Music App System Report:

The Music App is a web application designed to manage and organize music albums, songs, playlists, and user interactions. It provides functionalities for users with different roles such as admins, creators, and regular users. The application is built using Python and Flask for the backend, with Vue.js for the frontend.

Backend Architecture

Models:

1.Users: Represents registered users of the system. Each user has a unique username, email, password, role, and joining date.
2.Album: Represents a music album with attributes like name, artist, release date, and creator ID.
3.Songs: Represents individual songs with attributes like name, lyrics, genre, duration, album ID, artist, ratings, and flag status.
4.Playlist: Stores user-created playlists with attributes like name, user ID, and song IDs.
5.song_statistics: Tracks statistics related to songs and albums, including play counts and dates.
6.user_visit: Records user visits with attributes like user ID and visit time.
7.**Ratings**: Stores user ratings for songs with attributes like song ID, user ID and rating.

Relationships :

User-Role Relationship: There is a many-to-many relationship between users and roles, facilitated by the `UserRoles` model (commented out in the code).
Album-Songs Relationship: Each album can have multiple songs associated with it.
Song-Statistics Relationship: Each song can have statistical data associated with it, tracking play counts and related album play counts.
User-Visit Relationship: Records user visits to the application.

Frontend Architecture:

The frontend of the MusicApp is developed using Vue.js, a popular JavaScript framework for building user interfaces. The project structure consists of Vue components organized under the `components` directory. Key views include:

Admin Views: Interfaces for managing albums, songs, and users.
Creator Views: Interfaces for managing songs and albums.
User Views: Interfaces for browsing and interacting with songs and playlists.
Authentication Views: Signup and login forms.

The MusicApp follows a client-server architecture:

Client-Side(Frontend): Vue.js handles the presentation layer, providing an interactive and responsive user interface. Vue components are organized hierarchically and communicate with the backend via Flask.

Server-Side(Backend): The Flask backend serves as the application server, handling HTTP requests, executing business logic, and interacting with the database. It uses SQLAlchemy as an ORM (Object-Relational Mapper) for database operations. Additionally, Celery is integrated for handling asynchronous tasks, such as background processing of user activities.

Batch/Scheduled Jobs: Scheduled jobs are run on the redis server and these jobs are carried out using celery (Celery worker and Celery Beat) and tested on a demo SMTP server named 'Mailhog'.The two tasks being carried out are 'Daily Remainder task' which sends email alert everyday evening to all the users of the app who have not visited the app during the day and the 'Monthly Report Task' which sends a report to the Admin of the app on the first day of the month consisting of various app statistics like most played song/album,most active user etc.

Batch/Scheduled Jobs: Role-Based Authentication is carried out in this app using Token-Based Authentication system where the JWT are used to create,encode and decode the tokens which are later stored in the localStorage of the browser.

Database: The application utilizes a relational database namely PostgreSQL making use of SQLAlchemy to persist data related to users, roles, albums, songs, playlists, and statistics.

Conclusion:

The MusicApp provides a comprehensive platform for managing and enjoying music content. With features tailored for different user roles and a modern frontend powered by Vue.js, it offers an engaging user experience while ensuring efficient backend operations and data management.

For further enhancements, considerations might include improving authentication and authorization mechanisms, enhancing frontend design, implementing more robust error handling, and scaling the system for increased user traffic.

Project Video Link :

https://drive.google.com/file/d/1AAMkohGH27dYr3fTHRsXz1UZuZ-RZS2o/view?usp=drive_link