IITM Appdev1 Project

Music Streaming app

Project report

By: Janarthanan G

22f3002040

Diploma level

Description:

This is a music streaming application that is developed using Flask and utilizes SQLite as its database, featuring multi-user capabilities with Role-Based Access Control (RBAC). The app incorporates user authentication, enabling users to discover and play songs, rate them, and curate playlists. Additionally, users have the ability to manage their playlists. Creators are empowered to oversee songs and albums, while admins have the authority to manage both users and songs. Furthermore, admins have access to analytics to monitor and assess the app's performance.

Tech stack:

1. **Flask:** Flask is a backend framework for python. It includes flexible core functionality and an extensive ecosystem of supported modules like Flask-SQLAlchemy for database access, Flask-Login for session management, and Flask-RESTful for API development.
2. **Sqlite:** SQLite is a embedded database engine written in C. It's self-contained, i.e. it doesn't require a separate server process.
3. **Htmx :** Htmx is javascript library used perform AJAX operations without the of javascript. We can use html attributes to perform AJAX operations.
4. **Matplitlib:** Matplotlib is a powerful plotting library for Python. It provides a flexible and comprehensive set of tools for creating various types of plots and charts. Matplotlib is widely used in data visualization, scientific computing, and other fields where graphical representations of data are crucial.

Database Schema design:

This is my database Schema design for my music streaming app.

User Table: This table comprises an "id" as the primary key, along with fields for "username," "password," "number," "email," and "role," where the role can take on values such as "Creator," "Admin," or default to "User." It is instrumental in implementing Role-Based Access Control (RBAC).

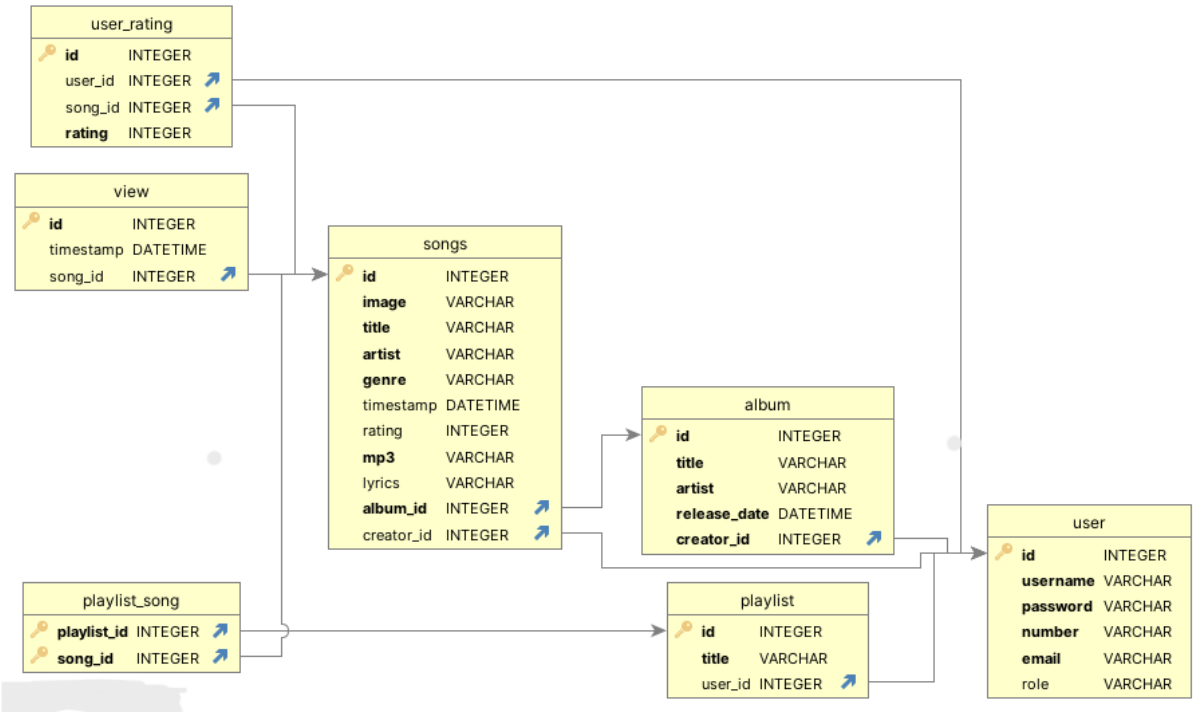
Songs Table: The Songs table includes an "id" as the primary key, as well as fields for "image," "title," "artist," "genre," "timestamp," "rating," "mp3," "lyrics," and "album_id" (a foreign key referencing the Album table). Additionally, it contains "creator_id" as a foreign key referencing the creator.

Album Table: This table features an "id" as the primary key, along with fields for "title," "artist," "release_date," and "creator_id" (a foreign key referencing the creator).

View Table: The View table consists of an "id" as the primary key, along with fields for "timestamp" and "song_id" (a foreign key referencing the Songs table).

Playlist Table: This table is composed of "id," "title," and "user_id," with "user_id" serving as a foreign key referencing the User table.

Playlist_song Table: To establish a many-to-many relationship between playlists and songs, the Playlist_song table includes "playlist_id" and "song_id."



User_rating Table: This table encompasses "id," "user_id," "song_id," and "rating" fields, providing a mapping for user-specific ratings assigned to particular songs.

API Design:

GET: Retrieve data (Users, songs, albums, playlists, etc)

POST: Create new entries (Users, songs, albums, playlists, etc)

DELETE: To delete Entries (Users, songs, playlists, etc.)

Architecture and design:

app.py: Within the app.py file, the Flask instance is initialized to serve as the hub for defining all routes and endpoints. When accessing specific endpoints, the `rendertemplate` function is employed to seamlessly load HTML, integrating Jinja templating for enhanced dynamic content rendering.

models.py: Utilizing SQLAlchemy, models for database entities are designed and implemented within the models.py file. These models are then seamlessly imported into the app.py file, allowing for the execution of query operations on the database. This modular approach enhances the maintainability and scalability of the overall application architecture.

api.py: This file serves as the foundation for constructing and managing a REST API implemented with the Flask-RESTful module. It contains classes inherited from the resource module, specifically designed for handling songs, albums, and generating graphs for the admin dashboard. The `app.py` file is responsible for initializing the API instance, where all the API classes are imported and configured.