Music Streaming Web Application Report

1. Introduction

The music streaming web application is a platform designed to allow users to discover, listen to, and manage music tracks. It provides functionality for both regular users and creators (artists) to interact with the system. The purpose of this report is to provide an overview of the application's features, architecture, and implementation details.

2. Features

- User Authentication: Users can register, log in, and log out of the system. Passwords are securely hashed using bcrypt before being stored in the database.
- Role-Based Access: The application supports two user roles: regular users and creators. Each role has access to different features and functionalities.
- Music Playback: Users can browse and listen to songs uploaded by creators. The application supports basic music playback controls such as play, pause, and skip.
- Playlist Management: Authenticated users can create and manage playlists to organize their favorite songs. Playlists are associated with user accounts and can be accessed across different devices.
- Rating System: Users can rate songs, and the application calculates and displays the average rating for each song.
- Creator Dashboard: Creators have access to a dashboard where they can upload new songs, update existing songs, and view their uploaded content.

3. Architecture

The application follows a client-server architecture, with the server-side implemented using Flask, a lightweight web framework for Python. The server interacts with a SQL database managed by SQLAlchemy, an ORM (Object-Relational Mapping) library.

- Frontend: The frontend of the application is built using HTML templates rendered by Flask's Jinja2 templating engine. CSS and JavaScript are used to style and enhance the user interface.

- Backend: The backend logic is implemented using Python and Flask. It includes route definitions, request handling, and database interactions.

- Database: The application uses a relational database (SQLite in development) to store user data, song metadata, playlists, ratings, and other information. SQLAlchemy provides an abstraction layer for interacting with the database.

4. Implementation Details

- User Model: The `User` model represents user accounts and includes fields for username, email, password (hashed), and a boolean flag indicating whether the user is a creator.

- Song Model: The `Song` model stores information about individual songs, including title, artist (creator), album, file path, genre, date posted, lyrics, rating, and duration.

- Creator Model: The `Creator` model represents artists or content creators and includes fields for name, email, password (hashed), and a rating.

- Album Model: The `Album` model represents albums, grouping songs by a common theme or release. It includes fields for name, rating, artist, and a relationship with the `Song` model.

- Playlist Model: The `Playlist` model stores associations between users and their playlists, as well as the songs included in each playlist.

- Rating Model: The `Rating` model tracks user ratings for individual songs, ensuring that each user can rate a song only once.

5. Conclusion

In conclusion, the music streaming web application provides a user-friendly platform for discovering and enjoying music. It leverages Flask and SQLAlchemy to implement essential features such as user authentication, music playback, playlist management, and rating systems. The application architecture is designed to be scalable and extensible, allowing for future enhancements and updates. Overall, the application aims to provide a seamless and enjoyable music streaming experience for users and creators alike.

6. Future Enhancements

- Improved User Interface: Enhance the frontend design and user experience with modern UI frameworks and responsive layouts.
- Advanced Music Recommendations: Implement machine learning algorithms to provide personalized music recommendations based on user preferences and listening history.
- Social Features: Introduce social features such as following other users, sharing playlists, and collaborative playlist creation.
- Integration with Music APIs: Integrate with external music APIs (e.g., Spotify, Apple Music) to expand the catalog of available songs and provide additional features.
- Enhanced Creator Tools: Provide creators with analytics dashboards, revenue tracking, and tools for promoting their music.
- Continuous Testing and Deployment: Implement automated testing and continuous integration/continuous deployment (CI/CD) pipelines to ensure the reliability and scalability of the application.

7. References

[1] Flask Documentation: https://flask.palletsprojects.com/

[2] SQLAlchemy Documentation: https://www.sqlalchemy.org/

[3] Bootstrap Documentation: https://getbootstrap.com/

[4] Jinja2 Documentation: https://jinja.palletsprojects.com/

[5] bcrypt Documentation: https://pypi.org/project/bcrypt/

[6] PyDub Documentation: https://pydub.com/

[7] FFmpeg Documentation: https://ffmpeg.org/

This report provides an overview of the music streaming web application, its features, architecture, implementation details, future enhancements, and references for further reading.

Video link- https://youtu.be/IlqdH8pqgvE