- **Author Information:**
  - *Name* : Patel Nujhatbanu Idrisbhai
  - *Roll No* : 21f3001798
  - *Email* : 21f3001798@ds.study.iitm.ac.in
  - I am currently pursuing a diploma in Programming. Having completed my diploma in Data Science, I am now working on a MAD2 project. My primary focus is on machine learning, and I look forward to exploring it further in my academic pursuits.
- **Description:**
  - This music app enables users to create accounts using unique email IDs, upload music, view average song ratings, and manage personal albums and playlists by adding or deleting songs. Users have control over their profiles, showcasing the number of uploaded songs, and can search for music. Additionally, the admin possesses the authority to blacklist/whitelist users, restricting song uploads for those in the blacklist. The app also provides features for admin to view top songs and various charts. Additionally, users who have not logged into the music app for 24 hours will automatically receive a login reminder email, as well as a monthly summary of their profile.
- **Technologies Used:**
  - *Flask:* Minimalistic Python web framework for backend development.
  - *Flask-SQLAlchemy:* Simplifies database integration with SQLAlchemy for user and music data.
  - *Matplotlib*: Data visualization library for creating charts in the app.
  - *Flask- Login:* Manages user sessions, authentication, and personalized profiles.
  - *VueJs, Jinja Templates, Bootstrap:* Standard SPA technologies for frontend design and user interface.
  - Celery: To set up the periodic task
  - Redis : Caching
- **DB Schema Design:**
  - User Table:
    - id, email, password, first_name, last_name, white_list, activeTime
    - Relationships: playlists, songs, albums
  - Song Table:
    - id, name, lyrics, duration, singer, genre, created_at, ratings, num_ratings
    - Relationships: user_id, albums, playlists
  - Rating Table:
    - id, user_id, song_id, rating
  - Playlist Table:
    - id, name, created_at

        - Relationships: user_id, songs
    - Album Table:
        - id, name, genre, artist, created_at
        - Relationships: user_id, album_songs
- **Architecture and Features:**
    -
    - The music app's architecture is designed with a clear separation between the backend and frontend components. Within the backend section, the organization is structured to facilitate efficient management of various functionalities. The main backend directory contains essential files and folders, including cached Python files, a database directory named "instance" housing the primary database file "database.db," and HTML files for email content and monthly summaries. The core backend functionality is encapsulated in the "main.py" file, orchestrating key operations of the application. Additionally, the "website" directory comprises a comprehensive set of Python files responsible for distinct aspects of the backend, such as admin-related tasks, API endpoints, database interactions, email sending capabilities, and management of playlists and albums. The inclusion of Celery tasks definition in "tasks.py" and Celery worker initialization in "worker.py" highlights the utilization of Celery for asynchronous task execution. On the other hand, the frontend section of the app adopts a different structure, focusing on the Vue.js framework for building the user interface and interaction components. It consists of various configuration files, dependencies, and source code organized within directories like "src" for source files and "public" for public assets. This modular architecture enhances scalability, maintainability, and extensibility, enabling smooth operation and seamless integration of features within the music app.
- **Video:**
    - https://drive.google.com/file/d/1q-pPerO47oO7p_Tr3TSgGLVzRUHwdsFD/view?usp=drive_link