needs and technical requirements.
- Defining features such as authentication, music playback, and playlist management.

3.2 Technology Selection:

- Choosing Flutter for the frontend due to its cross-platform capabilities.
- Selecting Firebase for cloud services and Hive for local storage.
- Using Bloc for state management to ensure organized application flow.

3.3 System Design and Architecture:

- Designing the UI/UX for an intuitive user experience.
- Structuring the database for efficient storage and retrieval of music metadata.
- Defining the Bloc structure for managing states effectively.

3.4 Application Development:

- Implementing authentication using Firebase Authentication.
- Integrating Firestore and Firebase Storage for song management.
- Developing music playback using the JustAudio package.
- Implementing local storage caching with Hive.
- Structuring state management using Bloc.

4.FUTURES AND FUNCTIONS:

4.1 User Features:

4.1.1 User Authentication

- Users can sign up, log in, and reset passwords using Firebase Authentication.
- Supports authentication via email/password or third-party providers (Google, Facebook).
- Ensures secure and seamless user login sessions.

4.1.2 Music Streaming

- Users can stream high-quality audio directly from Firebase Storage.
- Songs are loaded dynamically to optimize performance and reduce buffering.

4.1.3 Playlist Management

- Users can create, edit, and delete custom playlists.
- Playlists store references to songs, making them easy to organize.

4.1.4 Search and Filter

- Users can search for songs based on title, artist, or genre.
- Filtering options allow users to browse music by specific categories.

4.1.5 Background Playback

- Music continues playing even when the app is minimized.
- Users can control playback via notifications or the lock screen.

4.1.6 Playback Controls

- Includes play, pause, skip, seek, and repeat functionalities.
- Users can control volume and shuffle songs within playlists.

4.2 Admin Features:

4.2.1 Song Upload and Management

- Admins can upload songs directly to Firebase Storage.
- The uploaded song's metadata (title, artist, album, genre) is stored in Firestore.
- Admins can edit song details or delete songs if necessary.

4.2.2 Content Moderation

- If user-generated content is allowed, admins can approve or reject uploaded songs.
- Ensures that only high-quality, appropriate content is available for streaming.

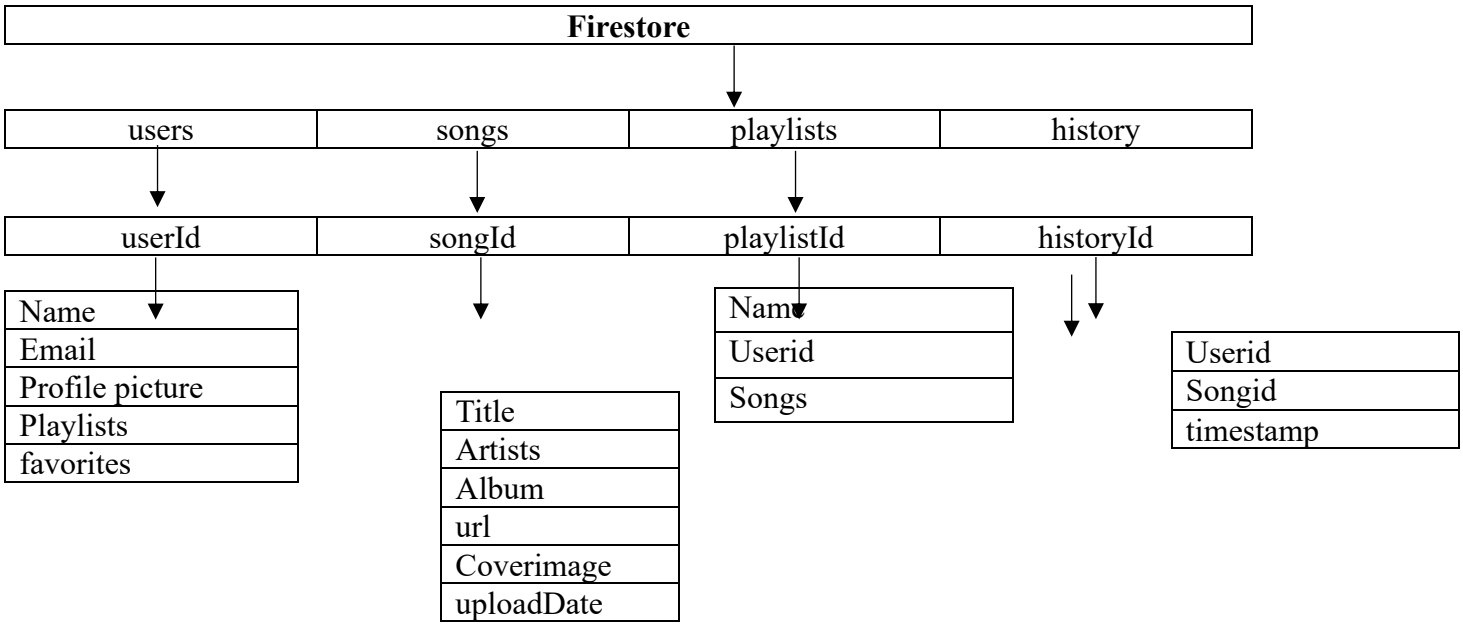
4.2.3 User Management

- Admins can monitor registered users, view their playlists, and manage user accounts.
- If needed, admins can ban users who violate app policies.

5.DATABASE DESIGN:

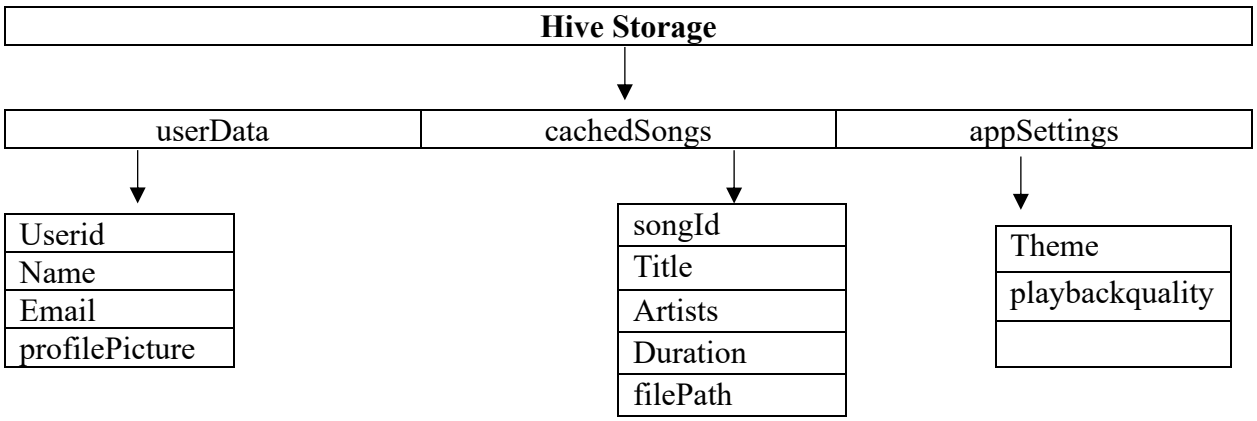
5.1 Firestore Database Structure:

The Firestore database for the Flutter Music Streaming Application is structured using collections and documents to store user data, songs, playlists, and playback history efficiently. The user's collection holds user profiles, including authentication details and favorite songs. The songs collection contains metadata such as title, artist, genre, and Firebase Storage links for streaming. The playlists collection allows users to create and manage custom song lists, while the history collection tracks recently played songs. This hierarchical structure ensures scalability, quick data retrieval, and seamless user experience.

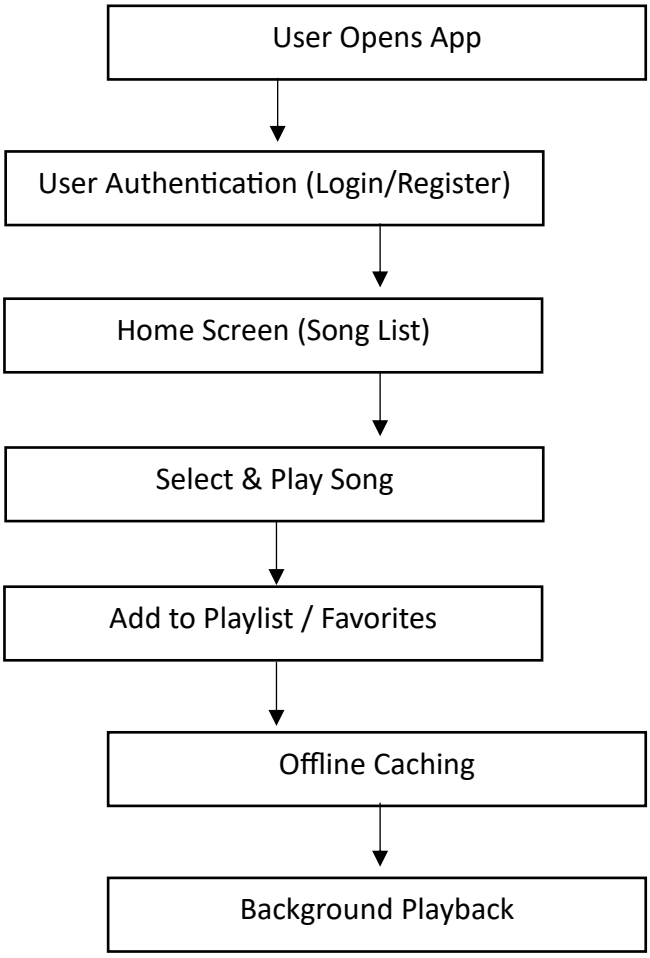


5.2 Hive Local Database Structure**:**

Hive is a lightweight and fast NoSQL database used for local storage and caching in the Flutter Music Streaming Application. It efficiently stores user data, cached songs, and app settings in key-value pairs, ensuring quick access and offline functionality. The user_data box stores user profiles, while the cached_songs box saves downloaded songs with their metadata for offline playback. Additionally, the app_settings box maintains theme preferences and playback settings. Hive's simplicity and speed make it ideal for handling local data efficiently without requiring heavy database operations.



6.USER WORKFLOW DIGRAM:



7.EXISTING SYSTEM:

Traditional music streaming applications rely on centralized cloud-based platforms that provide users with access to large music libraries. However, these systems often require a constant internet connection, making offline access limited. Many existing applications lack efficient caching mechanisms for offline playback, leading to higher data consumption. Additionally, some platforms have limited user control over playlist management and personalization features. Security concerns, such as unauthorized access to user data and content, also exist due to inadequate authentication and encryption mechanisms.

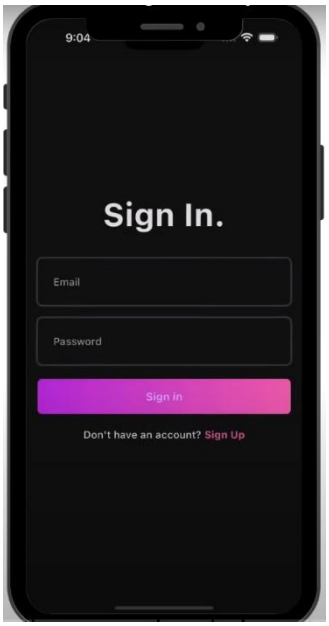
8.PROPOSED SYSTEM:

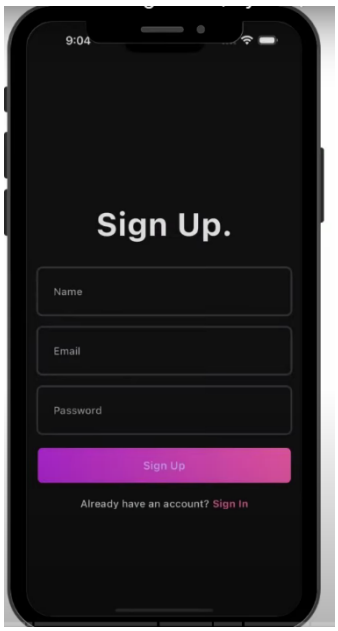
The Genzify Music Streaming Application is designed to overcome the limitations of existing systems by providing seamless online and offline music playback. It utilizes Firebase Firestore for structured song metadata storage, Firebase Storage for hosting audio files, and Hive for efficient offline caching. The integration of Bloc state

management ensures a smooth and responsive UI. With admin-controlled content management, secure authentication, and personalized user experiences like playlists and favorites, the system aims to be a scalable, user-friendly, and high-performance music streaming solution.

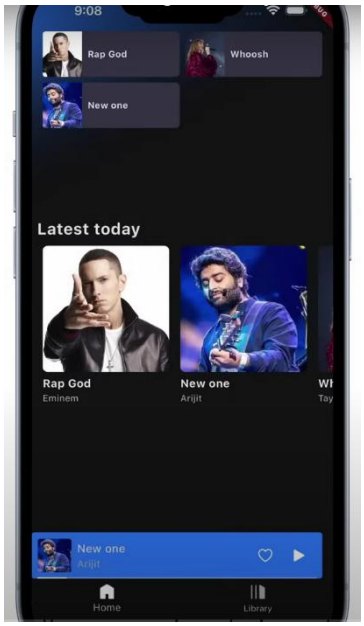
9.RESULTS:

Sign in & Signup page





Homepage



Musicplayer page

9.CONCLUSION:

The Genzify Music Streaming Application successfully delivers a seamless and engaging music experience with online streaming, offline playback, and personalized playlists. The integration of Firebase Firestore, Firebase Storage, and Hive ensures efficient data storage, retrieval, and caching. Bloc state management enhances performance, providing smooth UI interactions and real-time updates. Admins can efficiently manage music content, ensuring a well-organized library for users. Overall, this project demonstrates a scalable, high-performance, and user-friendly music streaming solution with future potential for advanced features like AI-based song recommendations and lyrics display.

10.REFERENCE:

1. "Flutter - a new way to build high-performance, low-latency mobile applications" by M. L. Hartnett and D. D. Hilliard (https://dl.acm.org/doi/pdf/10.1145/3356857.3356858)

2. "Firebase: Real-time Database and Authentication for Mobile Apps" by M. Mehta (https://www.packtpub.com/product/firebase-real-timedatabase-and-authentication-for-mobile-appsvideo/9781800202685)

3. React Native vs Flutter, Cross-Platform Mobile Application Framework, Thesis March 2018- Wenhua Wu

4. A clean approach to Flutter Development through the Flutter Clean architecture package, IEEE 2019, Shady Boukhary, Eduardo Colmenares.

5. Flutter for Cross-Platform App and SDK Development, Metropolia University Thesis May 2019- Lucas Dagne.

6. Exploring end user’s perception of Flutter mobile apps, Malmo University Nov 2019- Dahl, Ola.

7. Cross-Platform Framework comparison- Flutter vs React Native.

8. Flutter Native Performance and Expressive UX/UI, paper 2019- Tran Thanh.