Project Report: VibeFlow
Flask Music Streaming App

Author: Avinash

Roll No: 22F3001961

22f3001961@ds.study.iitm.ac.in

Abstract:

The Flask Music Streaming Application (VibeFlow) is a robust web-based service that provides users with an immersive music streaming experience. Developed using the Flask framework and Flask-SQLAlchemy for efficient database management, the application focuses on simplicity and user-centric design. This report outlines the key features, technologies used, system architecture for VibeFlow.

Introduction:

I, Avinash, a student pursuing a degree in Data Science & Application from IIT Madras, present the Flask Music Streaming Application. This project combines my passion for coding with a deep interest in music. VibeFlow is designed to provide a seamless and accessible platform for music lovers, leveraging the power of Flask and related technologies.

Technologies Used:

- **Flask**: Web framework for building the application.
- **SQLAlchemy**: Database ORM for efficient data management.
- **Flask-Login**: User authentication and session management.
- **SQLite**: Database management system.
- **HTML, CSS, JavaScript**: Front-end development.

Features and Functionalities:

- **User Registration and Login:**
  Secure sign-up for users to access their personalized music space.
- **Song Filtering:**
  Efficient search and sorting options for songs based on ratings, titles, or artists.
- **Playlist Management:**
  Intuitive tools for users to create, edit, and organize personal playlists.
- **Creator Dashboard and Album Management:**
  A dedicated interface for music creators to upload new tracks, create albums, and manage their discography.

- **Admin Dashboard for User and Song Management:**
  Robust admin controls to oversee user accounts, song entries, and overall content management for maintaining platform integrity.
- **Switching Roles between Creator and User:**
  User can switch roles between creator and User as and when required
- **Secure Routes:**
  User cannot access routes of admin using URL editing as @roles_required function check whether user is authorised for accessing this route or not.

Video Link:

https://drive.google.com/file/d/1Fn8U0shfp1bxVfhHFRStv5vrnsJorpT7/view?usp=sharing

System Architecture:

- App.py is Central code file containing all codes required for initialising music app.
- Controller.py is containing routes for all pages and directing request to appropriate handlers
- App database configuration is written in config.py inside application folder.
- Templates are in the templates folder. Template have different folder for storing template of different roles like Admin / Creator / User / logged Out.
- Database model is in models.py inside application folder.
- The database is in the db_directory folder as music_app.sqlite3.
- Background images and audio files are in the static folder.

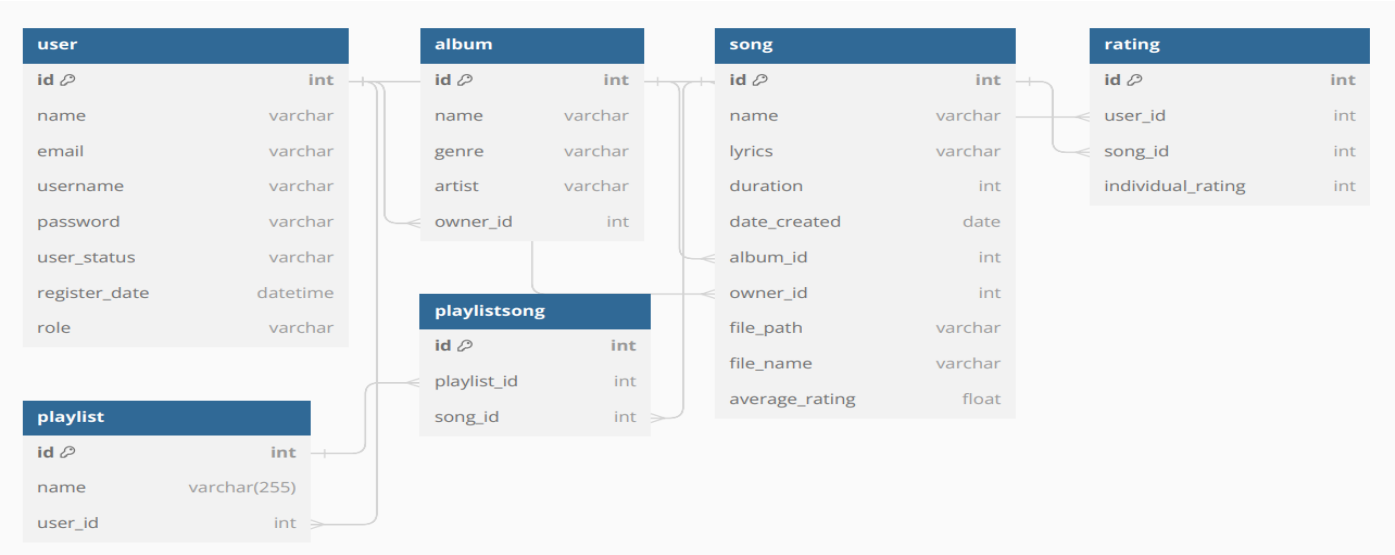
API Documentation:

1. **SongsData:**
   Purpose: CRUD operations on Songs.

2. **AlbumData:**
   Purpose: CRUD operation on Albums

Database Schema:

Flask SQLAlchemy was used for database management. The schema is briefed below:



Route Details:

All the route in controller.py are as follows:

Logged Out Routes:

1. **/** for index page
2. **/about** for about page
3. **/signup** for signup page
4. **/login** for login page
5. **/admin_login** for admin login page
6. **/logout** for logout page

Admin Routes:

1. **/admin** for admin dashboard
2. **/admin/user_management** for user management
3. **/admin/all_users** for all users
4. **/admin/all_creators** for all creators
5. **/admin/all_albums** for all albums
6. **/admin/promote_user/<int:user_id>** for promoting user to creator
7. **/admin/demote_user/<int:user_id>** for demoting creator to user
8. **/admin/user_management/block/<int:user_id>** for blocking user/
9. **admin/user_management/unblock/<int:user_id>** for unblocking user
10. **/admin/song/<int:song_id>** for playing song
11. **/admin/album/<int:album_id>** for playing album
12. **/admin/all_songs** for all songs
13. **/admin/search** for searching
14. **/admin/song/delete/<int:song_id>** for deleting song
15. **/admin/search for searching**

Creators Routes:

1. **/creator/dashboard** for creator dashboard
2. **/creator/switch_to_user** for switching to user
3. **/creator/album** for creator album
4. **/creator/album/create_new** for creating new album
5. **/creator/song** for creator song
6. **/creator/song/create_new** for creating new song
7. **/creator/song/delete/<int:song_id>** for deleting song
8. **/creator/song/edit/<int:song_id>** for editing song
9. **/creator/song/<int:song_id>** for playing song
10. **/creator/album/<int:album_id>** for playing album
11. **/creator/search for searching**

User Routes:

1. **/user/dashboard** for user dashboard
2. **/user_profile** for user profile
3. **/user_profile/edit** for editing user profile
4. **/user/all_albums** for all albums
5. **/user/all_songs** for all songs
6. **/user/song/<int:song_id>** for playing song
7. **/user/register_creator** for registering as creator
8. **/rate_song/<int:song_id>** for rating song

References:

- Flask Documentation: Link
- SQLAlchemy Documentation: Link
- Flask-Login Documentation: Link