B.P.H.E. SOCIETY'S

INSTITUTE OF MANAGEMENT STUDIES
CAREER DEVELOPMENT & RESEARCH

A PROJECT REPORT

ON

Unreleased Beats - Songs Management System For DJ's

By

Mr. Sandesh Pawar

Master of Computer Application (MCA)

In Partial Fulfilment Of

SAVITRIBAI PHULE PUNE UNIVERSITY

2024-2025

Under the Guidance of

Prof. Gauri Patil

1. Index

Chap		Details	Page No
1.		**Introduction**	3
	1.1	Abstract	4
	1.2	Scope Of System	4
	1.3	Operating Environment - Hardware and Software	5
	1.4	Brief Description of Technology Used Operating systems used	6
2.		**Proposed System**	12
	2.1	Study of similar System	13
	2.2	Users of System	13
3.		**Analysis and Design**	14
	3.1	System Requirements (Functional and Non-Functional requirements)	15
	3.2	Entity Relationship Diagram(ERD)	17
	3.3	Use Case Diagram	18
	3.4	Class Diagram	19
	3.5	Activity Diagram	20
	3.6	Sample Input and Output Screen	26
4.		**Coding**	30
	4.1	Project code	31
5.		**Testing**	40
	5.1	Test Strategy	41
	5.2	Unit Test Plan	42
6		**Limitations of Proposed System**	43
7		**Proposed Enhancements**	45

CHAPTER 1

INTRODUCTION

1.1 Abstract

Unreleased Beats is a DJ song management system that provides a platform for DJs to manage unreleased and released songs, create albums, and promote their songs. The system allows admins to upload songs, categorise them, associate them with albums, and handle song promotion requests. The project automates key processes such as song addition, album management, and promotion approvals.

1.2 Scope of the System

The system allows users (DJs) to

- Play songs / albums and download them.
- Submit promotion requests for songs.

The system allows Admin (DJs) to

- Add, edit, and delete songs.
- Create and categorise albums.
- Admins can approve or reject promotions.

1.3 Operating Environment

- **Hardware Requirements:**
    - **Developer Side:**
        - Processor: Core i3 or higher.
        - RAM: 4 GB or higher.
    - **Client Side:**
        - RAM: 512 MB and higher.
        - Browser: Chrome, Firefox, Safari.

- **Software Requirements:**
    - **Developer Side:**
        - Operating System: Windows 10+, macOS, or Linux.
        - Database: PostgreSQL.
        - Server: Node.js, Express.js.

- **Client Side:**
  - Browser: Latest version of Chrome, Firefox, or Safari.

1.4 Technology Used

- **Backend:** Node.js with Express.js for routing.
- **Database:** PostgreSQL for database .
- **Frontend:** EJS for rendering dynamic content.
- **CSS:** Custom CSS for styling and responsive design.

1.5 Brief Description of Technology

1. Node.js

Description: Node.js is a JavaScript runtime built on Chrome's V8 engine. It allows you to run JavaScript on the server side, enabling full-stack development with JavaScript as the common language for both frontend and backend. Node.js is known for its non-blocking, event-driven architecture, making it ideal for I/O-heavy tasks like handling web requests, file system operations, and database interactions.

Usage in the Project: Node.js powers the backend, handling routing, business logic, and database operations for "Unreleased Beats". It processes requests for song uploads, album management, and song promotion functionalities.

2. Express.js

- **Description:** Express.js is a lightweight, unopinionated web application framework for Node.js. It simplifies building web servers by providing a rich set of features for handling HTTP requests, routing, middleware, and more.
- **Usage in the Project:** Express.js is used to handle routing in the "Unreleased Beats" project, including requests such as adding songs, managing albums, and handling user promotions. It also manages sessions for admin login and user authentication.

3. EJS (Embedded JavaScript Templates)

- **Description:** EJS is a simple templating engine that allows you to generate HTML markup with embedded JavaScript. It helps in dynamically rendering content on web pages using data passed from the server.
- **Usage in the Project:** EJS is used to dynamically render the views in the "Unreleased Beats" project, allowing pages like the song management dashboard, album creation forms, and promotion approval interfaces to display updated content based on data from the database.

4. PostgreSQL

- **Description:** PostgreSQL is an open-source, powerful relational database system. It supports complex queries, transactions, and is known for its strong emphasis on extensibility and standards compliance.
- **Usage in the Project:** PostgreSQL is the database used to store all data for "Unreleased Beats", including song metadata, album details, user data, and promotion statuses. It manages relationships between songs, albums, and promotions, ensuring data consistency and reliability.

5. Multer

- **Description:** Multer is a Node.js middleware used for handling multipart/form-data, primarily used for uploading files. It allows you to handle file uploads in an Express.js application.
- **Usage in the Project:** Multer is used to handle the uploading of song files and cover images in "Unreleased Beats". It stores these files in designated directories and ensures the proper association of file paths with songs or albums in the PostgreSQL database.

6. pg (PostgreSQL Client for Node.js)

- **Description:** pg is a popular PostgreSQL client for Node.js that provides a comprehensive set of tools to interact with PostgreSQL databases. It allows you to execute SQL queries, manage database connections, and handle database transactions.
- **Usage in the Project:** The pg library is used to connect the Node.js application to the PostgreSQL database. It facilitates the execution of queries for creating, updating, deleting, and retrieving songs, albums, and promotions from the database.

7. Git/GitHub

- **Description:** Git is a version control system that helps track changes in code, and GitHub is a cloud-based platform that hosts Git repositories for collaborative development.
- **Usage in the Project:** Git is used to track code changes in the project, and GitHub serves as the central repository where the codebase is stored, allowing for version control and collaboration.

CHAPTER 2: PROPOSED SYSTEM

2.1 Study of Similar Systems

This system is inspired by other music management platforms such as SoundCloud and Mixcloud. These platforms offer song upload and categorization features that "Unreleased Beats" seeks to simplify and enhance. More Similar System like marathidjs, mumbaiDjs

1. SoundCloud -

SoundCloud is one of the most popular platforms for musicians to upload, share, and promote their tracks. It provides a social network-like environment where artists can directly interact with their fans. Features include track uploads, the ability to create playlists (similar to albums), and built-in tools for promotions.

Key Features:

- Song Upload: Artists can upload tracks in various audio formats. They can categorise these tracks by genre, add metadata, and share them publicly or privately.
- Album and Playlist Management: SoundCloud allows users to organise their music into playlists, making it easy to categorise songs by mood, genre, or event.
- Promotion Tools: Artists can promote tracks using SoundCloud's promotional features such as reposts, likes, and comments. Additionally, the SoundCloud Pro plan allows artists to use monetization and advanced analytics tools.
- Community Interaction: SoundCloud offers a strong social interaction system, where users can follow artists, comment on tracks, and share content across other platforms.

Limitations:

- Generic Audience: SoundCloud caters to all types of musicians, not specifically DJs, so the platform does not offer specialised tools for DJs to manage unreleased beats or exclusive tracks.
- Crowded Platform: As SoundCloud is open to anyone, it can be difficult for artists to stand out unless they invest heavily in promotions.

3. Mixcloud

Mixcloud is a platform primarily focused on long-form audio content, such as DJ mixes, radio shows, and podcasts. It's widely used by DJs for streaming mixes and live sets, allowing users to build an audience based on their mixes.

Key Features:

- **Streaming Mixes:** Mixcloud allows DJs to upload long-form mixes and audio content. Users can stream this content but cannot download it unless authorised by the uploader.
- **Licensing and Royalties:** Mixcloud has in-built licensing agreements with music rights holders, ensuring that DJs are compensated for the use of their music and that uploaded tracks are licensed.
- **Fan Support:** Like other platforms, Mixcloud allows fans to support DJs through subscriptions, which give them access to exclusive mixes, early releases, and more.

Limitations:

- **No Direct Downloads:** Mixcloud is primarily a streaming platform, and downloading tracks is not encouraged. This makes it less attractive for DJs who want to offer downloadable tracks to their fanbase.
- **Limited Song Management:** Mixcloud is focused on DJ sets rather than individual tracks, making it difficult for DJs to manage unreleased songs in an organised manner.

2.2 Users of the System

- **Admin**: Manages the platform, approves or rejects song promotions, creates and edits albums, and categorises songs.
- **DJs**: Upload their unreleased songs, view albums , download albums, and submit songs for promotion.
- **Visitors**: View and listen to promoted songs.

CHAPTER 3: ANALYSIS AND DESIGN

3.1 System Requirements

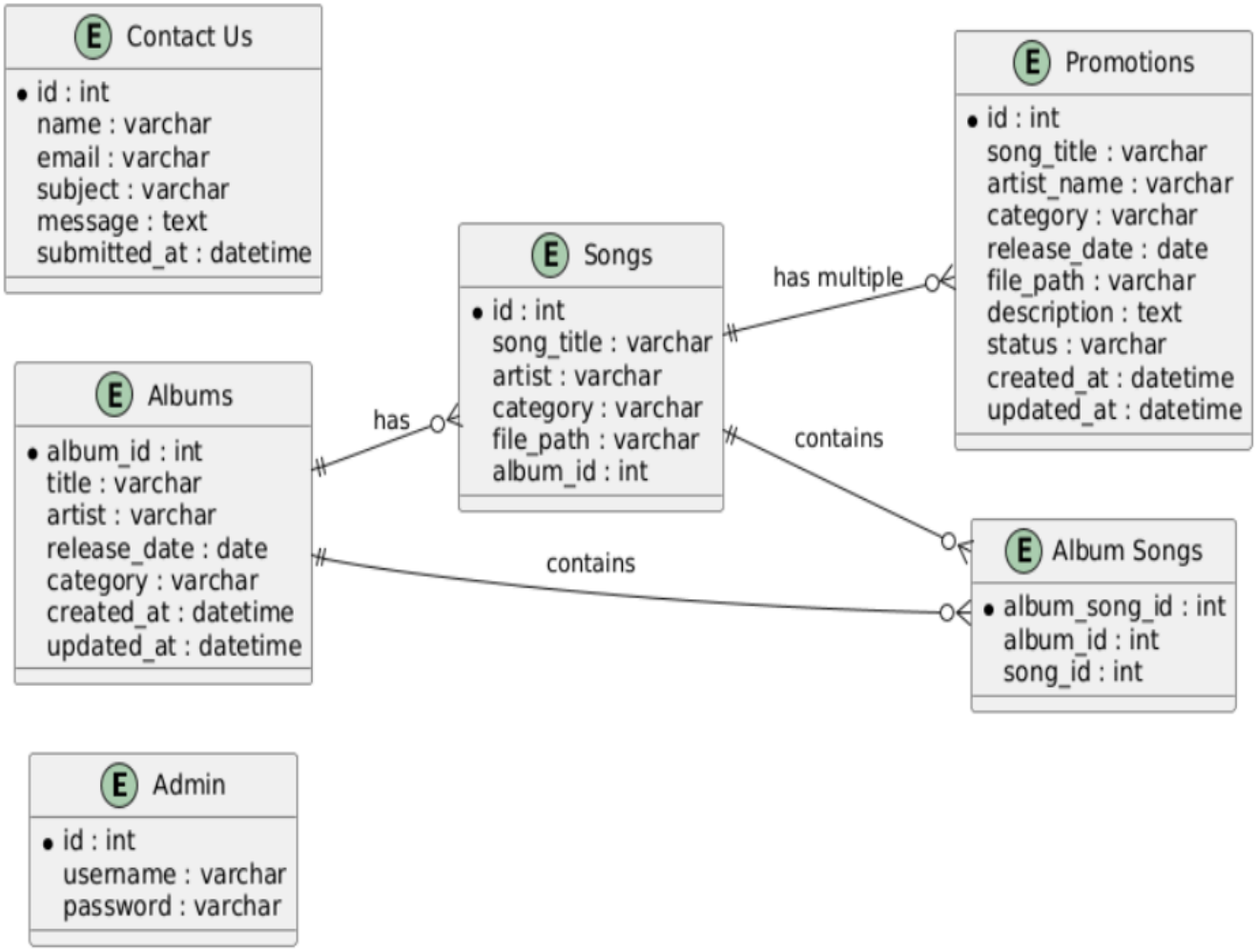
Functional Requirements:

1. Admin Login and Authentication: Admin must be able to log in securely.
2. Song Upload: Admin can upload unreleased songs.
3. Album Management: Admin can create albums and add songs to them.
4. Promotions: DJs/Users/Visitors can submit songs for promotion, and admins can approve or reject promotion requests.

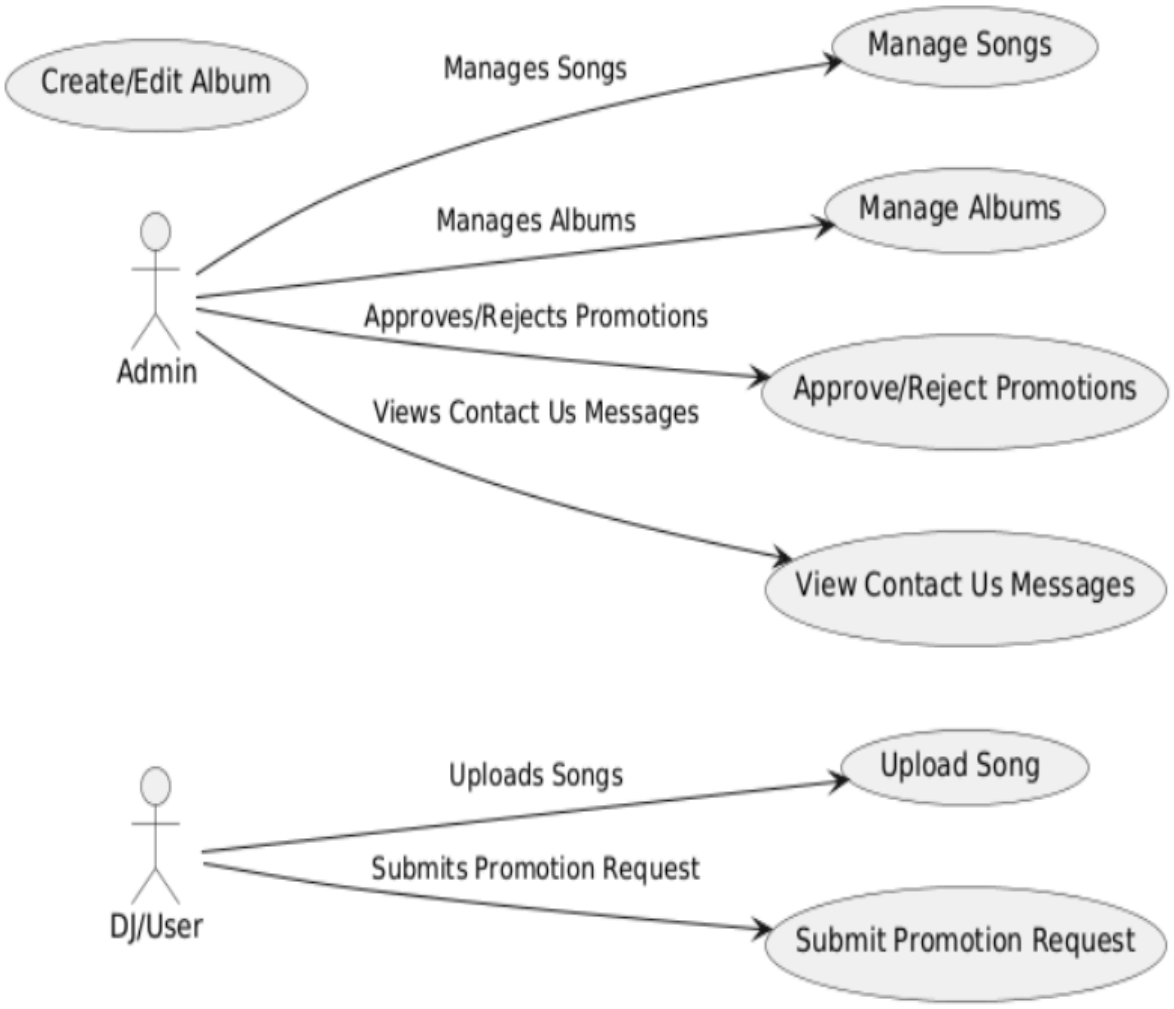
Non-Functional Requirements:

1. Usability: The system should be intuitive and easy to use for DJs.
2. Availability: The system must be available 24/7.
3. Security: All song and album data should be stored securely, with access limited to authorised users.

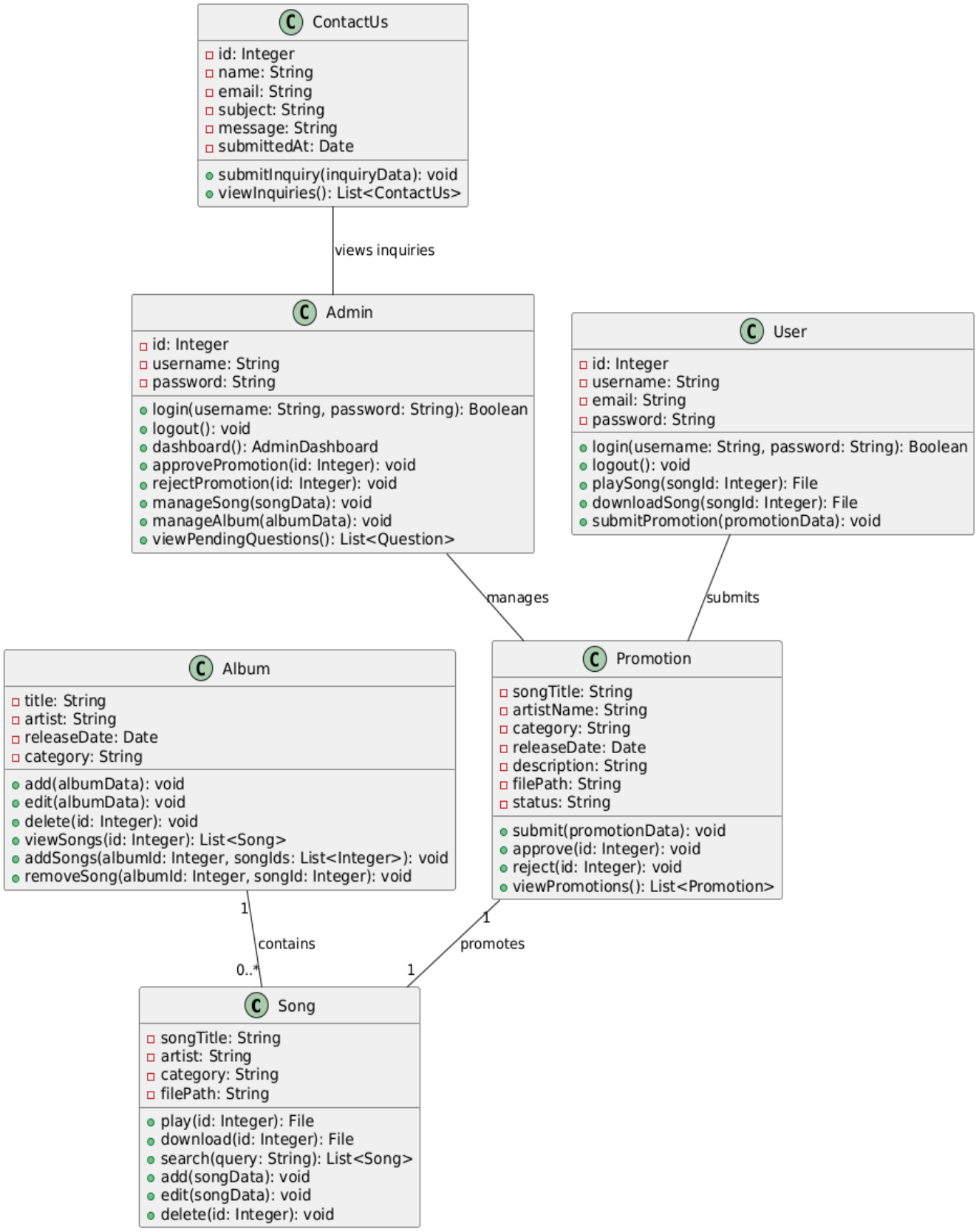
3.2 Entity Relationship Diagrams (ER


Contact Us
• id : int
name : varchar
email : varchar
subject : varchar
message : text
submitted_at : datetime
Promotions
• id : int
song_title : varchar
artist_name : varchar
category : varchar
release_date : date
file_path : varchar
description : text
status : varchar
created_at : datetime
updated_at : datetime
Songs
• id : int
song_title : varchar
artist : varchar
category : varchar
file_path : varchar
album_id : int
has multiple
Albums
• album_id : int
title : varchar
artist : varchar
release_date : date
category : varchar
created_at : datetime
updated_at : datetime
has
contains
contains
Album Songs
• album_song_id : int
album_id : int
song_id : int
Admin
• id : int
username : varchar
password : varchar

3.3 Use Case Diagram

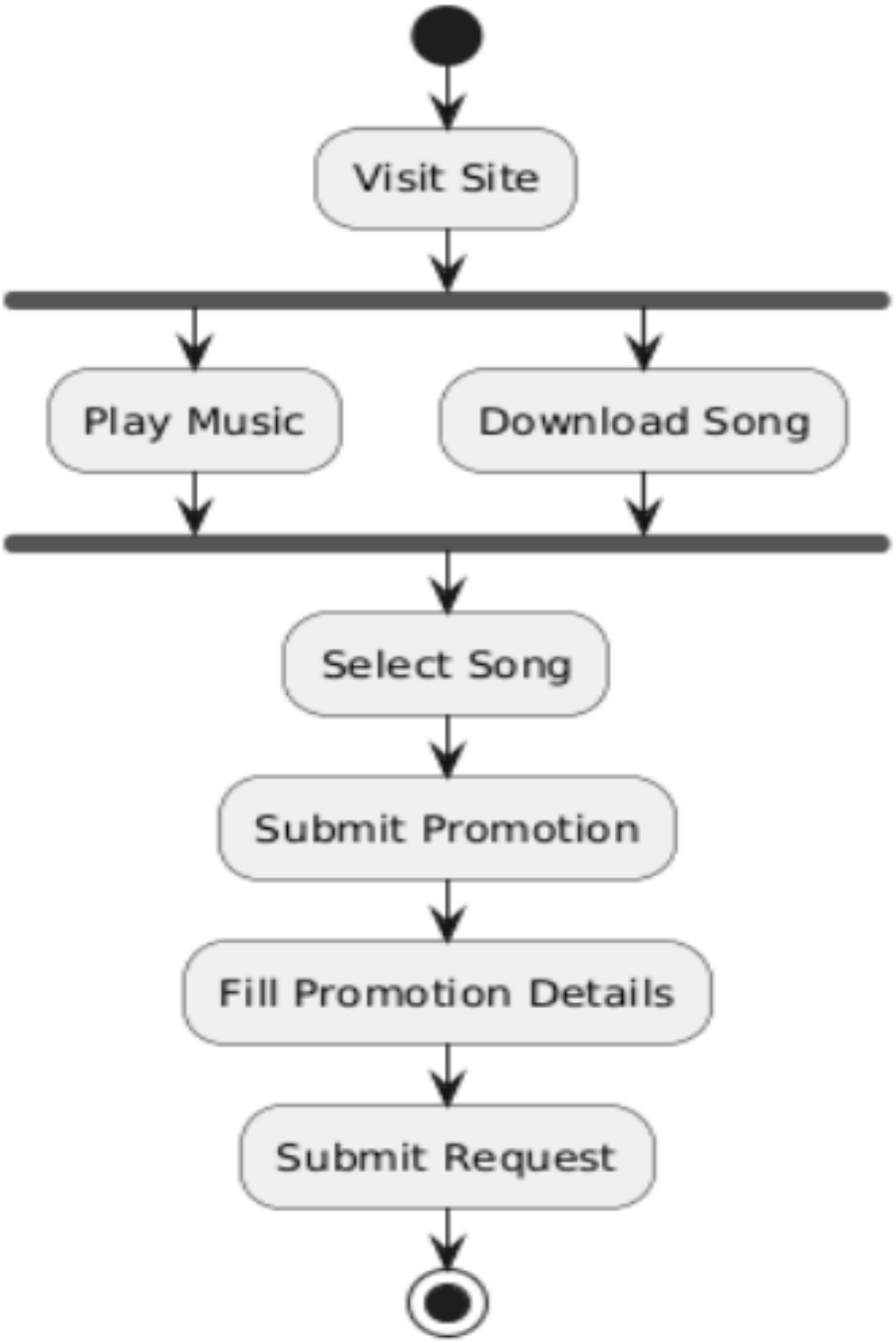

Create/Edit Album
Manages Songs
Manage Songs
Manages Albums
Manage Albums
Admin
Approves/Rejects Promotions
Approve/Reject Promotions
Views Contact Us Messages
View Contact Us Messages
Uploads Songs
Upload Song
Submits Promotion Request
DJ/User
Submit Promotion Request

3.4 Class Diagram


ContactUs
id: Integer
name: String
email: String
subject: String
message: String
submittedAt: Date
submitInquiry(inquiryData): void
viewInquiries(): List<ContactUs>
views inquiries
Admin
id: Integer
username: String
password: String
login(username: String, password: String): Boolean
logout(): void
dashboard(): AdminDashboard
approvePromotion(id: Integer): void
rejectPromotion(id: Integer): void
manageSong(songData): void
manageAlbum(albumData): void
viewPendingQuestions(): List<Question>
User
id: Integer
username: String
email: String
password: String
login(username: String, password: String): Boolean
logout(): void
playSong(songId: Integer): File
downloadSong(songId: Integer): File
submitPromotion(promotionData): void
manages
submits
Album
title: String
artist: String
releaseDate: Date
category: String
add(albumData): void
edit(albumData): void
delete(id: Integer): void
viewSongs(id: Integer): List<Song>
addSongs(albumId: Integer, songIds: List<Integer>): void
removeSong(albumId: Integer, songId: Integer): void
Promotion
songTitle: String
artistName: String
category: String
releaseDate: Date
description: String
filePath: String
status: String
submit(promotionData): void
approve(id: Integer): void
reject(id: Integer): void
viewPromotions(): List<Promotion>
1
contains
0..*
1
promotes
1
Song
songTitle: String
artist: String
category: String
filePath: String
play(id: Integer): File
download(id: Integer): File
search(query: String): List<Song>
add(songData): void
edit(songData): void
delete(id: Integer): void

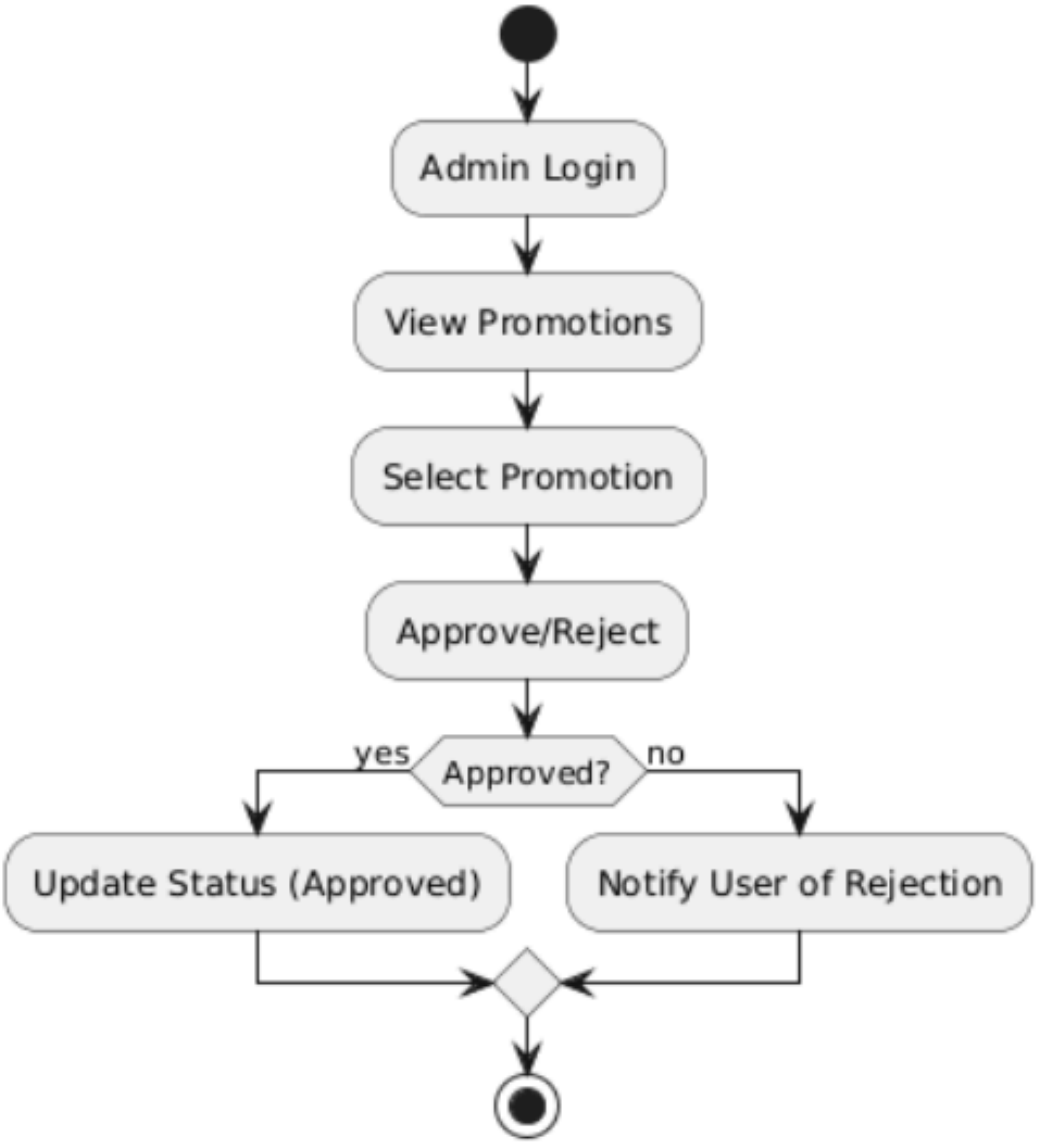
3.5Activity Diagrams

Activity Diagram : User



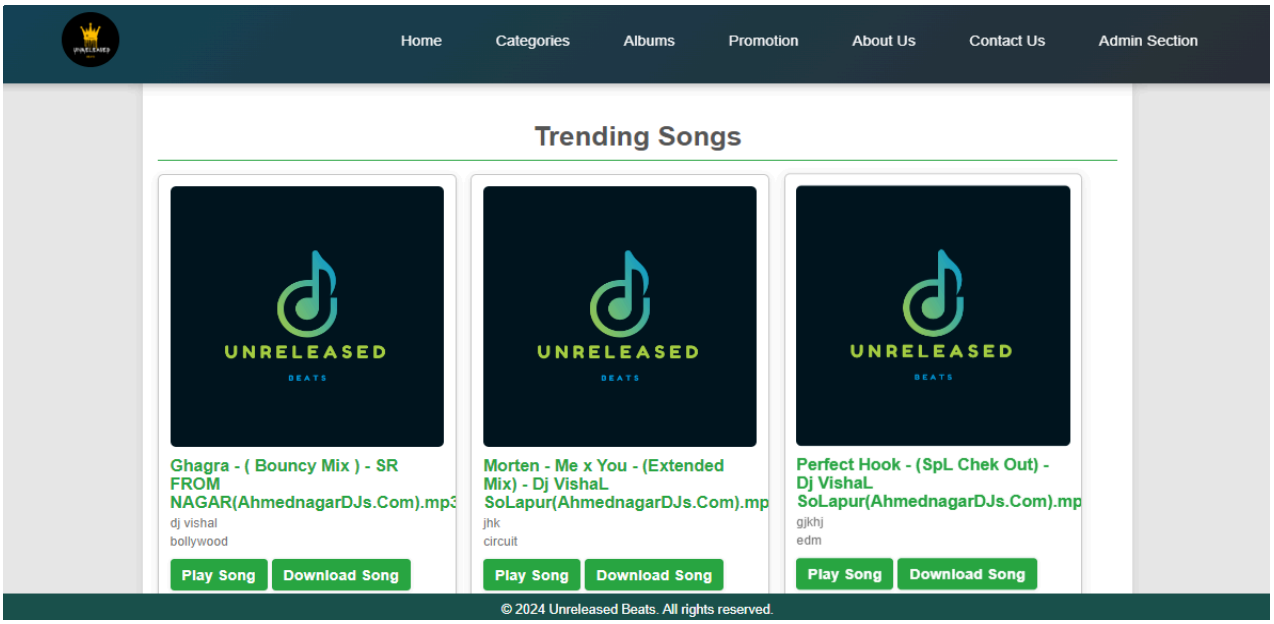
Activity Diagram : Admin

Activity Diagram : Promotions

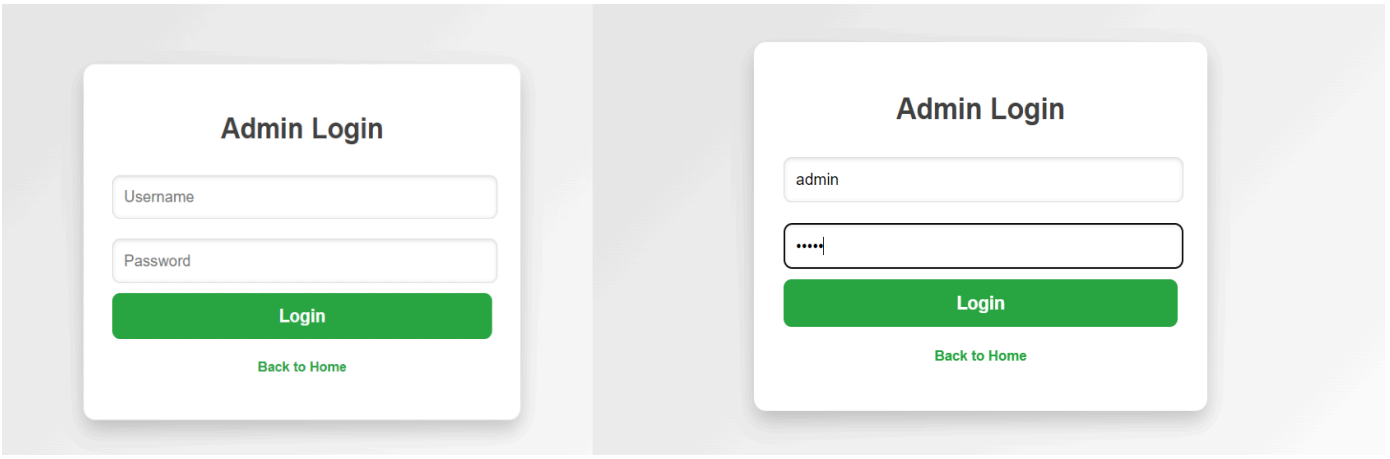


3.6 Sample Input and Output Screen

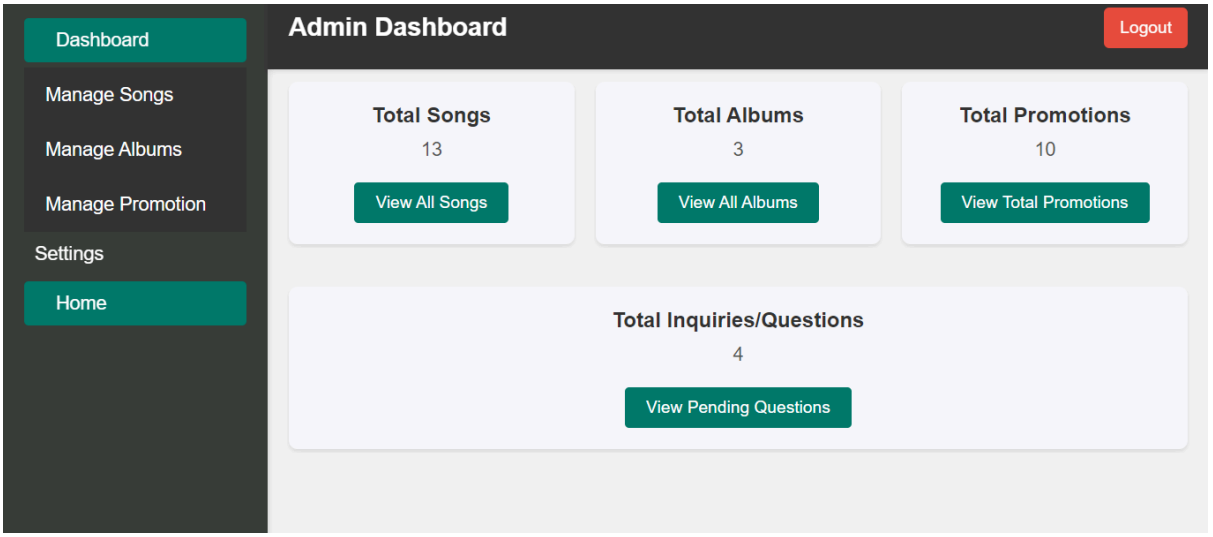
1. Home Screen



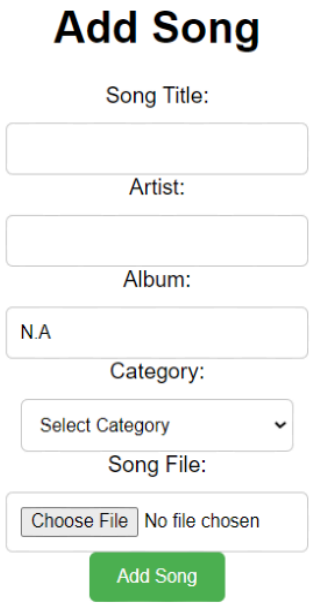
2. Admin login screens

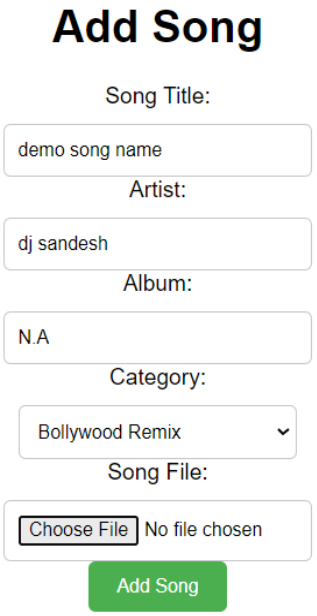


3. Admin Dashboard (After Login)

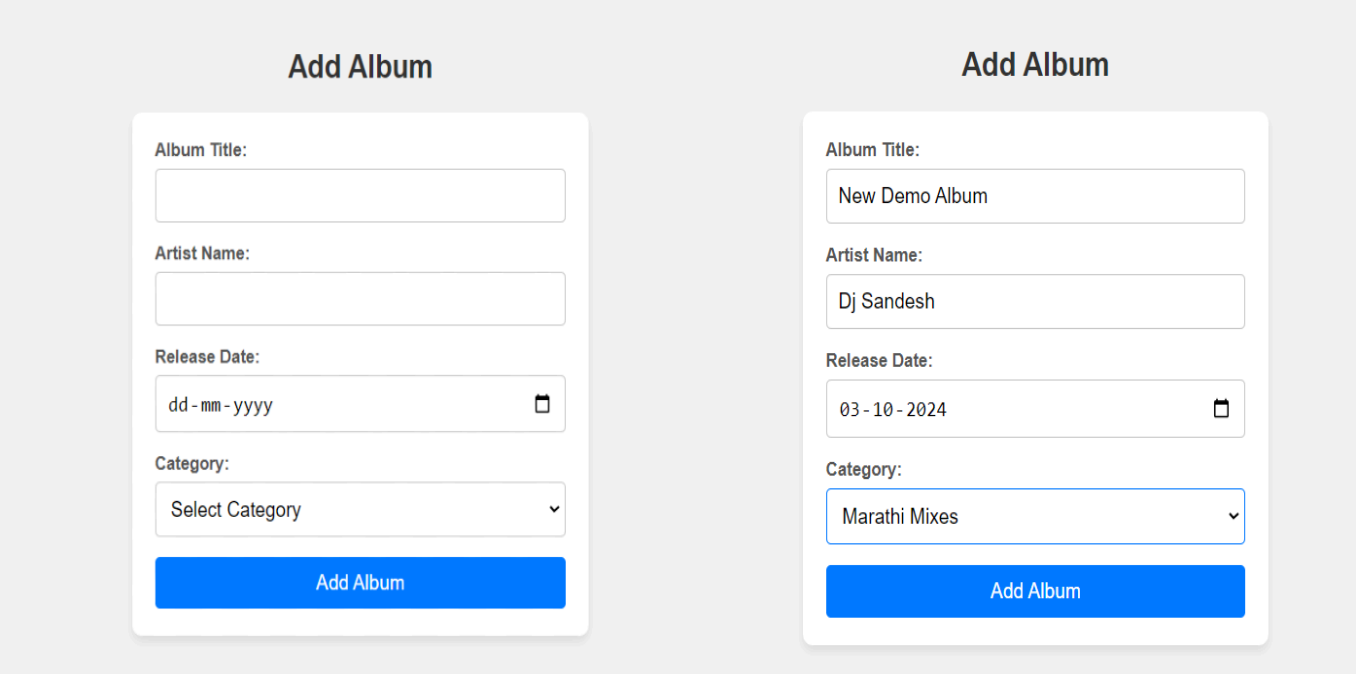


4. Add Song Page (Admin Login)

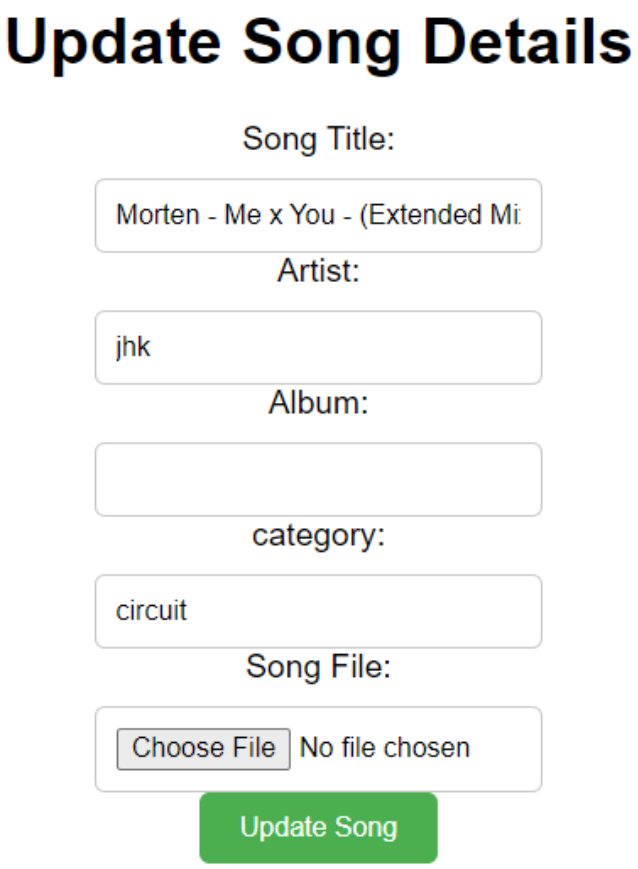




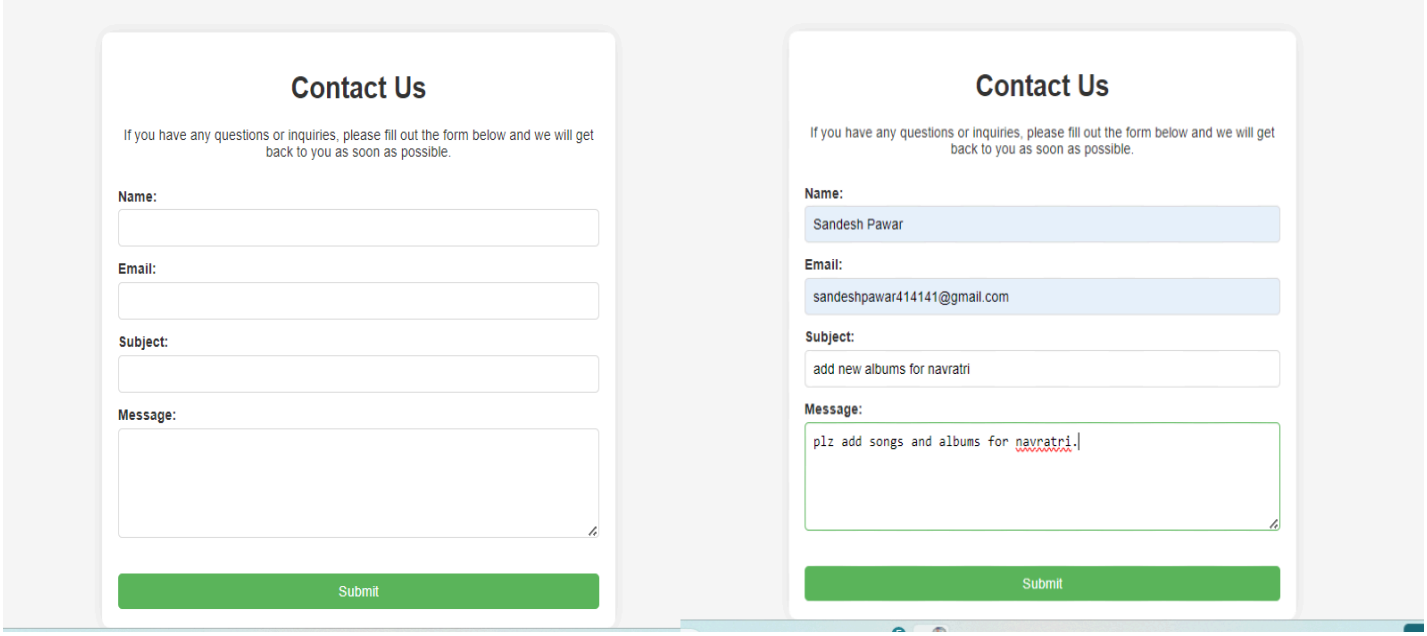
5. Add Album Page (admin)



6. Update Song (Admin)



7. Contact us Form (User)

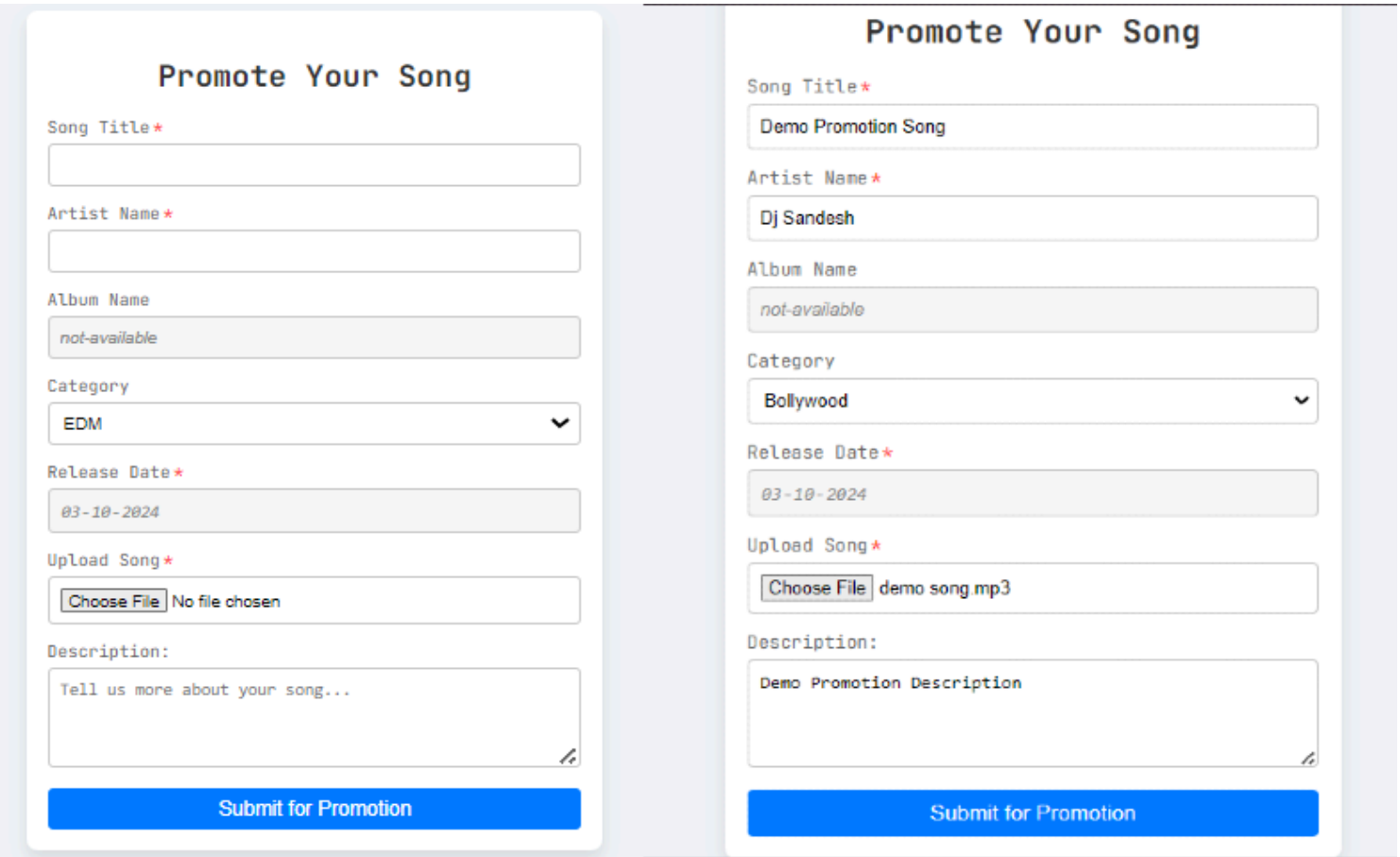


8. All Contacts/Suggestion Views (admin)

View All Contacts

ID	NAME	EMAIL	MESSAGE	DATE SUBMITTED
10	Sandesh Pawar	sandeshpawar414141@gmail.com	plz add songs and albums for navratri.	3/10/2024
9	Gans	gans@gmail.com	afasfadsfasfdas	19/8/2024
8	Jsjsj	sandeshpawar414141@gmail.com	Sisiiss	12/8/2024
7	Sudesh Vyas	sandeshpawar414141@gmail.com	faadsfasfad	12/8/2024
6	Sudesh Vyas	sandeshpawar414141@gmail.com	fdsfsafads	12/8/2024

9. Song Promotion Form (User)



4. Coding

Index.js

1. Code For Song Upload Route (Express.js)

- ***This route allows Admin to upload songs to the uploads folder on the server, storing the song data in PostgreSQL.***

```
const express = require('express');
const multer = require('multer');
const path = require('path');
const { Pool } = require('pg');

app.post("/admin/add-song", upload.single('song'), async (req, res) => {
   const { album, artist, category } = req.body;
   const songFile = req.file;

   if (!songFile) {
      return res.status(400).json({ error: 'No song file uploaded' });
   }

   const filePath = path.join('uploads', songFile.filename); // Use the filename
generated by multer
   const songTitle = songFile.originalname;

   try {
      // Insert song metadata into the database
      await db.query(
         'INSERT INTO songs (song_title, artist, category, file_path) VALUES ($1, $2,
$3, $4)',
         [songTitle, artist, category, filePath]
      );

      // res.status(201).json({ message: 'Song added successfully' });
      // below is success message for the admin dashboard
      req.session.successMessage = 'Song added successfully!';

      // res.redirect('dashboard',);
      res.send(`
            <html>
               <body>
                  <script>
                     alert("Song Added Successfully....");
```

```
                    window.location.href = "/admin/dashboard"; // Redirect to the home
page
                </script>
              </body>
            </html>` )
    } catch (err) {
      console.error('Error adding song:', err);
      res.status(500).json({ error: 'Error adding song' });
    }
});
```

2. Displaying Songs And Albums

- ***This route handles the displaying songs and albums at home page***

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home | Unreleased Beats</title>
  <link rel="stylesheet" href="styles/style.css">
</head>

<body class="jb-mono">
  <%- include("partials/header.ejs") %>

  <section>
    <h2 class="song-heading">Trending Songs</h2>
    <div class="songs-container">
      <% if (allSongs && allSongs.length > 0) { %>
        <% allSongs.forEach(function(song) { %>
          <div class="song-card">
            <img src="/images/thumbnail.png" alt="logo">
            <h3><%= song.song_title %></h3>
            <p><%= song.artist %></p>
            <p><%= song.category %></p>
            <p><%= song.genre %></p>
            <a href="/play/<%= song.id %>" class="button">Play Song</a>
            <a href="/download/<%= song.id %>" class="button">Download Song</a>
          </div>
```

```
      <% }); %>
    <% } else { %>
      <p>No songs found.</p>
    <% } %>
  </div>

  <div class="pagination">
    <% for (let i = 1; i <= totalPages; i++) { %>
      <a href="/?page=<%= i %>" class="<%= currentPage === i ? 'active' : ''
%>"><%= i %></a>
    <% } %>
  </div>
 </section>

 <section>
  <h2>Albums</h2>
  <div class="albums-container">
    <% if (allAlbums && allAlbums.length> 0) { %>
      <% allAlbums.forEach(function(album) { %>

        <div class="album-card">
          <img src="/images/thumbnail.png" alt="<%= album.title %> Cover">

          <h3>
            <%= album.title %>
          </h3>
          <p>
            <%= album.artist %>
          </p>
          <p>
            <%= album.release_date %>
          </p>
          <p>
            <%= album.category %>
          </p>

          <a href="/user/albums/viewAlbum/<%= album.album_id %>"
class="button"> View Album</a>
        </div>

        <% }); %>
```

```
        <% } else { %>
                        <p>No albums found.</p>
                        <% } %>
         </div>
         </section>

         <%- include("partials/footer.ejs") %>
        </body>

        </html>
```

3. Route for Deleting song

- ***This route/code handles the Deletion of Song***

```
app.get("/admin/delete-song", async (req, res) => {
   if (req.session.admin) {
      const result = await db.query("SELECT * FROM songs");
      const allSongs = result.rows;
      res.render('admin/deleteSong.ejs', { allSongs });
   }
})

app.get("/delete/:id", async (req, res) => {
   let songToDel = parseInt(req.params.id);

   console.log(songToDel);
   try {
      if (req.session.admin) {
         const sql = "DELETE FROM songs WHERE id = $1;";
         const values = [songToDel];
         await db.query(sql, values);
         res.redirect('/admin/dashboard');

      }
      else {
         res.render('admin/adminForm');
```

```
    }
    // res.json(songToDel, "delete successs")
  } catch (error) {
    console.log(error);
  }
```

5. Route for updating song info

- ***This route/code handles the updation of song***

```
app.get("/admin/edit-song", async (req, res) => {
    if (req.session.admin) {
        const result = await db.query("SELECT * FROM songs");
        const allSongs = result.rows;
        res.render("admin/editSong.ejs", { allSongs });

    }
})
app.get("/edit/:id", async (req, res) => {
    let songIdToUpdate = parseInt(req.params.id);
    let songTitle, songArtist, songAlbum, songcategory, songFilePath;
    let song = "SELECT * FROM songs WHERE ID = $1";
    let values = [songIdToUpdate];
    let result = await db.query(song, values);
    console.log(result.rows[0]);


    songTitle = result.rows[0].song_title;
    songAlbum = result.rows[0].album;
    songArtist = result.rows[0].artist;
    songcategory = result.rows[0].category;
    // filePath = result.rows.file_path;

    res.render("admin/updateForm.ejs", { songIdToUpdate, songTitle, songAlbum,
songArtist, songcategory });

})
app.post("/admin/update-song/:id", upload.single('song'), async (req, res) => {
    console.log(req.params.id);
```

```
    let songIdToUpdate = parseInt(req.params.id);
    let { songTitle, songArtist, songAlbum, songcategory } = req.body;

    // Get the file if it exists
    const songFile = req.file;
    let songFilePath = songFile ? path.join('uploads', songFile.filename) : null;

    // SQL query to update the song
    let updateSongQuery = `UPDATE songs
            SET
                song_title = $1,
                artist = $2,
                album = $3,
                category = $4,
                file_path = COALESCE($5, file_path)
            WHERE
                id = $6`;

    try {
        const getSongQuery = "SELECT * FROM songs WHERE id=$1";
        // Get the existing song record
        let result = await db.query(getSongQuery, [songIdToUpdate]);
        if (result.rows.length === 0) {
            return res.status(404).send("Song not found");
        }

        // Update the song record, keeping the original file path if no new file was
uploaded
        await db.query(updateSongQuery, [songTitle, songArtist, songAlbum,
songcategory, songFilePath, songIdToUpdate]);

        res.status(200).send("Song updated successfully");
    } catch (error) {
        console.error(error);
        res.status(500).send("An error occurred while updating the song");
    }
});
```

6. Code For Adding Album

- ***This is route for adding album***

```
app.get("/admin/add-album", (req, res) => {
   if (req.session.admin) {
      res.render("admin/addAlbum.ejs");
   }
})

// post request handler
app.post("/admin/add-album", async (req, res) => {

   if (req.session.admin) {
      const title = req.body.title;
      const artist = req.body.artist;
      const releaseDate = req.body.releaseDate;
      const category = req.body.category;

      console.log(title, artist, releaseDate, category);

      try {
         // Insert album into the database
         await db.query(
            'INSERT INTO albums (title, artist, release_date, category) VALUES ($1,
$2, $3, $4)',
            [title, artist, releaseDate, category]
         );

         req.session.success = 'Album added successfully!';
         res.redirect("/admin/dashboard");
      } catch (error) {
         console.error('Error inserting album:', error);
         req.session.error = 'There was an error adding the album.';
         res.redirect("/admin/dashboard");
      }
   }

})
```

7. Code for handling delete album functionality

```
app.get("/admin/delete-album", async (req, res) => {

   let result = await db.query("SELECT * FROM albums");
   let allAlbums = result.rows;

   res.render('admin/deleteAlbum.ejs', { allAlbums });
   console.log(allAlbums);
})
// /admin/deleteAlbum
app.get("/admin/deleteAlbum/:id", async (req, res) => {
   let deleteID = req.params.id; // Access the ID parameter from the URL
   console.log(deleteID);

   if (req.session.admin) {
      try {
         let sql = "DELETE FROM albums WHERE album_id = $1";
         let values = [deleteID];
         let query = await db.query(sql, values);
         console.log(query.rows);
         res.send(`
            <html>
               <body>
                  <script>
                     alert("Album Deleted Successfully....");
                     window.location.href = "/admin/dashboard"; // Redirect to the home
page
                  </script>
               </body>
            </html>`);
      } catch (error) {
         console.log(error);
      }
   }
});
```

5. Testing

5.1 Test Strategy

1. Functional Testing

Functional testing ensures that the system functions according to the specified requirements. Each function of the system is tested by providing appropriate inputs and verifying the output against the expected result.

- **Song Upload**: Validate that DJs can upload songs successfully, and the song gets saved in the database with the correct file path.
- **Album Creation**: Verify that DJs can create albums, add songs to albums, and edit album details.
- **Promotion Submission**: Check that DJs/User can submit songs for promotion, and the promotion status is updated when an admin approves or rejects the submission.
- **Admin Management**: Ensure that admins can manage songs, albums, and promotions as required.

---

2. Unit Testing

Unit testing focuses on testing individual units or components of the system. Each unit test verifies the behaviour of a small part of the system, such as a function or route, in isolation from the rest.

Test Case No : 1

Test Case Description: Test Admin login functionality
Preconditions: admin is registered and credentials are available
Test Steps:

1. Navigate to the admin login page.
2. Enter a valid username(*admin*) and password (*admin*).
3. Click the "Login" button.
   Expected Output: User is logged in and redirected to the admin dashboard

Actual Result: Redirected to Admin Dashboard, if credentials are wrong then its redirect to login failed page

Status: Pass

Test Case No : 2

Test Case Description: Test song upload with a valid file

Preconditions: Admin is logged in and navigated to the upload song page

Test Steps:

1. Choose a valid audio file (MP3 format).
2. Fill in song details like title and artist.
3. Click the "Add Song" button.

   Expected Output: Song is uploaded successfully

   Actual Result:Song uploaded successfully with mp3 filetype

   Status: Pass

Test Case No : 3

Test Case Description: Test album creation with valid details

Preconditions: Admin is logged in and navigated to the create album page

Test Steps:

1. Enter the album title, artist, release date, and category.
2. Click the "Create Album" button.

   Expected Output: Album is created successfully

   Actual Result: Album Created Successfully
3. **Status**: Pass

Test Case No : 4

Test Case Description: Test promotion approval by admin

Preconditions: Admin is logged in and navigated to the promotions page

Test Steps:

1. Click the "Approve" button for a pending promotion.
2. Confirm the approval action.
   Expected Output: Promotion status is updated to "Approved"
   Actual Result:Promotion get Approved
3. **Status**: Pass

Test Case No : 5

Test Case Description: Test song upload without authentication

Preconditions: Admin is not logged in

Test Steps:

1. Navigate to the upload song page.
2. Try to upload a song without logging in.
   Expected Output: User is redirected to the login page
   Actual Result: Admin login page appeared when I tried to upload the song.
   Status: Pass

Test Case No: 6

Test Case Description: Test system on multiple browsers (Chrome,MicrosoftEdge, Brave, OperaGX,)

Preconditions: User has access to different browsers

Test Steps:

1. Open the application in Google Chrome and navigate through Portal .
2. Repeat the same process in MicroSoft Edge.
3. Repeat the same process in Brave (Ad Blocking Browser).
4. Repeat the same process in Opera GX (Gaming Browser)

   Expected Output: Application behaves Same across different browsers

   Actual Result:Application is constant across different browsers
5. **Status**: Pass

CHAPTER 6: LIMITATIONS OF THE PROPOSED SYSTEM

“ The "Unreleased Beats" project provides a comprehensive platform for managing and promoting songs, particularly for DJs; there are certain limitations inherent to the current version of the system. These limitations are identified based on functionality, technical constraints, user feedback, and scalability considerations. A detailed understanding of these limitations can help in planning future improvements and expansions. “

6.1 Limited Scalability

The current architecture of the "Unreleased Beats" system is designed for moderate traffic and user engagement. As the platform grows in terms of users, songs, and promotions, the underlying infrastructure may face challenges in handling increased load. The main limitations related to scalability include:

- **Server Load**: With an increase in the number of users uploading and streaming songs, the server may experience performance bottlenecks. This can lead to slower page loads and degraded performance, especially if proper load balancing or horizontal scaling is not implemented.
- **Database Growth**: As the PostgreSQL database grows with more data (e.g., songs, albums, promotions), query performance may decline. The current setup does not support advanced database partitioning or sharding, which could become necessary as data size increases.

6.2 Limited Payment Integration

The platform intends to introduce a payment system for song promotions. However, the

initial implementation may have certain limitations:

- **Payment Gateway Support**: Currently, only basic payment gateway integration (e.g., Google Pay or Phonepe) is supported. The platform lacks support for region-specific payment methods, which may restrict users from certain geographical areas from promoting their songs.
- **Transaction Security**: Although basic encryption and security measures are in place, advanced features like fraud detection, multi-step authentication for transactions, and seamless handling of payment disputes have not been fully integrated.

6.3 Admin Customization of Categories

While the admin can manage song categories, there are still limitations in terms of the flexibility and granularity of this feature:

- **Limited Category Management**: Although admins can create and manage categories for songs and albums, there are restrictions on subcategories, which might be necessary as the song library grows.
- **Lack of Advanced Filtering**: Users and admins have limited options for filtering and sorting content. Advanced filtering based on multiple criteria (e.g., genre, artist popularity, and date) is not yet implemented.

6.4 Lack of Real-Time Features

The platform does not currently support real-time updates for certain features. For instance:

- **No Real-Time Notifications**: When a promotion is approved, or a song is added to an

album, users need to refresh their page to see updates. Real-time notifications via websockets or push notifications are not yet implemented.

- **No Real-Time Chat**: A real-time chat feature between DJs, users, and admins is not available. This could enhance engagement and collaboration on the platform.

6.5 Basic User Experience and Interface Design

While the current user interface is functional, it lacks advanced, polished design elements, particularly for mobile (small Screens) and tablet users:

- **Limited Mobile Responsiveness**: Although the platform is designed to be mobile-friendly, certain UI components, such as song cards and admin dashboards, may not render perfectly on smaller screens. This limitation can affect user experience on mobile devices.
- **Visual Customization**: Users do not have the option to customise their profiles or the appearance of the platform. Providing customization features (e.g., themes, fonts, colour schemes) could greatly enhance the user experience.

6.6 Security and Privacy Concerns

Even though the platform implements standard security practices, there are some limitations regarding data protection and privacy:

- **User Authentication**: The system does not currently use two-factor authentication or other advanced security features. This increases the risk of unauthorised access.
- **Song and Album Ownership**: While the platform allows users to upload songs, there are no stringent checks on song ownership. This could lead to copyright issues if unauthorised content is uploaded.

- **Data Encryption**: Sensitive data, such as user passwords, are stored using encryption, but there are no end-to-end encryption mechanisms for file uploads or messages between users and admins.

6.7 Manual Promotion Approval Process

The current promotion system requires manual approval by the admin. While this ensures that promotions are carefully reviewed, it also introduces delays and scalability issues:

- **Admin Workload**: As the number of promotion requests increases, the manual approval process will become time-consuming and inefficient. Automating certain checks (e.g., song quality, adherence to guidelines) could reduce the admin's burden.

Chapter 7 - Proposed Enhancement

"Unreleased Beats" platform continues to grow and evolve, several enhancements can be introduced to improve the system's functionality, user experience, scalability, and overall performance. These proposed enhancements aim to address the current limitations of the system (as outlined in Chapter 6) and to provide additional value to users and administrators.

7.1 Scalability Improvements

As the platform grows, ensuring its scalability is crucial. Enhancements related to the platform's scalability include:

- Cloud-Based Infrastructure: Transition the platform to a cloud-based infrastructure (e.g., AWS, Azure, or Google Cloud) to enable auto-scaling and handle increasing server load. This will ensure that the platform can support a large number of simultaneous users without performance degradation.
- Database Optimization: Implement database optimization techniques such as indexing, partitioning, and database sharding to improve the performance of queries and reduce the risk of bottlenecks as data grows. Caching mechanisms (e.g., Redis) can also be integrated to store frequently accessed data in memory.

7.2 Payment System Enhancement

The current limitations of the payment system can be addressed by integrating advanced payment features to streamline the promotion process and make it accessible to a broader user base:

- Multiple Payment Gateways: In addition to Google Pay or Phonepe, integrate other payment gateways that support international transactions (e.g., Razorpay, Square). This will make it easier for DJs from different regions to submit payments for song promotions.
- Subscription Plans: Introduce subscription models for song promotions. DJs can choose from different subscription plans (e.g., basic, premium) for additional promotion benefits, such as extended promotion periods or priority listing on the platform.
- Automated Invoicing: Add a feature to generate automatic invoices for payments made for song promotions. This will provide users with proper documentation for their payments and improve transparency.

7.3 Advanced Category Management

To enhance the flexibility and granularity of managing songs and albums, the following category management improvements can be introduced

- **Subcategories and Tags**: Allow admins to create subcategories within main categories (e.g., within "Hip-Hop," subcategories like "Trap" or "Trances" can be added). Additionally, introduce a tagging system where DJs can add relevant tags to songs (e.g., "Circuit mix ", "Festival Mix") to improve the discoverability of tracks.
- **Dynamic Filtering and Sorting**: Introduce dynamic filtering and sorting options for users to search and filter songs by multiple criteria (e.g., genre, artist, release date, popularity). This will make it easier for users to find songs that meet specific requirements.

7.4 Real-Time Features

Real-time capabilities can enhance user engagement and ensure the platform feels more responsive and interactive

- **Real-Time Notifications**: Implement real-time notifications using WebSockets or push notifications to alert users of key events, such as promotion approvals, new song uploads, or playlist updates. This will enhance communication between users and admins and provide immediate feedback to DJs on their actions.
- **Live Chat Support:** Add a live chat feature to allow DJs to communicate directly with the support team or other users in real time. This can be used to resolve queries, request assistance, or collaborate with other DJs on track promotion strategies.
- **Real-Time Song Analytics:** Provide real-time analytics for song performance. DJs can view live updates on how many times their songs have been played, downloaded, or promoted, which will help them make more informed decisions about future releases.

7.5 Improved User Interface and Experience

Improving the platform's user interface and experience (UI/UX) will make it more intuitive, engaging, and accessible:

- **Responsive Design Enhancements:** Refine the mobile responsiveness of the platform, ensuring that all UI components (e.g., song cards, admin dashboards) render correctly on different devices and screen sizes. This will enhance the experience for mobile and tablet users.
- **Dark Mode:** Introduce a dark mode option for users, which will allow them to switch to a darker theme for better readability in low-light environments. This is a popular feature among modern web platforms.

- **Customizable User Profiles:** Allow DJs to customise their profiles with personalised information (e.g., bio, profile picture, social media links). Additionally, provide users with the ability to customise their dashboard themes (e.g., colours, fonts) to create a more personalised experience.
- **Enhanced Visual Design:** Update the visual design of the platform with more engaging animations, modern typography, and better visual hierarchy to make navigation simpler and more enjoyable.

7.6 Security Enhancements

Ensuring the security of user data and platform integrity is crucial as the user base grows:

- **Two-Factor Authentication (2FA):** Introduce two-factor authentication for admins and DJs to add an additional layer of security during login. This will reduce the risk of unauthorised access to user accounts.
- **End-to-End Encryption for File Uploads:** Implement end-to-end encryption for song and file uploads to ensure that sensitive content is transmitted securely between users and the server.
- **Advanced Access Controls:** Implement role-based access control (RBAC) to define specific access levels for different user roles (e.g., DJs, admins, moderators). This will ensure that only authorised users can perform sensitive actions like promotion approvals or album deletions.

7.8 Social Media Integration

Enhancing the platform's integration with social media platforms will increase user engagement and song promotion opportunities:

- **Social Media Sharing:** Add social media sharing buttons (e.g., Facebook, Twitter, Instagram) to allow DJs and users to share songs, albums, or promotions directly from the platform. This will boost visibility and increase song discoverability.
- **Social Media Login:** Implement OAuth-based login, allowing users to sign in using their existing social media accounts (e.g., Google, Facebook). This will streamline the login process and make it more convenient for users to access the platform.