Music Streaming App
(AppDev2- Project)

Name : Naveen Kumawat

Roll No. : 21f1001711

Student Email : 21f1001711@ds.study.iitm.ac.in\
I am a diploma level student at IITM BS.

Description:

This app is a multi-user web app for listening to and uploading music.
The app is built on the Vue.js framework for the client-side and Flask for the server-side. Additional features include scheduled jobs and daily reminders using redis and celery and token-based authentication using flask security.

First I created the database schema of the app, then a proper login system, then developed the apis, logic, user interface. At last I implemented the celery jobs.

Technologies Used:

- Vue.js - The client side/ frontend part of the app is built on Vue.js.
- Flask - The server side/ backend part of the app is built on Flask.
- Redis and Celery are used for scheduled jobs/daily reminders via Google Chat and MailHog.
- Flask restful for managing the api calls.
- Flask security for token based authentication.
- Smtplib and MIMEMultipart to send multipart messages using simple mail transfer protocol.
- SQLite3 and Flask-SQLAlchemy - to create and manage the relational database for the app.
- Matplotlib - to plot the app statistics graphs for the admin dashboard.
- Bootstrap - for templates of the web pages.
- Jinja2 - for generating Monthly activity reports at backend

Database:

- Database models/tables for the app are created using flask-sqlalchemy.
- There are 7 Tables used in the database: User, Role, RolesUsers, Song, Album, Playlist, Playlist_Song_Table.
- Users are differentiated based on their roles using the RolesUsers table.
- Song and Album have many to one relationship.
- Playlist and Song have many to many relationships. This relationship data is stored in Playlist_Song_Table.

System Design:

- This web app follows MVC architecture style:-
    - Model(M) is handled by flask. Flask interacts with the database and manages the data model.
    - View(V) is handled by vue.js. Vue components are responsible for interactive user interface.
    - Controller(C) is handled by flask. Flask routes handle all the business logic at the backend.
- Instance Folder - stores the database of the app.
- Static Folder - stores all the graphs and audio files.
- Main.py - the code for the flask app instance, celery app instance and initializing api & database for the app.
- Sample_data.py - the code of some pre saved data of the app.
- Models.py - the code for creating database tables.
- Resources.py - the code for managing api calls on songs/albums.
- Worker.py, Celeryconfig.py, Task.py - the code for celery configuration, scheduled jobs and daily reminders.
- Views.py - the code for all the routes and endpoints.
- Requirements.txt, Imp_commands - store required dependencies.

Features Implemented:

- Separate login form for users and admin..
- Admin dashboard with app statistics and graphs.
- Admin can manage songs, albums and blacklist/delete creators.
- Admin/User can search songs/albums/artists/genres based on their type.
- Users can filter songs based on likes/views.
- Users can play songs, read lyrics, like/dislike/ flag songs, create/edit/delete playlists.
- Users can register to become creators.
- Creators can upload/edit/delete songs and albums.
- Monthly Activity report of creator is sent to creators email on first day of month.
- Daily notifications on google chat to users to visit the app if inactive for 24 hours.

To run the app:

Unzip the project folder, install all the dependencies from requirements.txt, run the commands mentioned in imp_commands.txt and run the main.py file.

Presentation Video Link:

https://drive.google.com/file/d/1y-f856c_OozdghbjE75jt5s9-LdcqRCM/view?usp=sharing