

**MINISTRY OF EDUCATION AND TRAINING
CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION
TECHNOLOGY**



FINAL PROJECT REPORT

CT313H: WEB TECHNOLOGIES AND SERVICES

MUZIK ONLINE

Student ID 1: B2206004

Student Name 1: Tran Thi Kim Phung

Student ID 2: B2206005

Student Name 2: Huynh Tu Phuong

FINAL PROJECT REPORT

SEMESTER 3, ACADEMIC YEAR: 2024-2025

CT313H: WEB TECHNOLOGIES AND SERVICES

- **Project/Application name:** MUZIK
- **Youtube link:** <https://www.youtube.com/watch?v=MjR5t9blds4>
- **GitHub link (for both frontend and backend):**
 - <https://github.com/24-25Sem3-Courses/ct313hm01-project-hfu186>
- **Student ID 1:** B2206005
- **Student Name 1:** Huynh Tu Phuong
- **Student ID 2:** B2206004
- **Student Name 2:** Tran Thi Kim Phung
- **Class/Group Number (e.g, CT313HM01):** CT313HM01

I. Introduction

- **Description:** This is a web-based music streaming application inspired by Spotify. It allows users to discover, play, and manage music online. Key features include user authentication, playlist creation, song favorites, album and artist management, and audio file uploading. The system supports different user roles such as guests, registered users, and administrators, and provides a RESTful API documented with OpenAPI standards.
- Functions:
 - Playlist Management**
 - Create, update, delete, and manage playlists.
 - Add cover images and a list of songs to each playlist.
 - Favorites & Likes**
 - Mark songs as favorites.
 - View your liked songs for quick access and playback.
 - Album & Artist Management**
 - Browse and search albums by name or artist.
 - Manage artist profiles including name, avatar, and biography.

- Each album can contain multiple songs and be linked to specific artists.

Advanced Search & Browse

- Search for songs, albums, artists, or playlists by keyword.
- Get suggestions based on genres, trends, or recent activity.

Audio Upload (Admin & Artist)

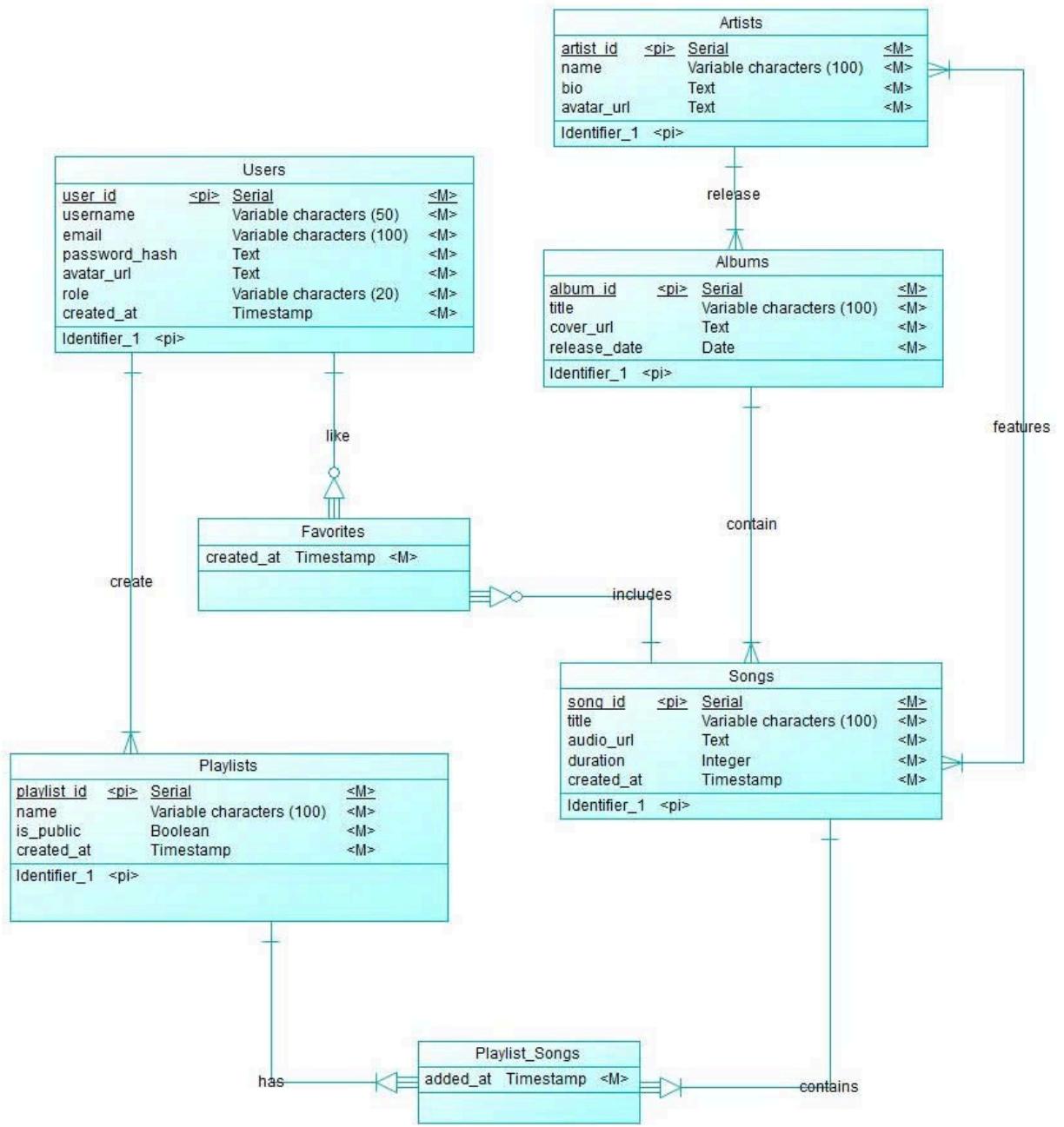
- Upload audio/music files with metadata (song title, artist, album, etc.).
- Support for common formats such as MP3.

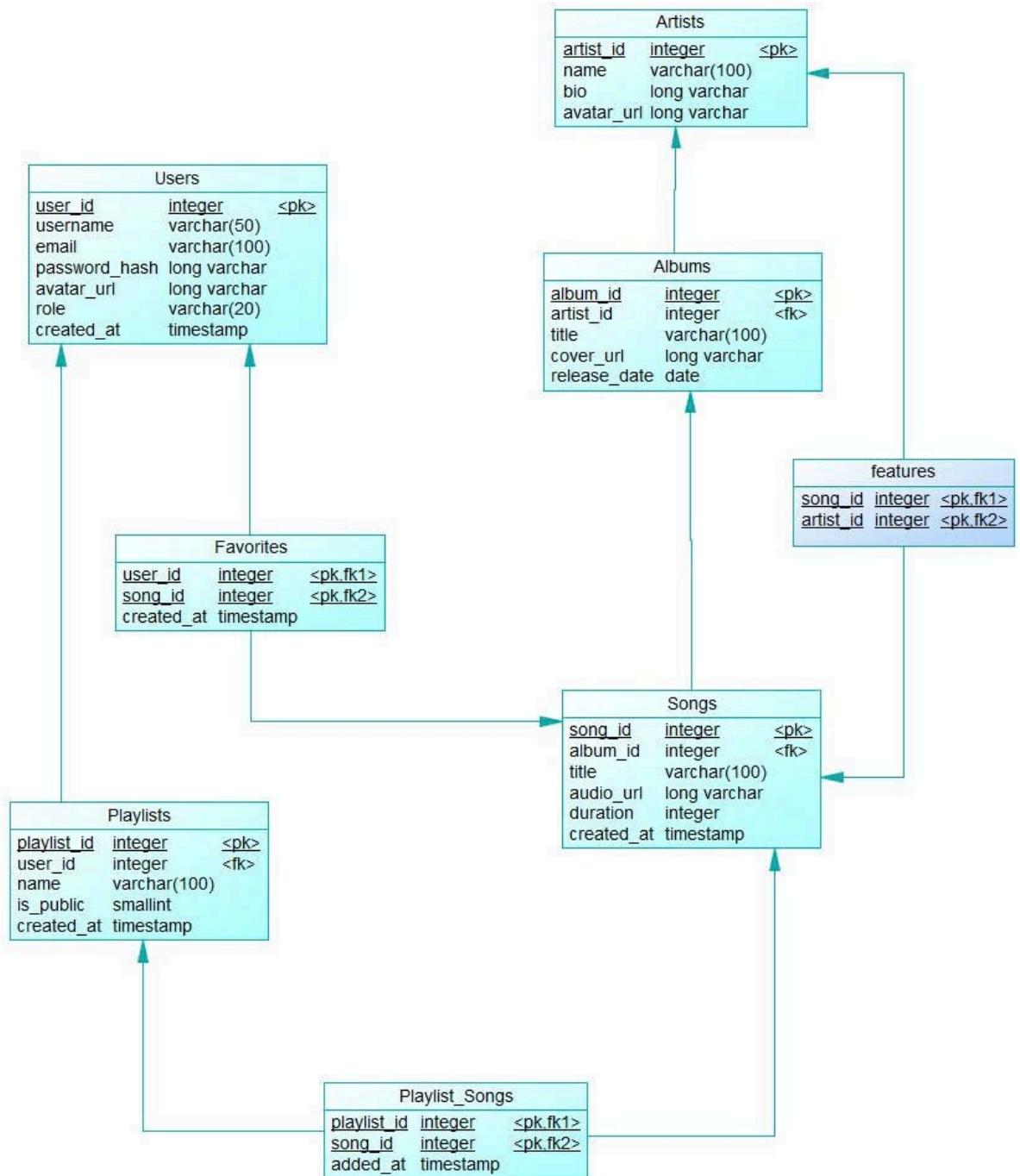
User Roles & Authentication

- Secure user registration and login with JWT authentication.
- Role-based access control: User, Admin.
Each user has a profile and can interact with content.

Admin Dashboard

- Manage users, songs, albums, and artists.
 - View system-wide statistics such as total counts of songs, users, artists, and albums.
 - Admin-only permission to delete all users.
-
- **Database schema**
 - * CDM





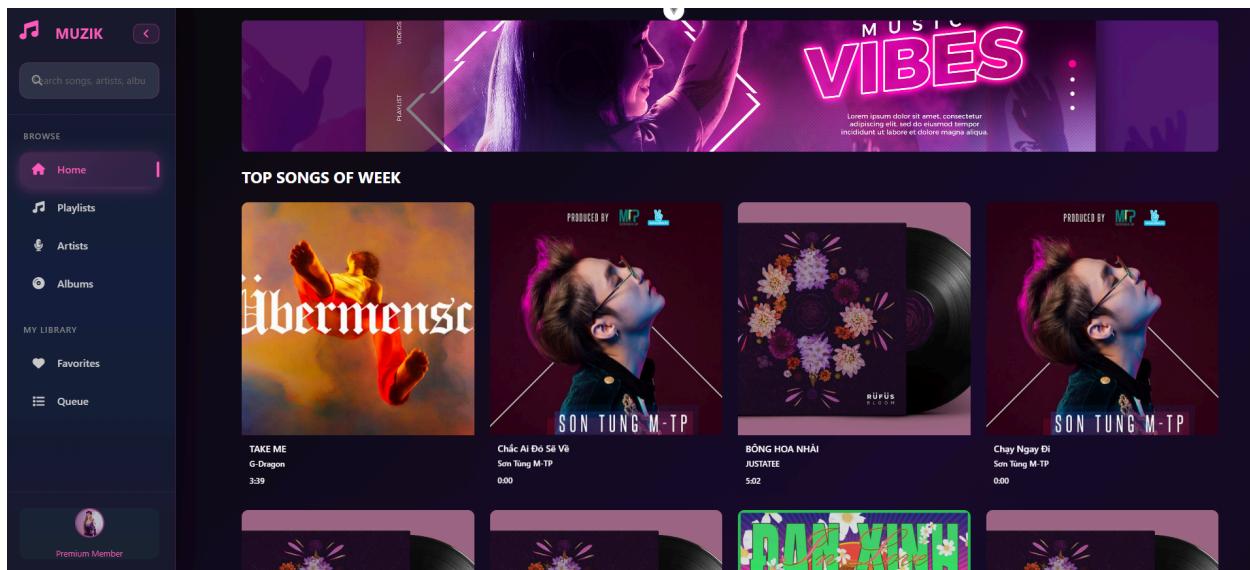
- A task assignment sheet for each member if working in groups.

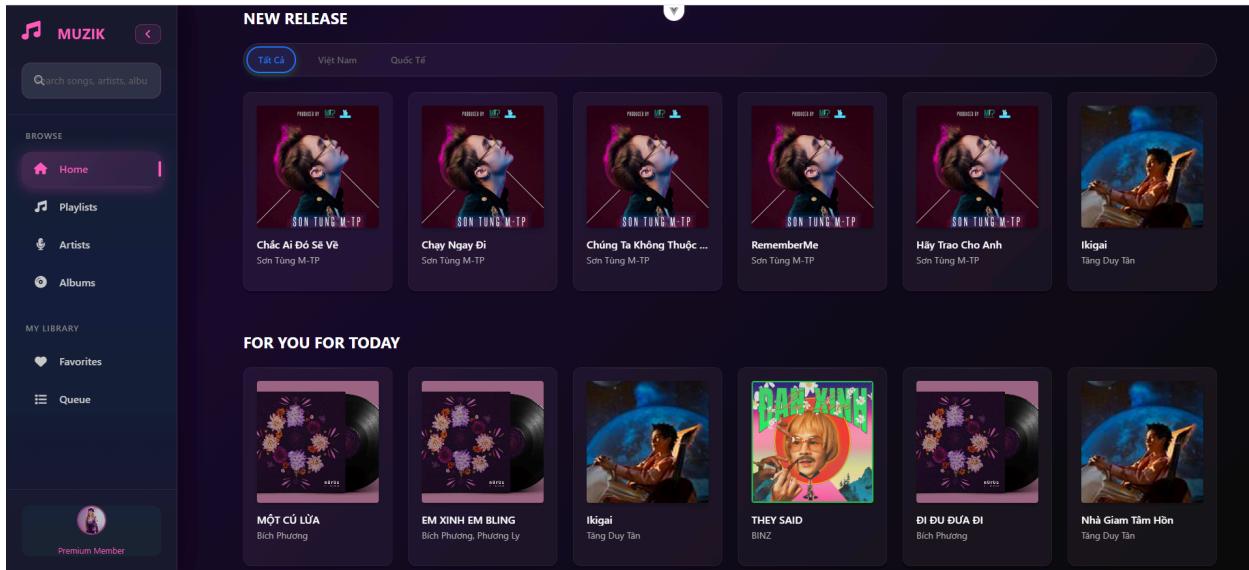
No.	Task	Member
1	Design database and data processing	Huynh Tu Phuong
2	Initialize the API Server	Tran Thi Kim Phung

3	Implement REST API for CRUD and document with OPEN API (Swagger UI)	Huynh Tu Phuong Tran Thi Kim Phung
4	Create functions to get data from server using Vue/Query; Tanstack	Huynh Tu Phuong
5	Implement client-side state for playing song and client side with route	Tran Thi Kim Phung
6	Implement UI admin	Tran Thi Kim Phung
7	Implement UI user	Huynh Tu Phuong
8	Design UI for VUE components using Bootstrap grid system	Huynh Tu Phuong Tran Thi Kim Phung
9	Build and deploy SPA & API using Caddy	Huynh Tu Phuong

II. Details of implemented features

Main HOME PAGE - USER

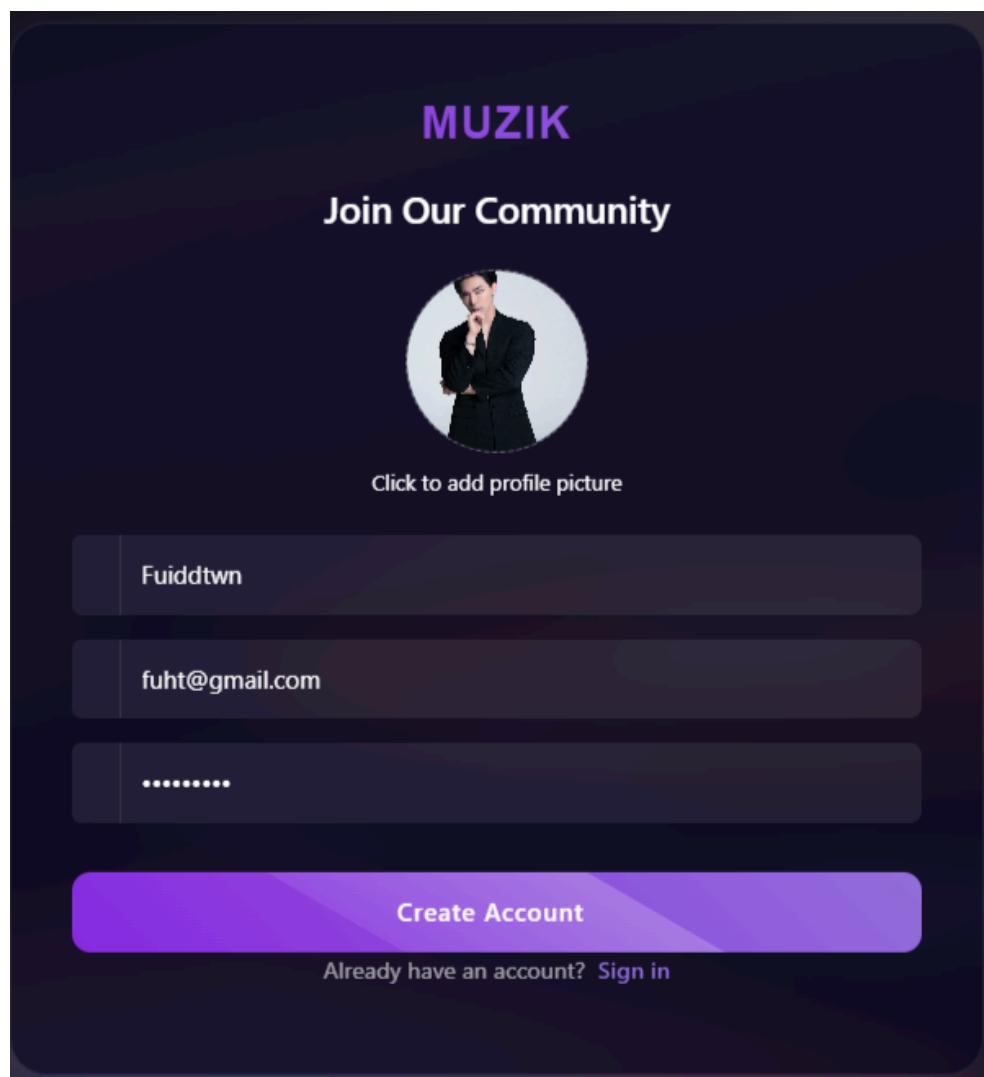
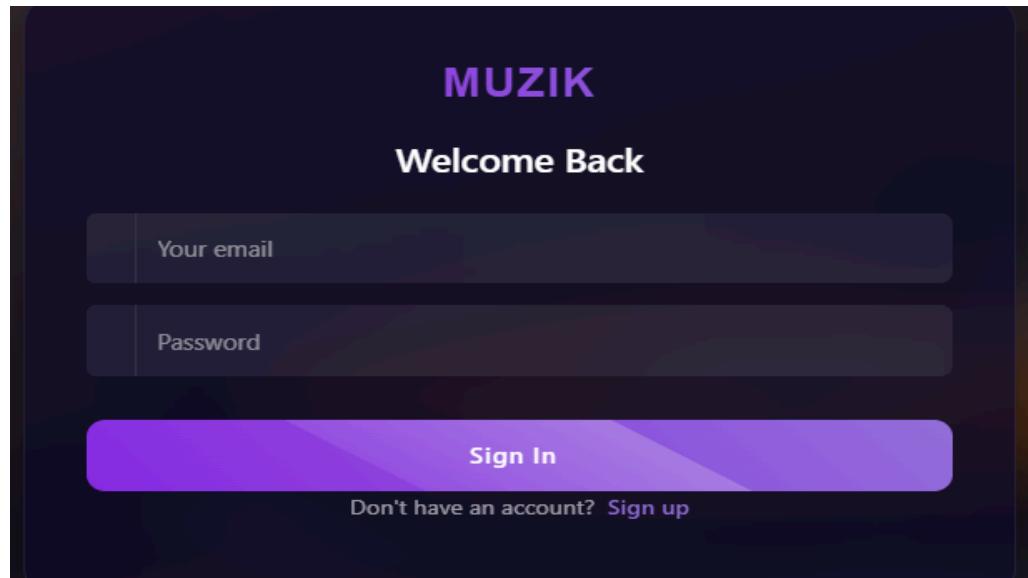




1. Feature: Log In Account

Description:

- Allows users to register a new account and log in using their credentials. Passwords are securely hashed before being stored, and a JSON Web Token is issued upon successful login to handle authentication in subsequent requests.
- This feature consists of two main components:
 - + **Register:** Allows new users to create an account by providing necessary information such as: user name, email address, password. Optionally includes additional fields like profile picture
 - + **Log In:** Lets existing users access their account by submitting: email, password



Library - use:

- **Bootstrap, Font Awesome icon**

Server - side APIs:

- **Register:**

- POST { /api/auth/register }: This endpoint is used to register a new user.
- Request Body: multipart/form-data
Fields:
 - ❖ username (string, required): User's display name (min 3 characters).
 - ❖ email (string, required): A valid email address.
 - ❖ password (string, required): Password (min 6 characters).
 - ❖ avatar_url (file, optional): Profile image.
- Response: application/json

- **Log in**

- POST { /api/auth/login }: This endpoint authenticates a user by their email and password
- Request Body: multipart/form-data
 - ❖ email (string, required): User's email address.
 - ❖ password (string, required): User's password.
- Response: application/json

Table: Users

Data Handling:

- Write: Register stores new user data with hashed password.
- Read: Login verifies email and compares hashed password.

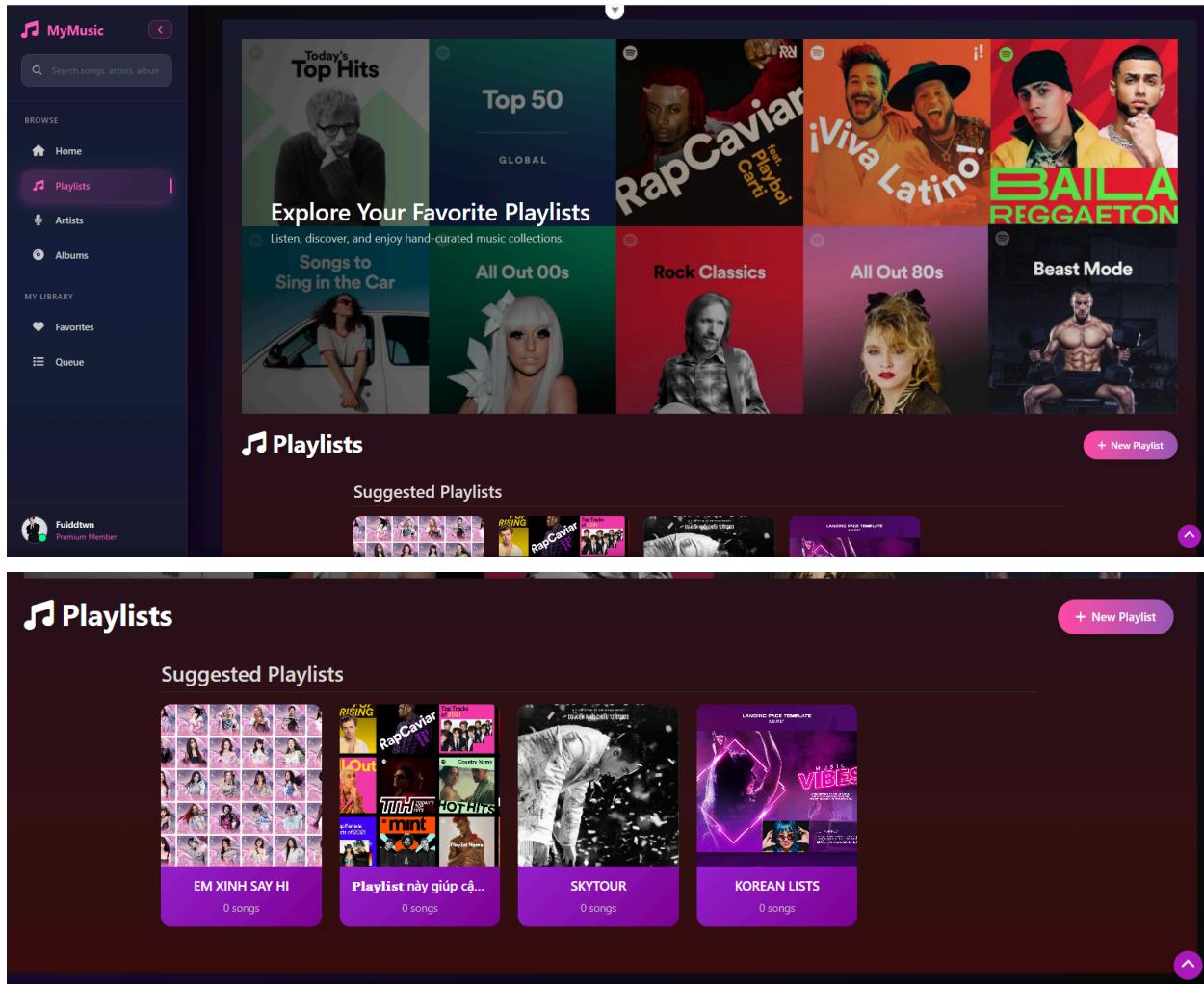
Client-side States:

isLoggedIn, email, password, avatarFile, username, avatarPreview, toast

2. Playlists

- Feature: View all playlists

Description: Retrieve a list of all playlists available in the system. This endpoint returns an array of playlists, each including basic information such as playlist name, cover image, creator, and number of songs. This endpoint is typically used by users to browse all available playlists.



Server - side APIs:

- **GET /api/playlists:** Retrieve a list of all playlists in the system.
Request Body: None
Response: application/json

Library - use:

- Bootstrap, Font Awesome icon

Table: *Playlists, Songs, Artists*

- **Read data:** Retrieve data from the system to get a list of all playlists in the database, including: Basic playlist information (ID, name, cover image). List of songs contained in each playlist. Artist information for each song.
- **Typical Client-side states:** loading, error:, showCreateModal, defaultPlaylistImage:, playlists:, topPlaylists: [], userPlaylists: [], systemPlaylists
- **Feature: View a playlist detail**
Description: Get detailed information about a specific playlist by ID.

#	TITLE	ARTIST	
1	AAA	Bích Phương	3:56
2	EM CHỈ LÀ	Bích Phương	5:14
▶	EM XINH EM BLING	Bích Phương	3:22
4	ĐỒNG DAO XINH GÁI	Orange	3:41
5	THE REAL AURA	Orange	5:00

Server - side APIs:

- **GET/api/playlists/:id:** Get details of a specific playlist
Request Body: playlist_id: integer
Response: application/json

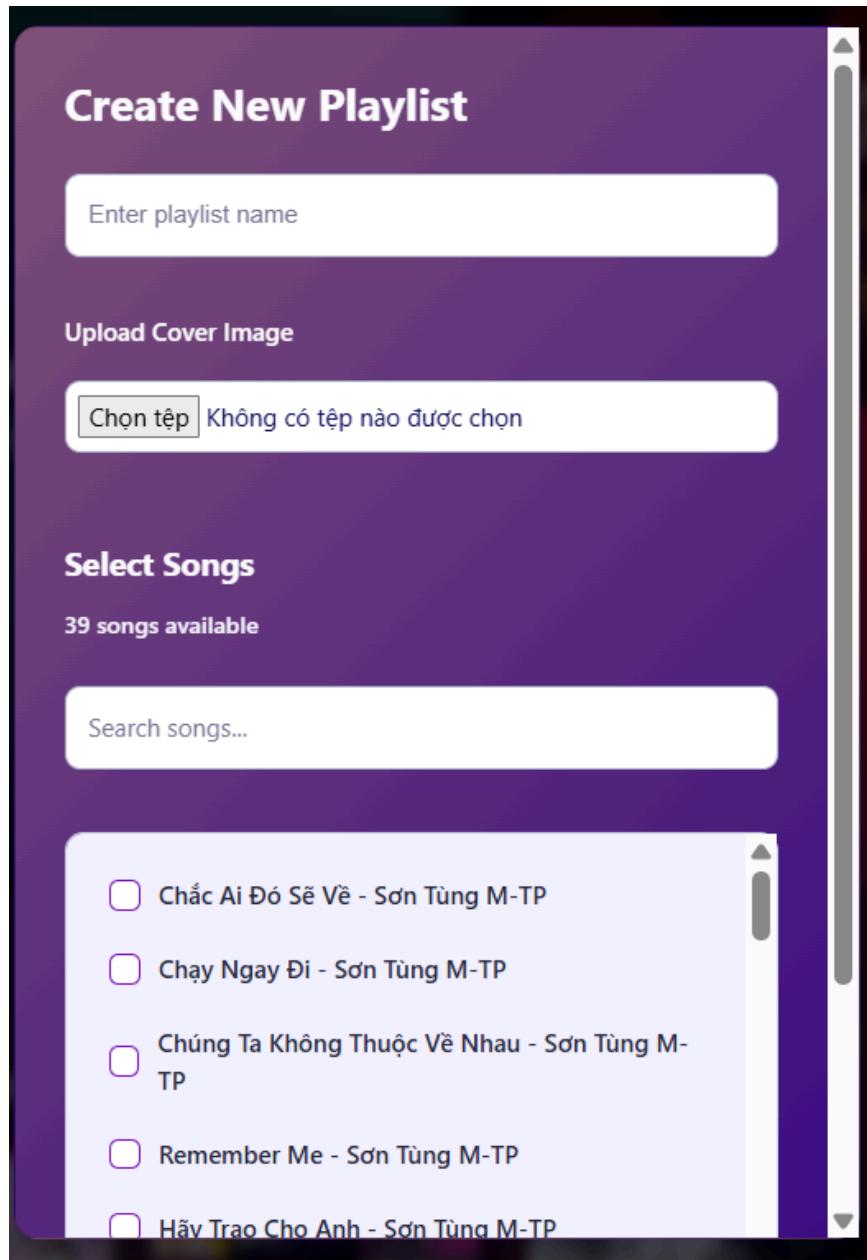
Tables: Playlists, PlaylistSongs, Songs, Artists

Read data: Retrieve data from the system to get a playlist in the database, including: Basic playlist information (ID, name, cover image). List of songs contained in each playlist. Artist information for each song.

Typical Client-side states: playlistId, playlist[], isLoading, error, player, route, router

- **Feature: Create a playlist**

Description: Create a new playlist in the system. This endpoint allows a user to submit a playlist name, optional cover image, and a list of songs (by title or ID). Upon successful creation, the new playlist is saved and associated with the authenticated user. This endpoint is typically used when users want to organize their favorite songs into custom playlists.



Library - use:

- Bootstrap, Font Awesome icon

Server - side APIs:

- **POST /api/playlists:** Create a new playlist with a list of songs.

Request Body: multipart/form-data

- ❖ name (string, required): Playlist name.
- ❖ song_ids (array<integer>, required): List of song IDs to include in the playlist.
- ❖ cover (string(binary), optional): Playlist cover image file.

Response body: application/json. Information of added playlist

Table: Playlists, Songs, Artists, Playlistsongs

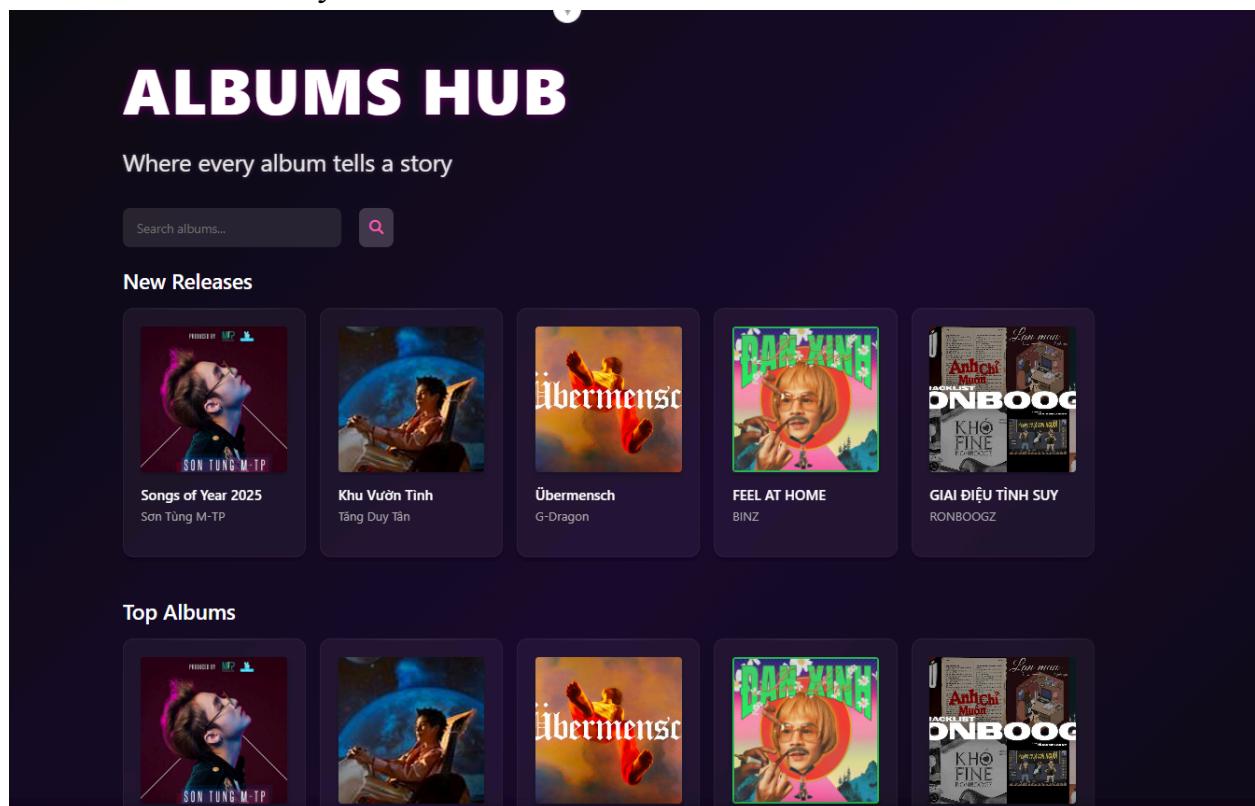
- **Read data:** Retrieve data from the system to get a list of all playlists in the database, including: Basic **playlist** information (ID, name, cover image). List of **songs** contained in each playlist. **Artist** information for each song.
- **Write data:** Insert new data into the system by creating playlists and associating them with songs and artists. Creating a new **playlist** with a name, optional cover image, and the ID(s) or title(s) of songs. Inserting records into the playlists table (for the playlist info). Inserting related entries into the **playlistsongs** junction table to link songs to the playlist.

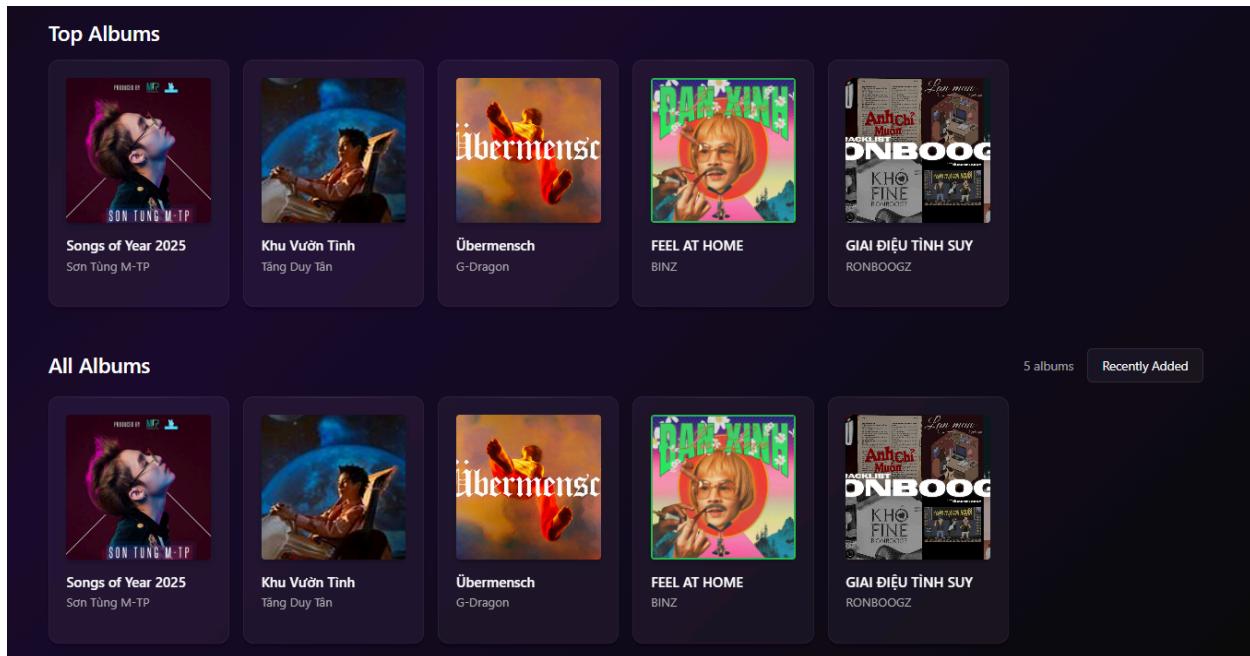
Typical Client-side states: playlistName, songSearch, selectedSongs, file, mutationError, isLoading, fetchError, songs, filteredSongs, isCreating

3. Albums

- Feature: View all albums

Description: Start playback of all songs within a specific album. This endpoint retrieves the full list of tracks associated with the given album ID, ordered by track number or release sequence, and initiates playback starting from the first track. It is typically used when a user selects an album to listen to in its entirety.





Server - side APIs:

- **GET /api/albums:** Retrieve a list of all playlists in the system.
 - **Parameter:** none
 - **Response:** application/json

Library - use:

- **Bootstrap, Font Awesome icon**

Table: Playlists, Songs, Artists

- **Read data:** Retrieve data from the system to get a list of all playlists in the database, including: Basic playlist information (ID, name, cover image). List of songs contained in each playlist. Artist information for each song.

Client-side states: currentSong, playlist, currentIndex, isPlaying, volume, currentTime, duration, isLoading, error, favorites, togglePlay.

- Feature: View songs with Album Details

Description: This feature allows the client to retrieve and display detailed information of a specific album, including album metadata (title, cover image, release date, artist) and a list of all associated songs. The songs are ordered by their track number or release sequence. This is typically used when a user opens an album page to view its content before starting playback.

Tracks

#	TITLE	ARTIST	DURATION
1	CRAYON	G-Dragon	3:16
2	TAKE ME	G-Dragon	3:39
3	DRAMA	G-Dragon	3:54
4	POWER	G-Dragon	2:23
5	TOO BAD	G-Dragon	2:33

Tracks

#	TITLE	ARTIST	DURATION
1	CRAYON	G-Dragon	3:16
2	TAKE ME	G-Dragon	3:39
3	DRAMA	G-Dragon	3:54
4	POWER	G-Dragon	2:23

TAKE ME
G-Dragon

0:08 3:39

Server-side APIs:

- **GET /api/albums/{id}**: Retrieve details of a specific album by album ID.
 - **Parameter:** id:integer
 - **Response:** application/json

Tables: Albums, Songs, Artists

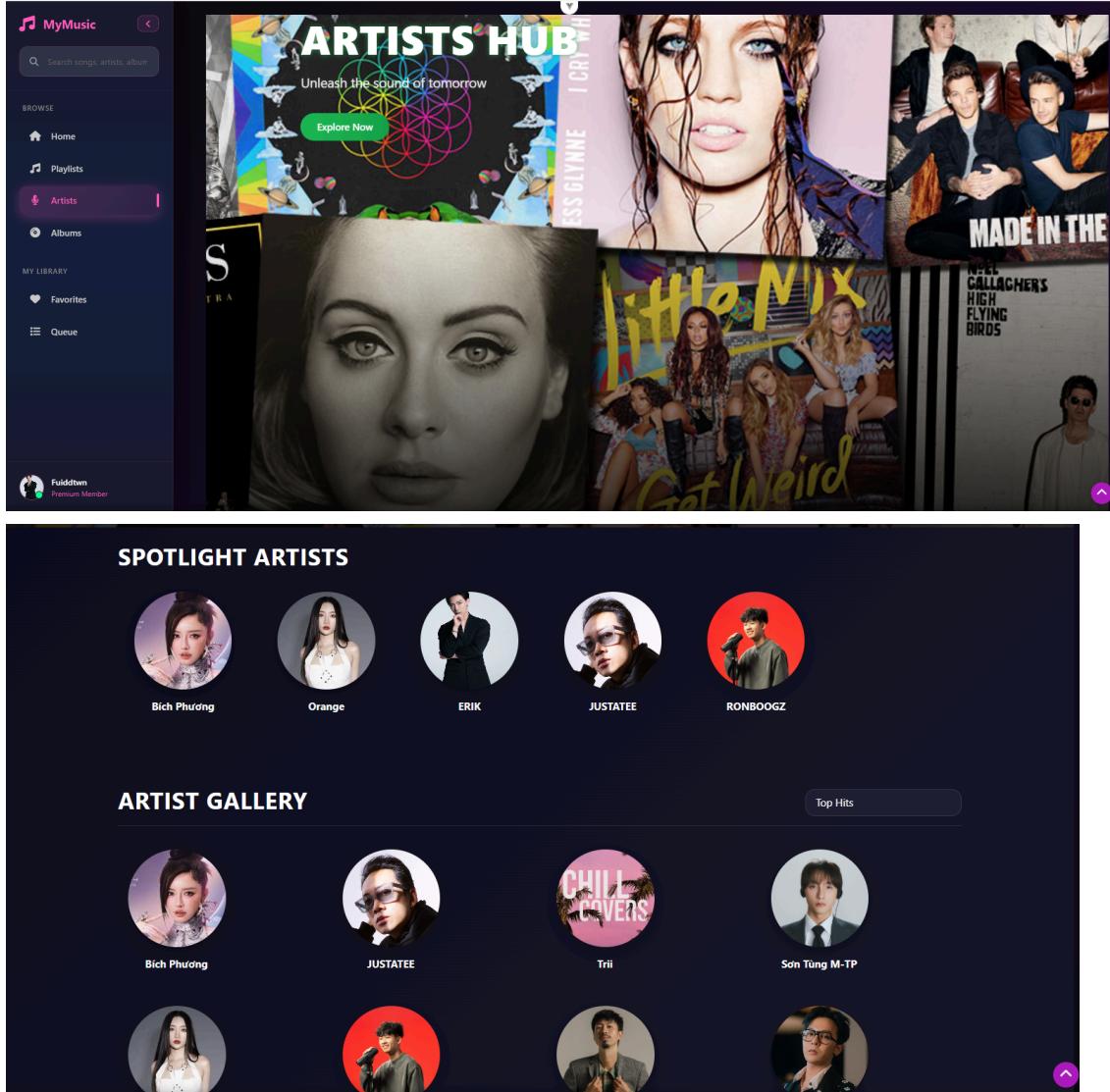
- + **Read data: The system retrieves data from the following sources:**
 - Album information: ID, title, cover image, release date.
 - Artist information: Linked via artist_id in the Albums table.
 - Songs list: All songs where album_id = {id}, sorted by track_number or release date.

Client-side states: route, audioRef, currentSong, album, isLoading, error, albumSongs, isSongsLoading, songsError, player, route

4. ARTISTS

- Feature: View all artists

Description: Allows users to view a list of all artists in the system. Each artist is displayed with their avatar and artist name in a grid format (grid gallery). This feature is commonly used in the Artist Gallery section of the user interface to explore art and access each artist's detail page.



Library - use:

- Bootstrap, Font Awesome icon

Server-side API

- GET /api/artists
 - Parameters: None
 - Response: Content-Type: application/json

Tables: Artists

Read data: The system retrieves data from the following sources:

- Artist information: Name, avatar image

Client-side states: router, sortBy, artists, isLoading, error, sortedArtists

- Feature: View artist detail

Description:

This feature enables users to view detailed information about a specific artist. Upon clicking on an artist from the gallery, the user is navigated to the artist's detail page. This page includes comprehensive data such as artist bio, avatar, list of released songs, and associated albums.

The screenshot shows a dark-themed artist detail page. At the top left is a circular profile picture of Bích Phương. To the right of the photo is her name, "Bích Phương", followed by a green circular icon with a white dot. Below her name is a brief bio: "Bích Phương là nữ ca sĩ người Việt Nam, nổi tiếng với các bản ballad nhẹ nhàng và những ca khúc pop bắt tai. Cô nổi lên từ Vietnam Idol và sau đó thành công với nhiều hit như 'Bùa Yêu', 'Đi Đu Đưa Đì'." Below the bio is a section titled "Popular Tracks" which lists five tracks by Bích Phương with their titles and durations: AAA (3:56), ĐI ĐU ĐƯA ĐÌ (3:40), NẮNG CHÉN TIỀU SẦU (2:50), CẨM KÝ THỊ HỌA (5:05), and MỘT CÚ LỪA (3:28). The background of the page has a gradient from dark purple to red.

The screenshot shows a music application interface. On the left is a sidebar with navigation options: Playlists, Artists (which is selected and highlighted in blue), and Albums. Below that is a "MY LIBRARY" section with "Favorites" and "Queue". The main content area is titled "All Songs" and displays a list of tracks by Bích Phương. The tracks listed are: AAA, ĐI ĐU ĐƯA ĐÌ, NẮNG CHÉN TIỀU SẦU, CẨM KÝ THỊ HỌA, MỘT CÚ LỪA, IKIGAI, EM CHỈ LÀ, and EM XINH EM BLING. Each track entry includes the title, artist (Bích Phương), and duration. At the bottom of the screen, there is a playback control bar showing the currently playing track, "ĐI ĐU ĐƯA ĐÌ" by Bích Phương, with a progress bar at 0:07 and a total duration of 3:40. There are also standard media controls like play/pause, previous/next, and volume.

Library - use:

- Bootstrap, Font Awesome icon

Server-side API

- GET /api/artists/{id}
 - **Parameters:** id (path parameter): The unique identifier of the artist
 - **Response:Content-Type:** application/json

Tables :Artists, Songs, Albums, Songartists

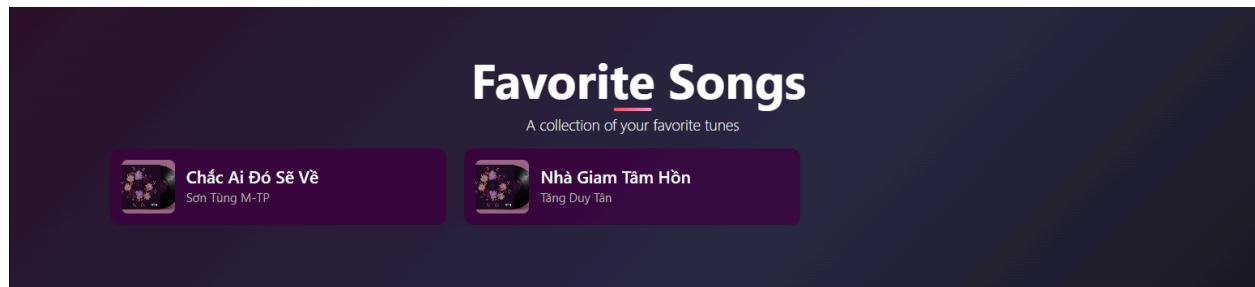
- Read data:
 - Artist info: name, bio, avatar image
 - Songs: all songs where artist_id = {id}
 - Albums: all albums where artist_id = {id}

Client-state:route, currentSong, audioRef, artistId, artist, artistSongs, artistAlbums, isLoading, error, popularSongs, allSongs, player.

5. Favorites

- Feature: View Favorite Songs by User

Description: Allows users to view a personalized list of their favorite songs. This feature is commonly accessed from the "Favorites" section of the application, where users can play, manage, or remove songs they've previously marked as favorite. Each song displays its title, artist, album cover, and duration. This feature enhances user engagement by enabling easy access to frequently listened tracks.



Library - use:

- Bootstrap, Font Awesome icon

Server-side API

- GET /api/users/:user_id/favorites
 - Retrieve all favorite songs for a specific user by user_id.

- **Parameters:** user_id: integer (from route param)
Response: Content-Type: application/json. Returns a list of favorite songs, including song details, artist, and album info.

Tables: Users, Songs, Favorites, Artists, Songs

Read Data:

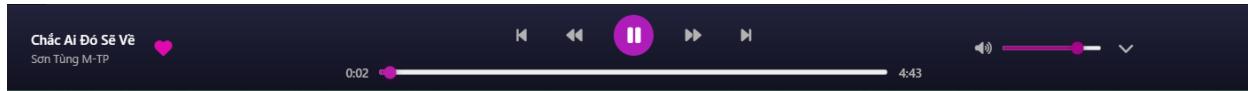
The system performs joins across tables to retrieve:

- User's list of favorite song IDs from Favorites
- Song information: title, duration, audio_url, cover_image
- Artist details: name, avatar_url

Typical Client-side States: userId, favoriteSongs, isLoading, error, currentSong, audioRef,.isPlaying, playlist, player, route, isFavorited:

- Feature: Add Favorite Song

Description: Allows authenticated users to mark a specific song as their favorite. When a user taps the "favorite" icon on a playing song.



Server-side API

- **POST** /api/users/:user_id/favorites: Add a specific song to the authenticated user's favorites.

Path Param: user_id: integer – ID of the authenticated user (from route)

Request Body: song_id :integer

Response: Content-Type: application/json

Table: Users, Favorites ,Songs

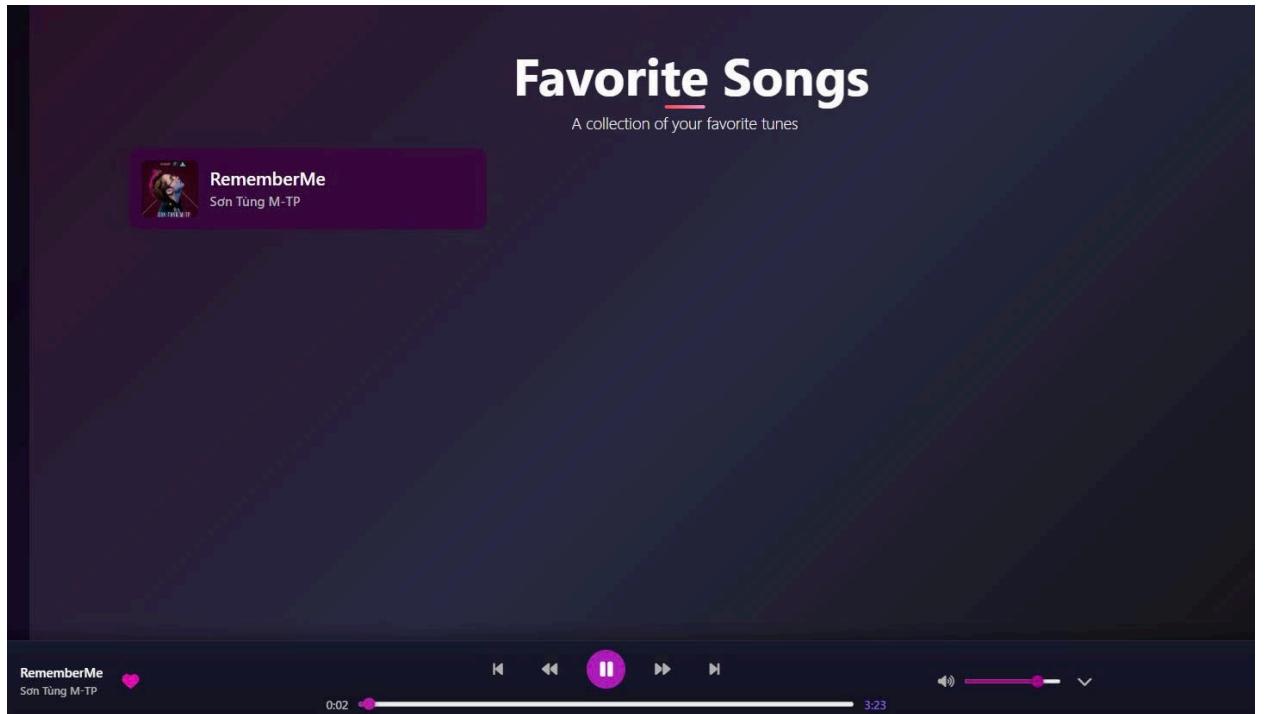
● Write Data

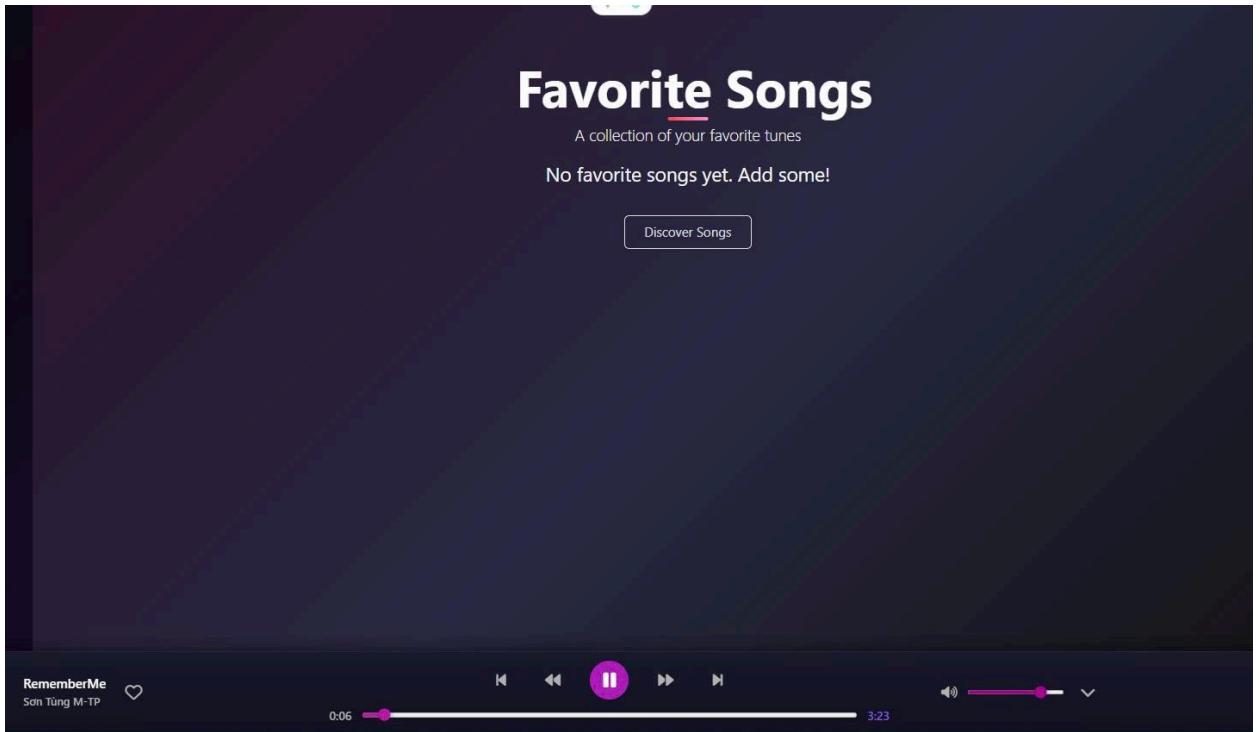
- Receive user_id from route and song_id from request body. Validate input and check if the song already exists in the user's favorites.
- If not exists, insert a new row into Favorites.
- Return confirmation message and inserted data.

Client-side States : userId, isFavorite, handleToggleFavorite(song), auth, player, favoriteSongs, error, loading

- **Feature: Remove Favorite Song**

Description: Allows authenticated users to remove a previously marked favorite song. When a user taps the "favorite" icon again on a song that is already favorited, the system removes the corresponding record from the Favorites table





Server-side API

- **DELETE /api/users/:user_id/favorites:** Remove a specific song from the user's favorite list.
Path Param: user_id: integer – ID of the authenticated user (from route)
Request Body: song_id :integer
Response: Content-Type: application/json

Table: Users, Favorites, Songs

Write Data:

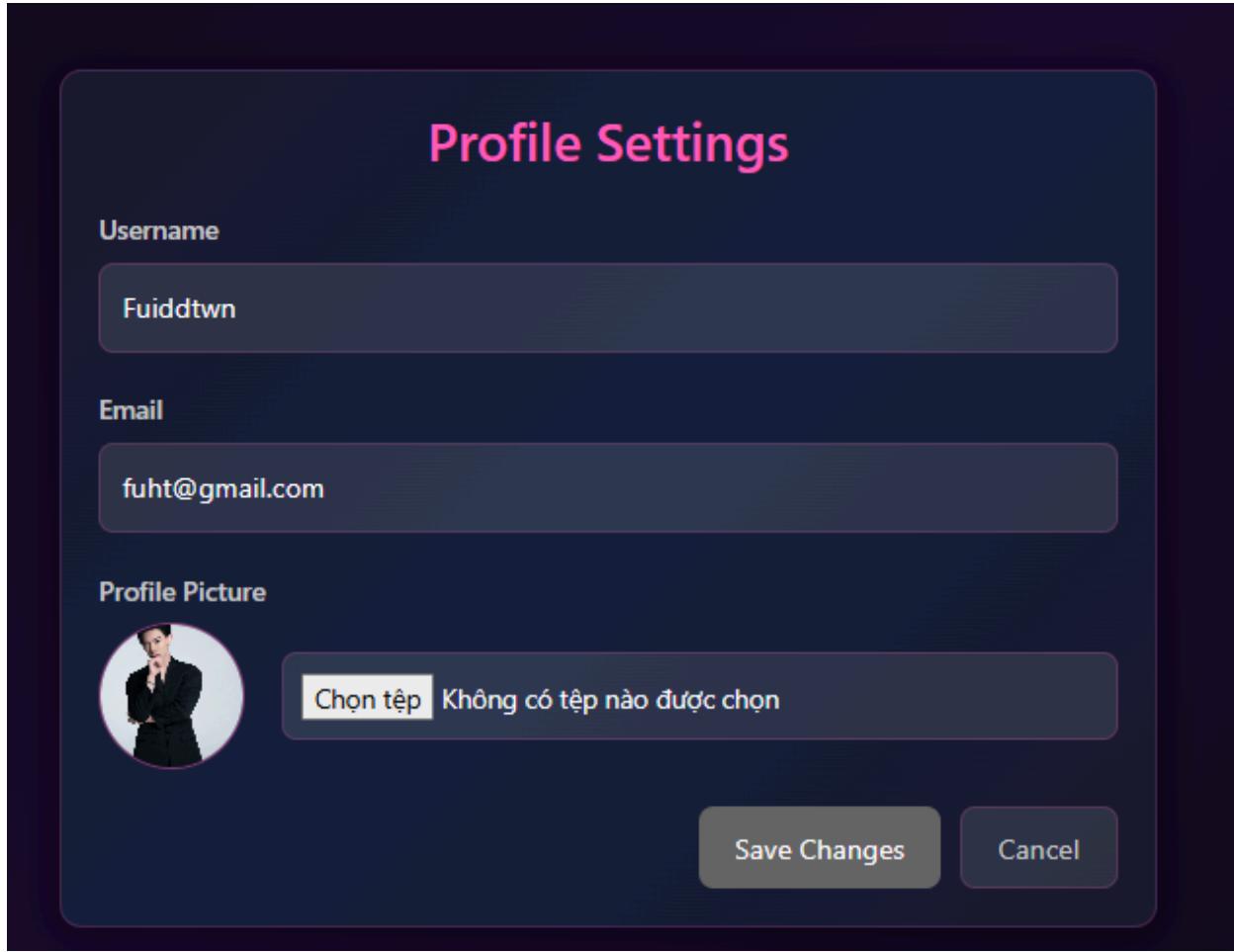
Receive user_id from route and song_id from request body.
Validate input and check if the song exists in the user's favorites.
If exists, remove the corresponding entry from Favorites.
Return a success message confirming removal.

Client-side States: userId, isFavorite, handleToggleFavorite(song), auth, player, favoriteSongs, error, loading.

6. Feature: Update information

Description: Allows an authenticated user to update their personal profile information such as name, email, password, and avatar. This feature helps users

manage their account details to keep them current and personalized. Typically accessed from the "Account Settings" or "Profile" section.



Server-side API

- **PUT /api/users/:user_id:** Update user information by user ID.

Path Param: user_id: *integer* – ID of the authenticated user (from route)

Request Body: Content-Type: multipart/form-data

- + Username, Email, Avatar_url

Response: Content-Type: application/json

Tables: Users

Write Data:

1. Receive user_id from route and form data from request body.
2. Validate and sanitize input fields (name, email, password).

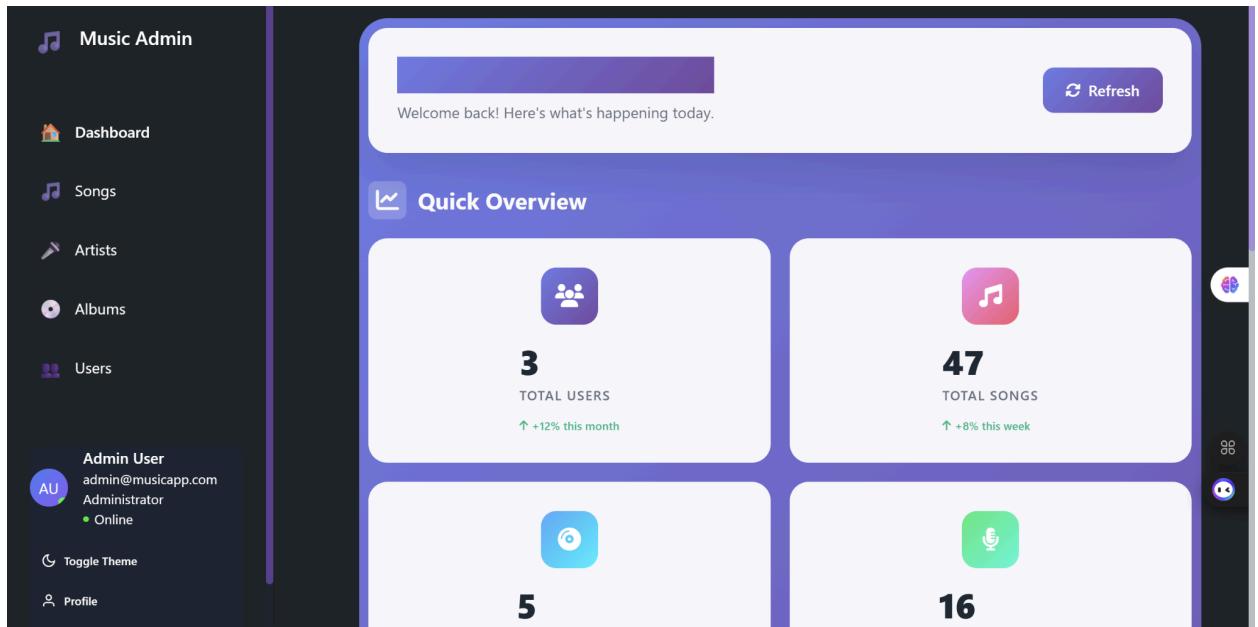
3. If avatar image is provided, upload and store it.
4. Update user's record in the Users table.
5. Return updated user info in response.

Client-side States: router, userId, username, email, avatarFile, avatarPreview, userData, isLoading, isError, isFormChanged, isUpdating, authService, handleUpdateProfile(), handleFileChange(), handleCancel()

ADMIN

Details of implemented features

1. Main HOME PAGE - ADMIN



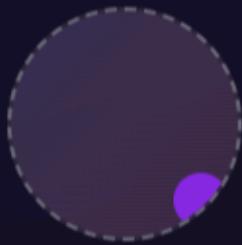
2. Feature: Log In Account

Description:

- Allows users to log in using their credentials. Passwords are securely hashed before being stored, and a JSON Web Token is issued upon successful login to handle authentication in subsequent requests.
- This feature consists of one main components:
 - + **Log In:** Lets existing users access their account by submitting email and password

MUZIK

Join Our Community



Click to add profile picture

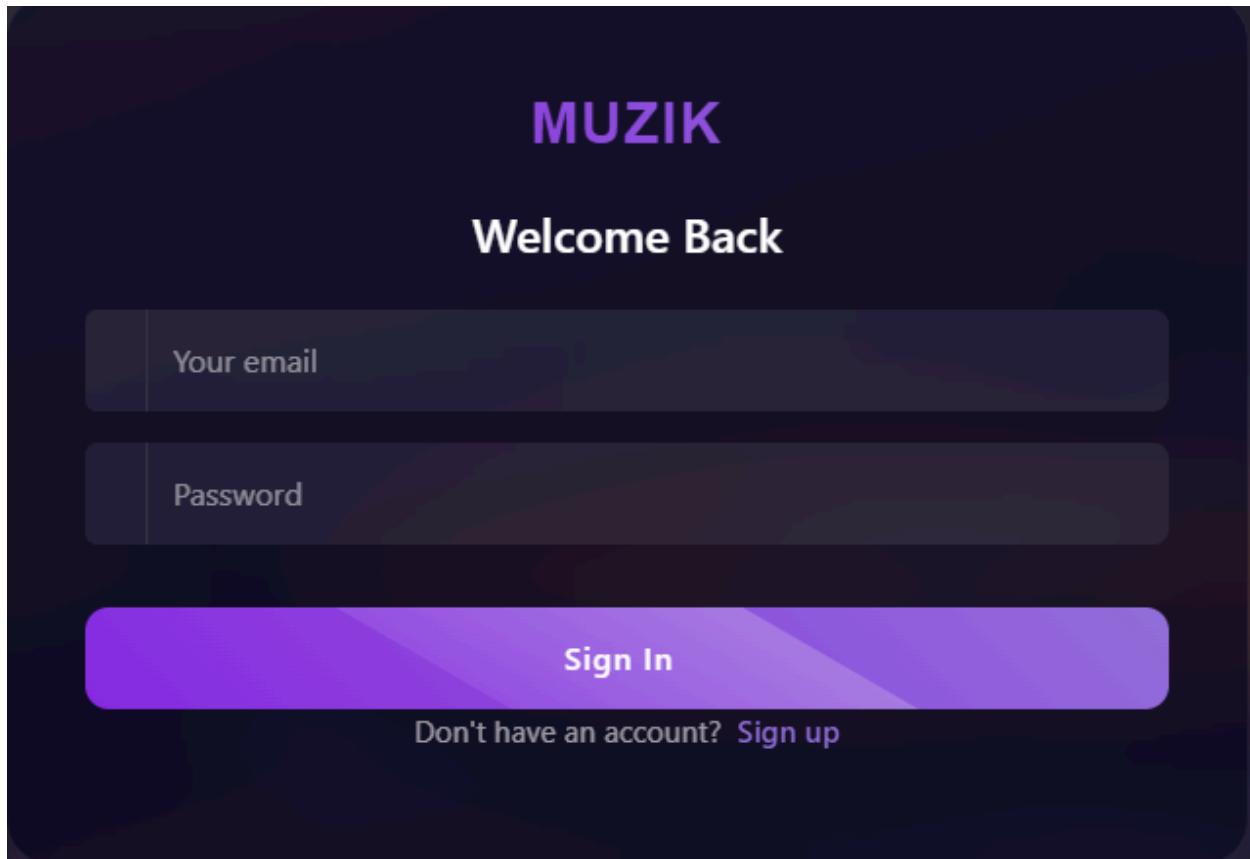
Username

Your email

Password

Create Account

Already have an account? [Sign in](#)



Library - use:

- **Bcrypt**: Hash user passwords before storing them in the database
- **Jsonwebtoken**: Generate and verify JWT access tokens after successful login
- **Express-rate-limit**: Limits the number of API requests to prevent spam and protect the system

Server - side APIs:

- Log in

- POST { /api/auth/login }: This endpoint authenticates a user by their email/username and password
- Request Body: application/json
 - ❖ email (string, required): User's email address or username
 - ❖ password (string, required): User's password.
- Response: application/json

Table: Users

Data Handling:

- Read: Login verifies email/username and compares submitted password with hashed password from database
- JWT token validation with automatic expiration checking

Client-side States : email, password, isSubmitting, loginError, adminToken, isLoggedIn, currentAdmin, authService, handleLogin(), handleInputChange(), handleLogout()

3. Songs Management

- Feature: View all songs

Description: Retrieve a list of all songs available in the system. This endpoint returns an array of songs, each including basic information such as song title, artist name, duration, audio file path, and associated metadata. This endpoint is typically used by admins to browse and manage all songs in the music library.

The screenshot shows a dark-themed application interface. On the left is a sidebar with the following items:

- Music Admin
- Dashboard
- Songs
- Artists
- Albums
- Users
- Admin User (highlighted)

 - admin@musicapp.com
 - Administrator
 - Online

- Toggle Theme
- Profile

The main content area is titled "Songs Library (47 items)". It includes a search bar and a "Add Song" button. A table lists 10 songs with columns: ID, Title, Artist, Duration, Audio File, and Actions. The table rows are as follows:

ID	Title	Artist	Duration	Audio File	Actions
34	Chắc Ai Đó Sẽ Về	Sơn Tùng M-TP	–	/uploads/audio/Chắc Ai Đó...	
35	Chạy Ngay Đi	Sơn Tùng M-TP	–	/uploads/audio/Chạy Ngay ...	
36	Chúng Ta Không Thuộc Về Nhau	Sơn Tùng M-TP	–	/uploads/audio/Chúng Ta K...	
37	RememberMe	Sơn Tùng M-TP	3:23	public/uploads/audio/Reme...	
38	Hãy Trao Cho Anh	Sơn Tùng M-TP	–	/uploads/audio/Hãy Trao C...	
39	Ikigai	Tăng Duy Tân	–	/uploads/audio/Ikigai.mp3	
40	Khu Vườn Tình	Tăng Duy Tân	–	/uploads/audio/Khu Vườn T...	
41	Nhà Giảm Tâm Hồn	Tăng Duy Tân	–	/uploads/audio/Nhà Giảm T...	

Libraries: knex (query builder), zod (validation, if used), multer (if response includes audio metadata)

Server - side APIs:

- GET /api/songs: Retrieve a list of all songs in the system.

Request Body: None

Response: application/json

Table: Songs, Artists, Albums

- **Read data:** Retrieve data from the system to get all songs with:
 - + Basic song information (ID, title, duration)
 - + Associated artist information
 - + Audio file paths and metadata

+ Creation timestamps

Client-side States : songDetails, isFetching, error, songId, fetchSongDetails(), Router

- **Feature: View a song**

Description: Get detailed information about a specific song by ID, including full metadata, artist details, and file information.

The screenshot shows a dark-themed web application interface for 'Music Admin'. On the left, there's a sidebar with navigation links: 'Music Admin' (selected), 'Dashboard', 'Songs', 'Artists', 'Albums', 'Users', 'Admin User' (with email 'admin@musicapp.com' and status 'Administrator, Online'), 'Toggle Theme', and 'Profile'. The main content area displays a table of songs with columns: ID, Title, Artist, Duration, Audio File, and Actions. A modal dialog is open over the table, centered on song ID 34. The modal header says 'development.id.vn says' and contains the following details:
ID: 34
Title: Chắc Ai Đó Sẽ Về
Artist(s): Sơn Tùng M-TP
Duration: 0:00
Audio: /uploads/audio/Chắc Ai Đó Sẽ Về.mp3
Image: /uploads/img/unnamed.jpg

Libraries: knex, zod

Server - side APIs:

- **GET /api/songs/:id:** Get details of a specific song

Request Body: song_id: integer

Response: application/json

Tables: Songs, Artists, Albums

Read data: Retrieve detailed song information including artist relationships and file metadata.

Client-side States : songDetails, isFetching, error, songId, fetchSongDetails(), router

- **Feature: Create a song**

Description: Create a new song in the system. This endpoint allows admins to upload an audio file along with song metadata (title, artist, album). The system processes the audio file, extracts metadata like duration, and stores both the file and metadata in the database.

The image shows a dark-themed user interface for a music application. On the left is a sidebar with navigation links: Music Admin, Dashboard, Songs, Artists, Albums, Users, and a profile section for 'Admin User' (admin@musicapp.com, Administrator, Online). The main content area has two parts: a modal window titled 'Add New Song' in the foreground and a table of songs in the background.

Add New Song Modal:

- Title ***: Không Cảm Xúc
- Artist ***: Nguyễn Phi Hùng
- Album**: Nỗi Buồn Bất Tân
- Audio File ***: KhongCamXuc-NguyenPhiHung-2543995.mp3 (3.49 MB)
- Duration**: 3:49

Songs Table:

ID	Title	Artist	Duration	Audio File	Actions
100	TÚ 1 SẾ THÀNH 2	KAI ĐINH	4:11	public/uploads/audio/Tu1S...	
101	HÈ	Phương Ly	4:26	public/uploads/audio/jlga...	
102	GIỮ ANH CHO NGÀY MAI	ERIK	4:50	public/uploads/audio/Giu ...	
103	SO ĐẬM	Phương Ly	3:40	public/uploads/audio/SO Đ...	
104	LUÔN YÊU ĐÓI	ĐEN VÂU	3:54	public/uploads/audio/LUON...	
105	ANH	Phương Ly	4:00	public/uploads/audio/Boom...	
106	EM CHỈ LÀ	TIỀN TIỀN	5:14	public/uploads/audio/emch...	
110	Không Cảm Xúc	Nguyễn Phi Hùng	3:49	public/uploads/audio/Khon...	

Showing 41 to 48 of 48 entries

Libraries: multer (file upload), knex, zod (input validation), ffprobe or music-metadata (audio duration)

Server - side APIs:

- **POST /api/songs/:** Create a new song

Request Body: FormData containing:

- title: string (required)
- artist: string (required)
- album: string (optional)
- audio_files: File (required)

- duration: number (auto-calculated)

Response: application/json

Tables: Songs, Artists, Albums

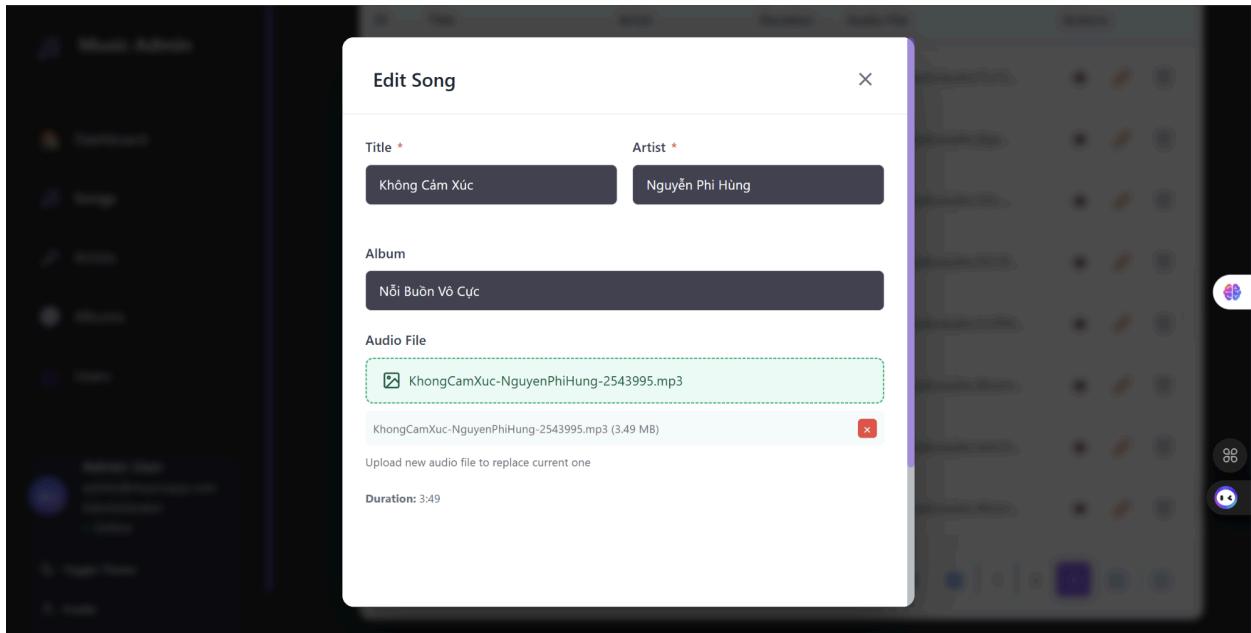
Read data: Fetch artist/album list for form dropdowns.

Write data: Insert new song record with file upload handling and metadata extraction.

Client-side States : title, artist, album, audioFile, uploadProgress, isCreating, validationErrors, successMessage, handleSubmit(), handleFileChange()

- Feature: Edit a song

Description: Update an existing song's information including metadata and optionally replace the audio file. Supports partial updates where only changed fields are updated.



Libraries: multer, knex, zod

Server - side APIs:

- **PUT /api/songs/:id:** Update an existing song

Request Body: FormData containing updated fields

Response: application/json

Read data: Fetch current song data and reference lists (artists, albums).

Write data: Update only changed fields; optionally update audio file.

Client-side States : editedSong, audioFile, isUpdating, updateError, handleUpdate(), handleInputChange(), handleCancel()

- Feature: Delete a song

Description: Remove a song from the system including its audio file and database records.

ID	Title	Artist	Duration	Audio File	Actions
100	HÈ	Phương Ly	4:26	public/uploads/audio/jlga...	
101	GIỮ ANH CHO NGÀY MAI	ERIK	4:50	public/uploads/audio/Giu ...	
102	SO ĐẬM	Phương Ly	3:40	public/uploads/audio/SO Đ...	
103	LUÔN YÊU ĐỜI	ĐEN VÂU	3:54	public/uploads/audio/LUÔN...	
104	ANH	Phương Ly	4:00	public/uploads/audio/Boom...	
105	EM CHỈ LÀ	TIỀN TIỀN	5:14	public/uploads/audio/emch...	
106	Không Cảm Xúc	Nguyễn Phi Hùng	3:49	public/uploads/audio/Khon...	

ID	Title	Artist	Duration	Audio File	Actions
100	TỬ 1 SẾ THÀNH 2	KAI ĐỊNH	4:11	public/uploads/audio/Tu1S...	
101	HÈ	Phương Ly	4:26	public/uploads/audio/jlga...	
102	GIỮ ANH CHO NGÀY MAI	ERIK	4:50	public/uploads/audio/Giu ...	
103	SO ĐẬM	Phương Ly	3:40	public/uploads/audio/SO Đ...	
104	LUÔN YÊU ĐỜI	ĐEN VÂU	3:54	public/uploads/audio/LUÔN...	
105	ANH	Phương Ly	4:00	public/uploads/audio/Boom...	
106	EM CHỈ LÀ	TIỀN TIỀN	5:14	public/uploads/audio/emch...	

Libraries: multer, knex, fs

Server - side APIs:

- **DELETE /api/songs/:id:** Delete a song

Request Body: song_id: integer

Response: application/json

Read data: Used to confirm selected song before delete.

Write data: Deletes song record and audio file from storage.

Client-side States : selectedSong, confirmDeleteModalOpen, isDeleting, deleteError, handleConfirmDelete(), handleOpenModal(), handleCloseModal()

4. Artists Management

- Feature: View all artists

Description: Retrieve a list of all artists in the system with their profile information, avatar images, and basic statistics.

The screenshot shows the 'Music Admin' application interface. On the left is a dark sidebar with the following navigation items: Dashboard, Songs, Artists (selected), Albums, Users, and Admin User (with details: admin@musicapp.com, Administrator, Online). Below these are links for Toggle Theme and Profile. The main content area is titled 'Artists Hub (16 items)' and contains a table with the following data:

Avatar	ID	Artist Name	Actions
	35	Bích Phương	
	30	Orange	
	31	ERIK	
	32	JUSTATEE	
	38	RONBOOGZ	

Libraries: multer, knex, zod

Server - side APIs:

- **GET /api/artists:** Retrieve all artists

Request Body: None

Response: application/json

Table: Artists, Users

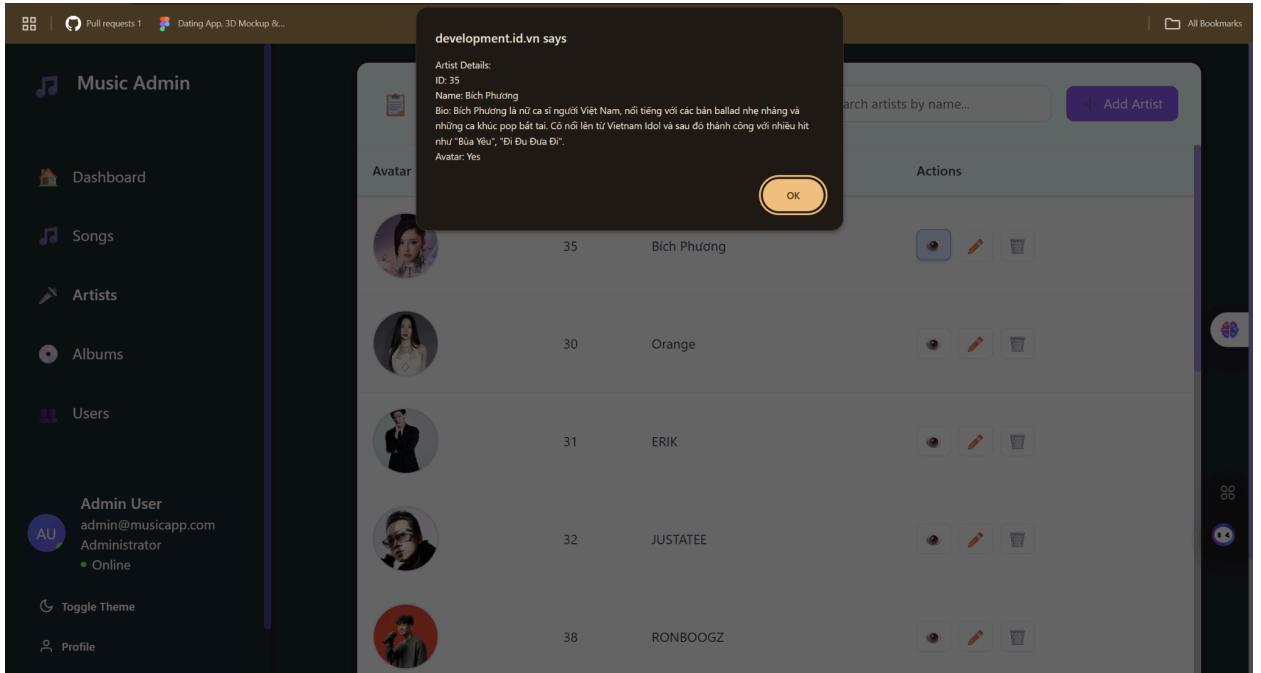
Read data: Retrieve data from the system to get all artists with:

- + Basic artist information (ID, name, biography)
- + Avatar image paths - Associated user information
- + Creation timestamps

Client-side States : artists, isLoading, error, searchText, selectedArtist, handleSelectArtist(), fetchArtists()

- Feature: View an artist

Description: Get detailed information about a specific song by ID, including full metadata, artist details, and file information.



Libraries: knex, zod

Server - side APIs:

- **GET /api/artists/:id:** Get details of a specific artist

Request Body: artist_id: integer

Response: application/json

Tables: Songs, Artists, Albums, Users

Read data: Retrieve detailed artist information including user relationships and associated content.

Client-side States : formData, avatarFile, previewAvatar, isSaving, formErrors, successMsg, handleInputChange(), handleChangeFile(), handleSaveArtist(), handleCancel()

- Feature: Create an artist

Description: Create a new artist in the system. This endpoint allows admins to add an artist profile with biographical information and avatar image upload. The system processes the avatar image and stores both the profile data and image in the database.

The screenshot displays two main views of the Music Admin application. On the left is the 'Music Admin' dashboard with a dark theme, featuring a sidebar with links for Dashboard, Songs, Artists (which is highlighted in purple), Albums, and Users. It also shows a user profile for 'User phung@email.com admin' who is online. On the right is the 'Artists Hub' view, which lists 18 artists. The table includes columns for Avatar, ID, Artist Name, and Actions. The first three entries are shown:

Avatar	ID	Artist Name	Actions
	48	TIỀN TIỀN	
	49	Troye Sivan	
	50	Nguyễn Phi Hùng	

Below the table, it says 'Showing 16 to 18 of 18 entries'. At the top of the page, there is a search bar, a 'Refresh Data' button, and a notification bell with 3 notifications. A modal window titled 'Add New Artist' is open, prompting for 'Artist Name' (Troye Sivan) and 'Biography' (A real life angel). An optional 'Description' field is present. An 'Avatar Image' section shows a file named '2018.06.10_Troye_Sivan_at_Capital_Pride_w_Sony_A7III_Washington_DC_USA_03462_(426906555).JPG' (321.24 KB). A preview of the uploaded photo is displayed.

Libraries: multer (file upload), knex, zod

Server - side APIs:

- **POST /api/artists/:** Create a new artist

Request Body: FormData containing:

- + name: string (required)
- + bio: string (optional)
- + avatar_url: File (optional)

Response: application/json

Tables: Users, Artists

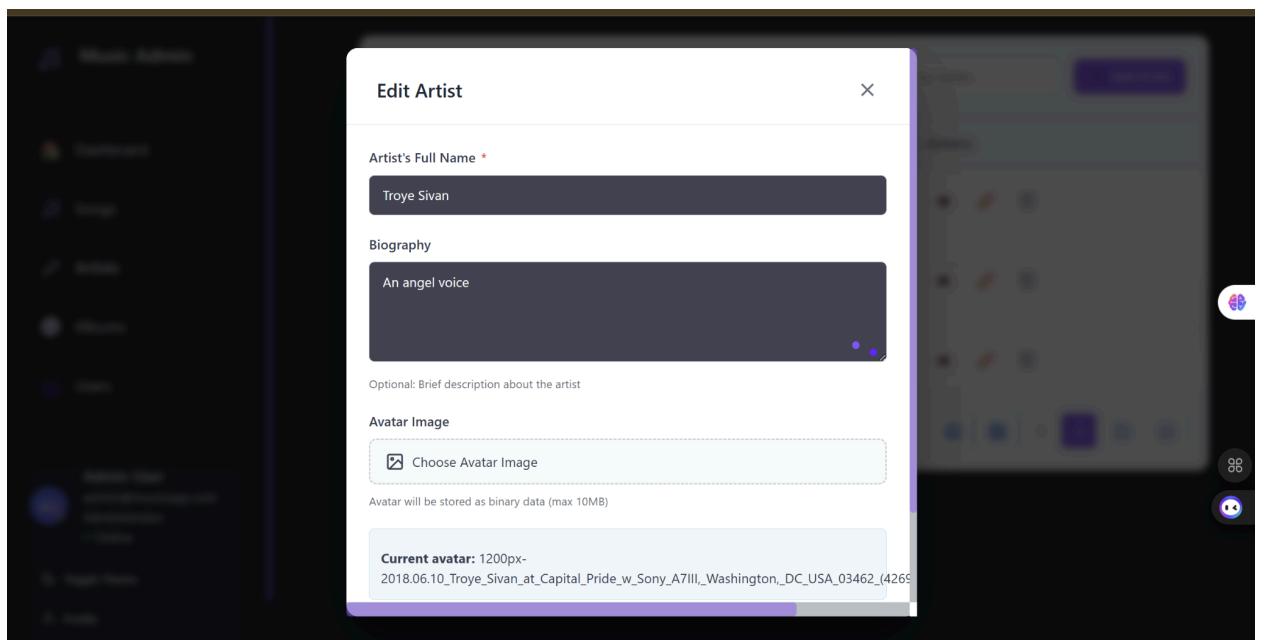
Read data: Fetch related user list.

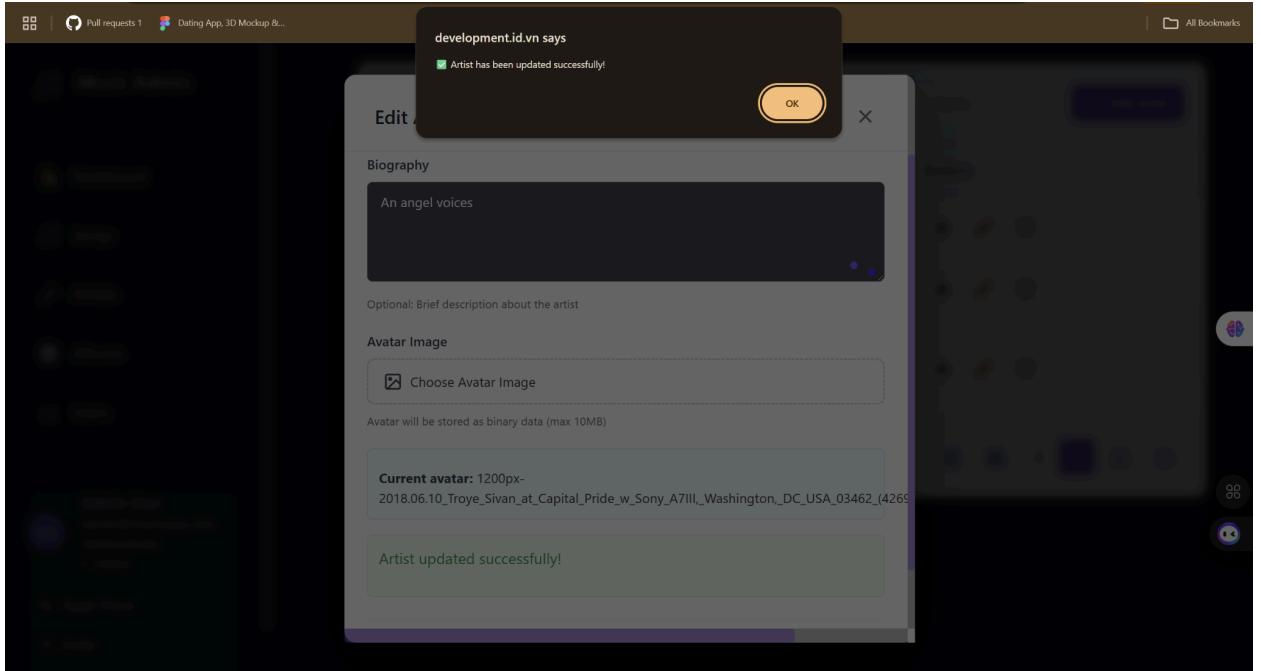
Write data: Insert new artist record with avatar image upload handling and profile data storage.

Client-side States : formData, avatarFile, previewAvatar, isSaving, formErrors, successMsg, handleInputChange(), handleFileChange(), handleSaveArtist(), handleCancel()

- **Feature: Edit an artist**

Description: Update an existing artist's information including name, biography, and optionally replace the avatar image. Supports partial updates where only changed fields are updated.





Libraries: multer, knex, zod

Server - side APIs:

- **PUT /api/artists/:id:** Update an existing artist

Request Body: FormData containing:

- + name: string (required)
- + bio: string (optional)
- + avatar_url: File (optional)

Response: application/json

Tables: Users, Artists

Read data: Fetch current artist data and user mapping.

Write data: Update artist record with optional avatar image replacement and profile data updates.

Client-side States : formData, avatarFile, previewAvatar, isSaving, formErrors, successMsg, handleInputChange(), handleFileChange(), handleSaveArtist(), handleCancel()

- Feature: Delete an artist

Description: Remove an artist from the system including their profile data, avatar image, and database records.

The screenshots show a dark-themed web application interface for managing artists. On the left is a sidebar with navigation links: Dashboard, Songs, Artists, Albums, Users, and Admin User (with details: admin@musicapp.com, Administrator, Online). Below the sidebar are 'Toggle Theme' and 'Profile' buttons.

The main area displays a table of artists with columns: Avatar, ID, Artist Name, and Actions. The table shows three entries:

Avatar	ID	Artist Name	Actions
TI	48	TIỀN TIỀN	
NP	50	Nguyễn Phi Hùng	
Troye Sivan	49	Troye Sivan	

Below the table, a message says "Showing 16 to 18 of 18 entries".

In the top screenshot, a modal dialog from "development.id.vn says" asks, "Are you sure you want to delete 'Troye Sivan'?". It has "OK" and "Cancel" buttons. In the bottom screenshot, the modal shows a green checkmark and the message "Artist 'Troye Sivan' has been deleted successfully." It also has an "OK" button.

Libraries: multer, knex, fs

Server - side APIs:

- **DELETE /api/artists/:id:** Delete an artist
Request Body: artist_id: integer
Response: application/json

Read data: Fetch current artist data and user mapping.

Write data: Remove artist entry and delete avatar if exists.

Client-side States : selectedArtist, confirmDelete, isDeleting, deleteError, successMsg, handleDeleteArtist(), handleOpenDeleteModal(), handleCancelDelete()

5. Albums

- Feature: View all albums

Description: Retrieve a list of all albums available in the system. This endpoint returns an array of albums, each including basic information such as album title, artist name, release date, cover image, and associated metadata. This endpoint is typically used by admins to browse and manage all albums in the music library.

Cover	ID	Album Title	Artist	Release Date	Actions
	8	Songs of Year 2025	Sơn Tùng M-TP	5/25/2025	
	9	Khu Vườn Tình	Tăng Duy Tân	2/22/2025	
	10	Übermensch	G-Dragon	2/21/2025	
	13	GIAI ĐỊỆU TÌNH SUY	RONBOOGZ	5/24/2025	
	11	FEEL AT HOME 2025	BINZ	7/19/2025	

Libraries: zod, knex

Server - side APIs:

- **GET /api/albums:** Retrieve a list of all albums in the system
Request Body: None
Response: application/json

Table: Artists, Albums

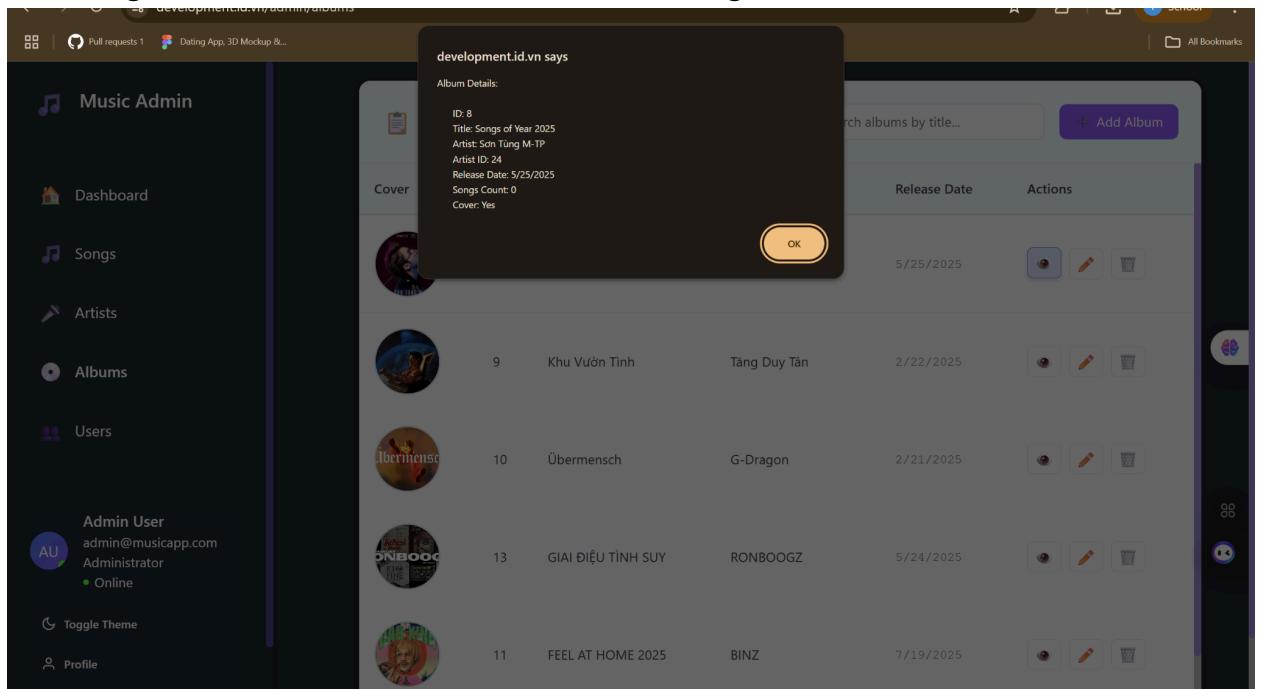
Read data: Retrieve data from the system to get all albums with:

- + Basic album information (ID, title, release date)
- + Associated artist information

Client-side States : albums, isLoading, error, selectedAlbum, searchTerm, handleSelectAlbum(), fetchAlbums()

- Feature: View an album

Description: Get detailed information about a specific album by ID, including full metadata, artist details, track listing, and cover information.



Libraries: knex, zod

Server - side APIs:

- **GET /api/albums/:id:** Get details of a specific album

Request Body: album_id: integer

Response: application/json

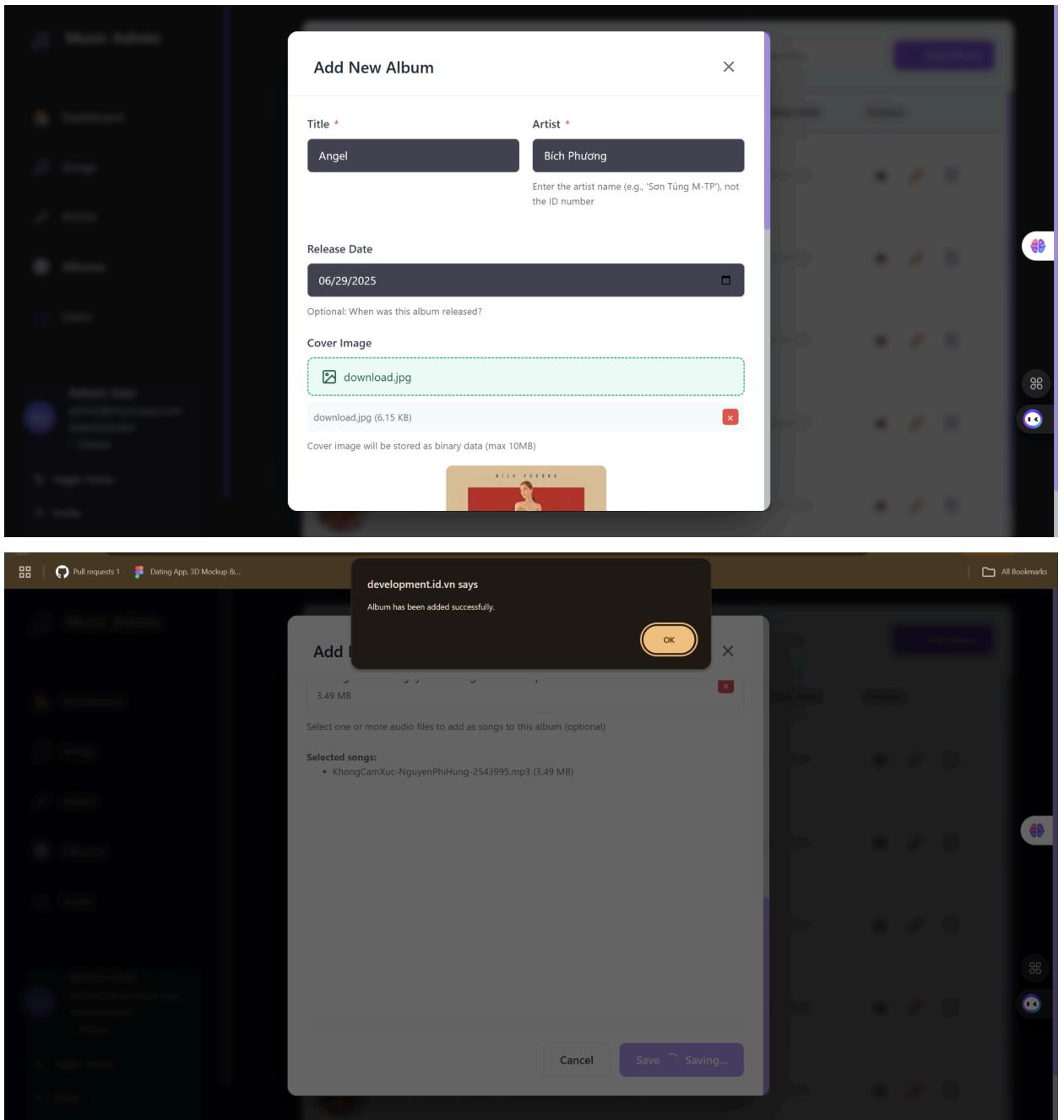
Tables: Songs, Artists, Albums

Read data: Retrieve detailed album information including artist relationships and track listings.

Client-side States : albumDetails, albumId, albumSongs, isLoading, error, artistInfo, trackList, fetchAlbumDetails(), router

- Feature: Create an album

Description: Create a new album in the system. This endpoint allows admins to add an album with cover art upload and artist association. The system processes the cover image and stores both the album data and cover art in the database.



Libraries: knex, zod, multer

Server - side APIs:

- **POST /api/albums/:** Create a new album

Request Body: FormData containing:

- + title: string (required)
- + artist: string (required)
- + release_date: date (optional)

+ cover: File (optional)

Response: application/json

Tables: Albums, Artists

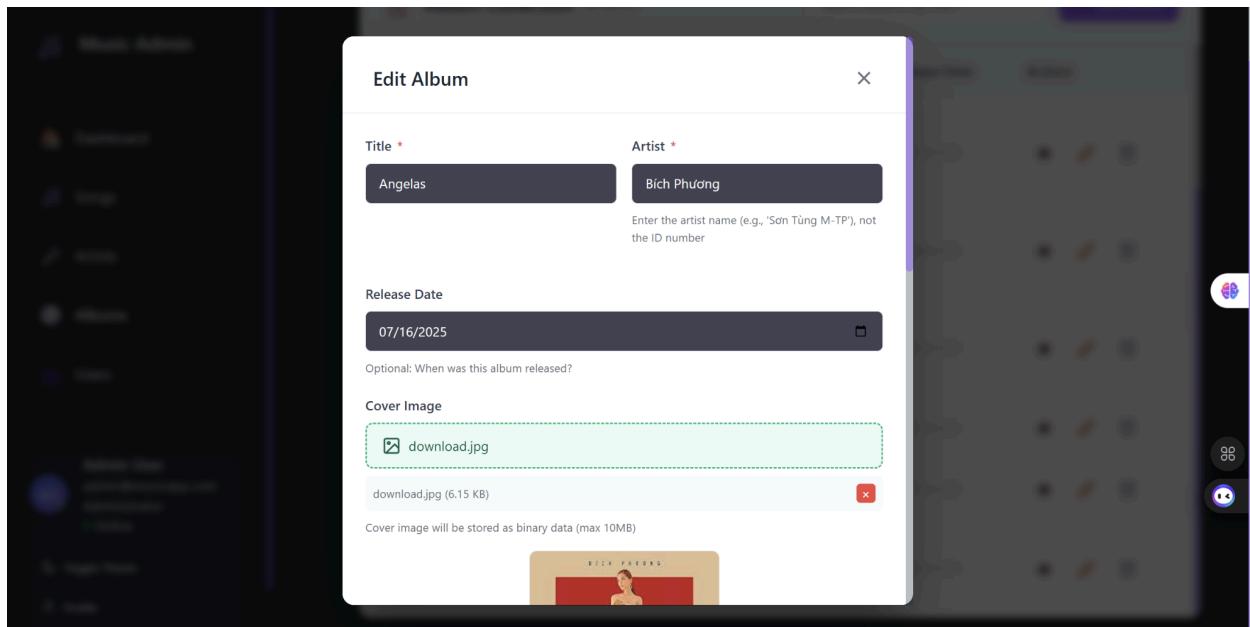
Read data: Retrieve artist list for selection before creation.

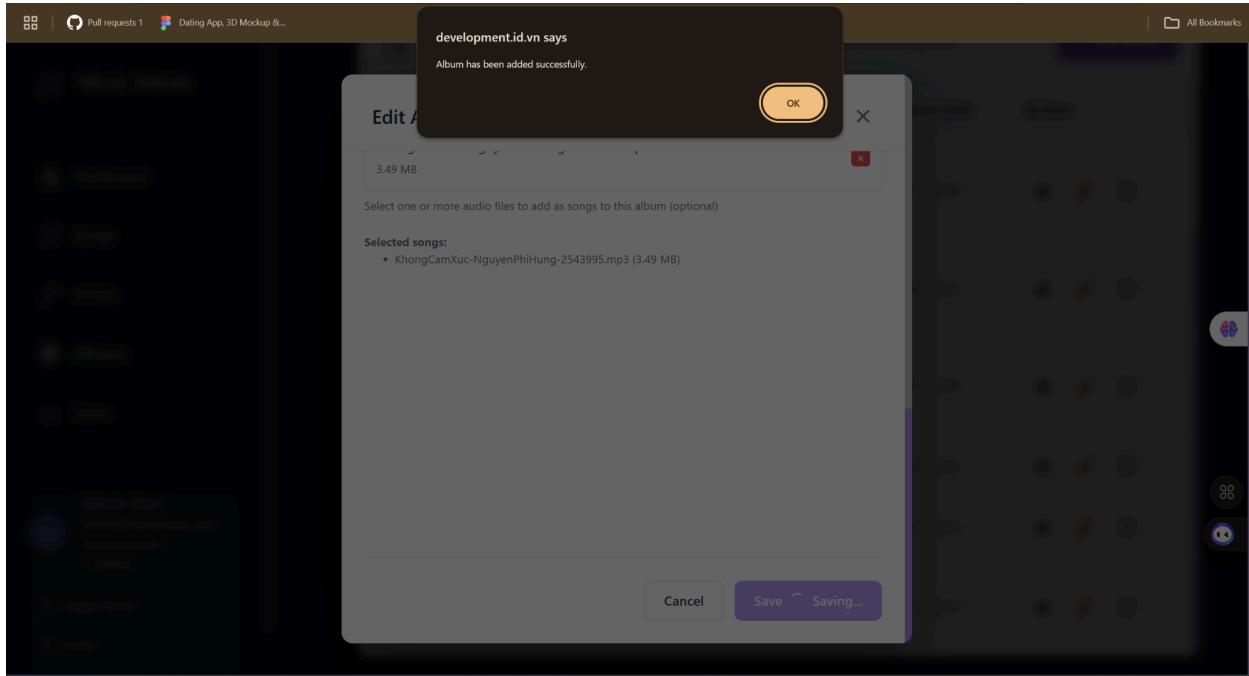
Write data: Insert new album record, upload and store cover image if provided.

Client-side States : albums, isLoading, error, selectedAlbum, searchTerm, handleSelectAlbum(), fetchAlbums()

- **Feature: Edit an album**

Description: Update an existing album's information including title, artist, release date, and optionally replace the cover image. Supports partial updates where only changed fields are updated.





Libraries: knex, zod, multer

Server - side APIs:

- **PUT /api/albums/:id:** Update an existing album

Request Body: FormData containing:

- + title: string (optional)
- + artist: string (optional)
- + release_date: date (optional)
- + cover: File (optional)

Response: application/json

Tables: Albums, Artists

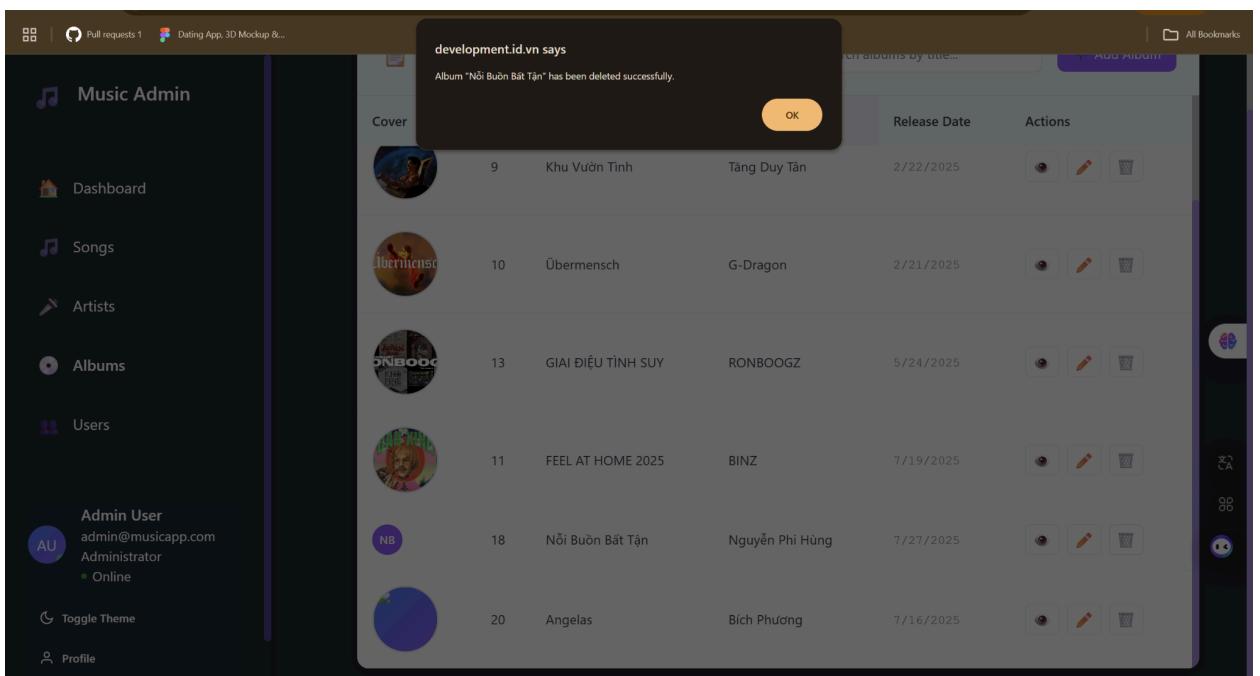
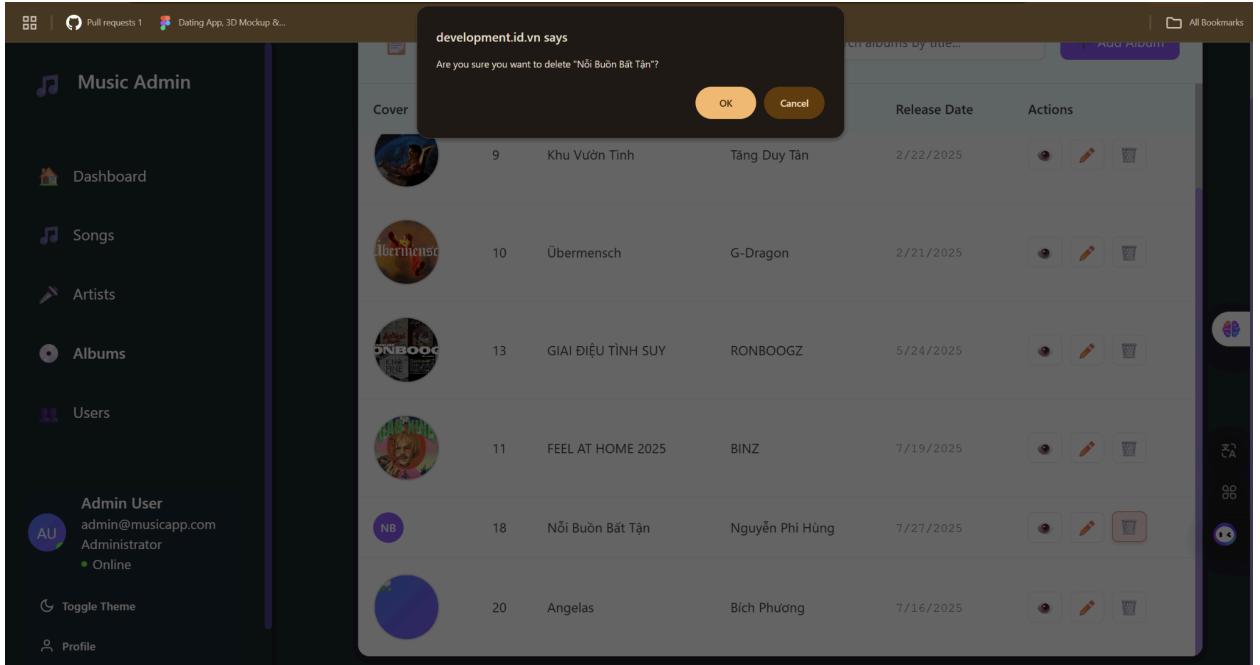
Read data: Fetch current album details and artist list.

Write data: Update album record with optional cover image replacement and metadata updates.

Client-side States : albums, isLoading, error, selectedAlbum, searchTerm, handleSelectAlbum(), fetchAlbums()

- Feature: Delete an album

Description: Remove an album from the system including its cover image and database records.



Libraries: knex, fs, multer

Server - side APIs:

- **DELETE /api/albums/:id:** Delete an album

Request Body: album_id: integer

Response: application/json

Read data: Confirm selected album before deletion.

Write data: Remove album from DB and delete cover image if exists.

Client-side States : selectedAlbum, confirmDelete, isDeleting, deleteError, successMsg, handleDeleteAlbum(), handleOpenDeleteModal(), handleCancelDelete()

6. Users

- Feature: View all users

Description: Retrieve a list of all users available in the system. This endpoint returns an array of users, each including basic information such as username, email, role, avatar image, and account status. This endpoint is typically used by admins to browse and manage all user accounts in the system.

The screenshot shows a dark-themed application interface titled "Music Admin". On the left is a sidebar with navigation links: Dashboard, Songs, Artists, Albums, and Users. Below these is a user profile section for "Admin User" with the email "admin@musicapp.com" and role "Administrator". At the bottom of the sidebar are "Toggle Theme" and "Profile" buttons. The main area is titled "Data Table (2 items)" and contains a table with two rows of user data. The columns are Avatar, ID, Username, Email, Role, Joined, and Actions. Row 1: Avatar (blue circle), ID 1, Username phung, Email phung@email.com, Role Admin, Joined 7/25/2025, Actions (Edit, Delete). Row 2: Avatar (blue circle), ID 2, Username admin, Email admin@musicapp.com, Role Admin, Joined 7/25/2025, Actions (Edit, Delete).

Avatar	ID	Username	Email	Role	Joined	Actions
	1	phung	phung@email.com	Admin	7/25/2025	
	2	admin	admin@musicapp.com	Admin	7/25/2025	

Libraries: knex, zod

Server - side APIs:

- **GET /api/users:** Retrieve a list of all users in the system

Request Body: None

Response: application/json

Table: Users

Read data: Retrieve data from the system to get all users with:

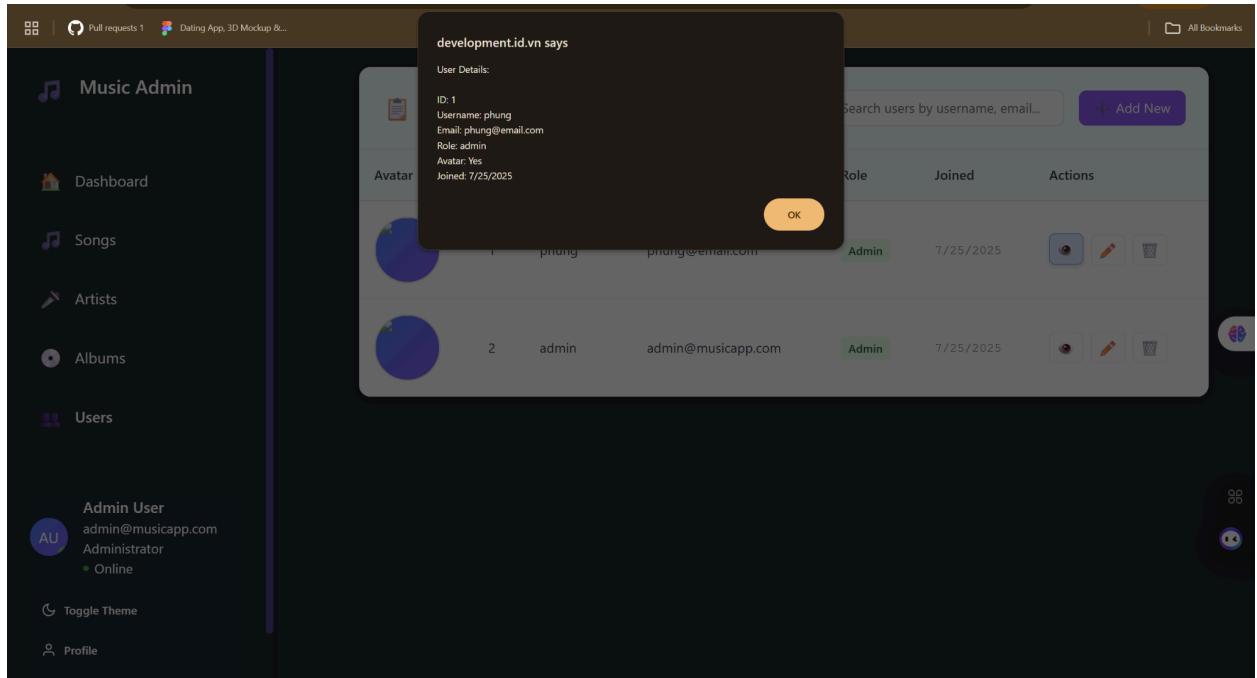
- + Basic user information (ID, username, email)
- + Role and permission levels
- + Avatar image paths
- + Account creation timestamps

Client-side States : users, selectedUser, isLoading, error, searchInput, sortField,

handleSelectUser(), fetchUsers()

- **Feature: View a user**

Description: Get detailed information about a specific user by ID.



Libraries: knex, fs

Server - side APIs:

- **GET /api/users/:id:** Get details of a specific user

Request Body: user_id: integer

Response: application/json

Tables: Users

Read data: Retrieves user ID, username, email, role, avatar (true or false), and account creation timestamp

Client-side States : users, selectedUser, isLoading, error, searchInput, sortField, handleSelectUser(), fetchUsers()

- **Feature: Delete a user**

Description: Remove a user account from the system including their profile data, avatar image, and associated records.

The screenshot shows a dark-themed web application interface titled "Music Admin". On the left is a sidebar with icons for Dashboard, Songs, Artists, Albums, and Users. Below these are sections for "Admin User" (admin@musicapp.com, Administrator, Online) and links for "Toggle Theme" and "Profile". The main content area displays a table of users with columns: Avatar, ID, Username, Email, Role, Joined, and Actions. Two users are listed: "phung" (ID 1) and "admin" (ID 2). A modal dialog box from "development.id.vn" asks "Are you sure you want to delete 'phung'?". It has "OK" and "Cancel" buttons. At the top of the page, there are browser tabs for "Pull requests 1" and "Dating App, 3D Mockup &...".

This screenshot shows the same web application after the user "phung" has been deleted. The modal dialog now displays the message "User 'phung' has been deleted successfully." The "OK" button is highlighted. The user "phung" is no longer visible in the table of users.

Libraries: knex, fs

Server - side APIs:

- **DELETE /api/users/:id:** Delete a user

Request Body: user_id: integer

Response: application/json

Read data: Fetch user details for confirmation before deletion.

Write data: Remove user record and associated avatar if exists.

Client-side States : selectedUser, userDetails, isLoading, error, fetchUserDetails(),

router