Project Report

MusicWave

Prepared By:
Karan Mewada
Enrollment No: AF04964361

Table Of Contents

Acknowledgement

The Project ‘’ **MusicWave “is** the project work carried out by

Name: **Karan Mewada** Enrollment No. **AF04964361**

I would like to express our sincere gratitude to our project guide for valuable guidance, continuous support, and encouragement throughout this project.
We are also thankful to our faculty members, family, and friends for their constant help and motivation during the development of ***MusicWave.***

Abstract

MusicWave is a full-stack music management and streaming platform designed to provide users with an engaging and personalized experience for discovering, playing, and uploading music. The system supports two main roles — regular users and administrators.

Users can sign up, log in, and browse available tracks, while admins can upload and manage music as well as oversee registered users.

The application offers a seamless and modern user interface with responsive design, built-in audio player controls, dynamic navigation, and secure authentication.
MusicWave enhances digital music accessibility while ensuring proper data management, role-based security, and a pleasant listening experience.

1.Introduction

MusicWave is a web-based music platform that allows users to explore, play, and download music seamlessly. It features role-based access, where regular users can manage their profiles and enjoy music, while administrators can upload tracks and manage users.

Built with **React**, **Node.js/Express**, and **MySQL**, MusicWave offers a **dark-modern, responsive UI**, interactive music playback with adjustable speed, and a user-friendly experience. The platform is designed for scalability, usability, and security, providing a polished and intuitive music management solution.

1.1 Background Of the study

In today's digital era, music has become one of the most accessible forms of entertainment, with millions of users relying on online platforms to discover, listen to, and manage their favorite tracks. Traditional music consumption methods, such as CDs and radio, are gradually being replaced by interactive and personalized digital platforms. However, many existing solutions lack features that combine ease of use, music management, and administrative control in a single platform.

The increasing demand for user-friendly and versatile music applications has created a need for platforms that not only allow users to explore and play music but also provide administrators with tools to manage content and users efficiently. Ensuring security, responsive design, and a seamless user experience are critical aspects of developing such applications.

MusicWave is developed to address this need by offering a web-based music management system that is both interactive and accessible. It provides users with the ability to search, play, and download music, while administrators can upload new tracks, manage users, and maintain the overall music library. By integrating modern technologies like **React**, **Node.js/Express**, and **MySQL**, MusicWave aims to deliver a scalable, secure, and engaging platform for music enthusiasts.

1.2 Motivation

The rapid growth of digital media consumption has transformed how people access and enjoy music. Despite the abundance of music platforms, many lack a seamless integration of user-friendly features and effective administrative control, often resulting in fragmented experiences for both users and content managers.

The motivation behind developing **MusicWave** stems from the need to create an interactive and unified music platform that caters to both music enthusiasts and administrators. On the user side, the platform aims to provide a simple and enjoyable way to search, play, and download music, while maintaining a responsive and visually appealing interface. For administrators, it offers tools to efficiently manage users, monitor content, and maintain the music library without complications.

Furthermore, the project seeks to enhance practical skills in full-stack development, combining front-end and back-end technologies, database management, and user authentication. By creating MusicWave, the project aspires to bridge the gap between accessibility, functionality, and modern web design in music management applications.

1.3 Significance of the study

The development of **MusicWave** holds significance for multiple stakeholders in the digital music ecosystem:

For Users: MusicWave provides an accessible, intuitive, and engaging platform for discovering, listening to, and downloading music. The responsive design and modern interface improve user experience, making music exploration effortless and enjoyable.

For Administrators: The platform enables efficient management of both music content and users. Admins can upload, delete, and monitor music tracks, as well as manage user accounts, ensuring a secure and organized system.

For Developers and Students: MusicWave serves as a practical example of full-stack web development, integrating front-end technologies like React with back-end APIs and database management. It demonstrates the application of authentication, authorization, and responsive design principles.

For the Industry: By addressing common gaps in usability and content management, MusicWave highlights the importance of combining user-centered design with robust administrative tools, contributing to improved practices in music streaming and management platforms.

1.4 Features of MusicWave

1. User Authentication and Authorization

- Users can **sign up** and **log in** securely.
- Admins have a separate **admin login** to access the dashboard.
- Supports session management using **local Storage** to remember logged-in users.

2. Music Discovery and Playback

- Users can **browse a collection of music tracks**.
- Built-in **music player** with play/pause controls.
- Users can **adjust playback speed** (0.5x to 2x).
- Music tracks can be **downloaded** directly from the platform.

3. Search Functionality

- Users can **search music by title or artist**.
- Real-time filtering makes it easy to find preferred tracks.

4. User Profile Management

- Users can **view their profile information** (username, email).
- Users can **change their password** securely.

5. Admin Dashboard

- Admins can **upload new music**, specifying title, artist, and file.

- Admins can **view, edit, and delete music tracks**.
- Admins can **view registered users** and delete users if needed.
- Provides a **centralized interface** for managing the platform efficiently.

6. Responsive and Modern UI

- Fully **responsive design** for desktop and mobile devices.
- Modern look with **gradients, hover effects, and polished typography**.
- User-friendly navigation and **consistent styling across pages**.

7. Security Measures

- Passwords are handled securely (backend encryption).
- Only admins have access to administrative functions.
- Email input fields include **HTML5 validation** to ensure proper email format.

8. Scalability and Extensibility

- Built with **React** for modular, component-based development.
- Backend-ready API calls allow **easy integration with future features**.
- Designed to **add new features** like playlists, ratings, or comments in the future.

1.5 Expected Outcomes

Enhanced Music Access and Enjoyment

- Users can easily **discover, play, and download music** from a centralized platform.
- Music playback with adjustable speed allows **personalized listening experiences**.

Efficient User Management

- Admins can **view, manage, and delete users** as needed.
- Users have **secure access to their profiles** and can update their passwords.

Seamless Administrative Control

- Admins can **upload, view, and remove music tracks** easily.
- A clear and responsive admin dashboard ensures **efficient platform management**.

Improved User Interaction and Engageme

- Search functionality enables users to **quickly find music by title or artist**.
- Intuitive and visually appealing UI enhances **user satisfaction and retention**.

Reliable Security and Validation

- Users can only sign up with **valid emails**, and passwords are handled securely.
- Admin and user privileges are **properly enforced**, ensuring platform integrity.

Responsive and Accessible Platform

- The web app works well on **desktop, tablet, and mobile devices**.
- Modern UI design improves **user experience and accessibility**.

Foundation for Future Expansion

- Modular architecture allows for **easy integration of new features** like playlists, ratings, or comments.
- Provides a scalable foundation for **growth and enhancement** of the music platform.

2.System analysis

The **MusicWave** system is a web-based music management and playback platform. It allows users to discover, listen to, and download music, while providing administrators with the tools to manage users and music content effectively. The system is designed for simplicity, efficiency, and scalability, ensuring a seamless experience across devices.

2.1 Problem Definition

- Users often face **difficulty in accessing and organizing music** across multiple sources.
- There is a lack of **centralized platforms** for both music consumption and administration.
- Admins struggle with **user management and music content control** on conventional platforms.
- Many existing systems lack **personalized playback controls**, such as adjustable speed.

2.2 Objectives of the System

- To provide a **centralized web platform** for managing and listening to music.
- To allow **user registration and secure authentication**.
- To provide **admin functionalities** including user management, music uploads, and deletions.
- To offer **interactive features** like search, playback control, and music download.
- To ensure **responsiveness and accessibility** across different devices.

2.3 system requirement

Functional Requirements

- User registration and login with email validation
- Secure user authentication (user/admin)
- Profile update and password change
- Music play, pause, and speed contro
- Music download and search feature
- Admin dashboard for management
- Admin can upload/delete music
- Admin can view/delete users
- Responsive navigation for all devices

Non-Functional Requirements

- Fast loading and smooth performan
- Secure data handling and password protection
- User-friendly and responsive UI
- Reliable and stable system
- Scalable for future updates
- Compatible with major browser
- Easy to maintain and update

Hardware Requirements

User Side (Client):

- **Processor:** Intel i3 or equivalent
- **RAM:** 4 GB minimum
- **Storage:** 500 MB free space
- **Network:** Stable internet connection
- **Device:** PC, laptop, or smartphone

Server Side:

- **Processor:** Intel i5 or equivale
- **RAM:** 8 GB or mo
- **Storage:** 50 GB or more for music files and databaseNetwork: High-speed internet
- **OS:** Windows or Linux server environment

Software Requirements

Frontend:

- **React.js** for building the user interface
- **HTML5, CSS3** for structure and styling
- **Axios** for API requests
- **React Router** for navigation

Backend:

- **Node.js** for server runtime
- **Express.js** framework

Database:

- **MySQL** or any relational database for storing users and music information

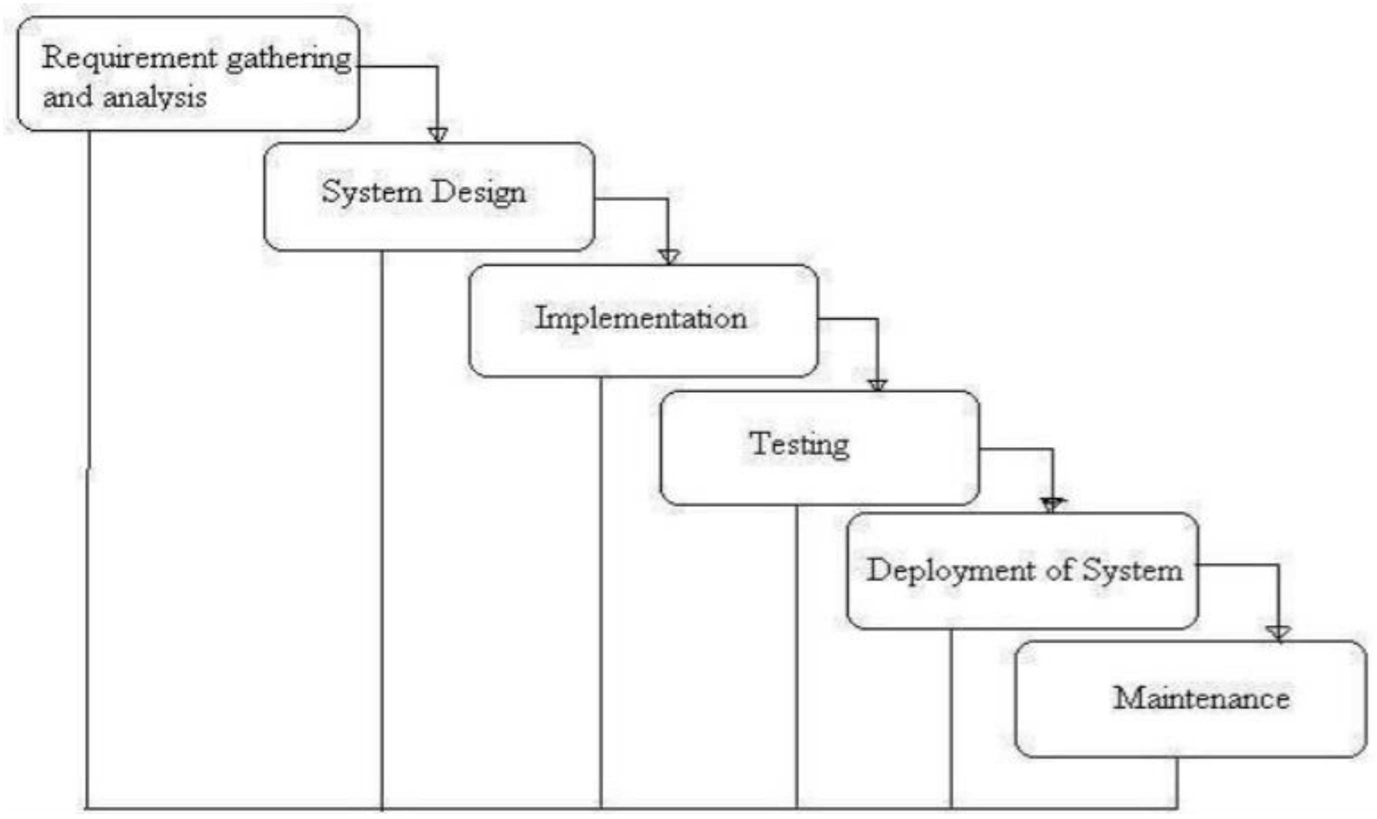
Development Tools:

- Vite.js as the build tool
- Visual Studio Code as IDE
- Git/GitHub for version control

Browser Requirements:

- Google Chrome, Mozilla Firefox, Microsoft Edge

2.4 SDLC



Phases of Development

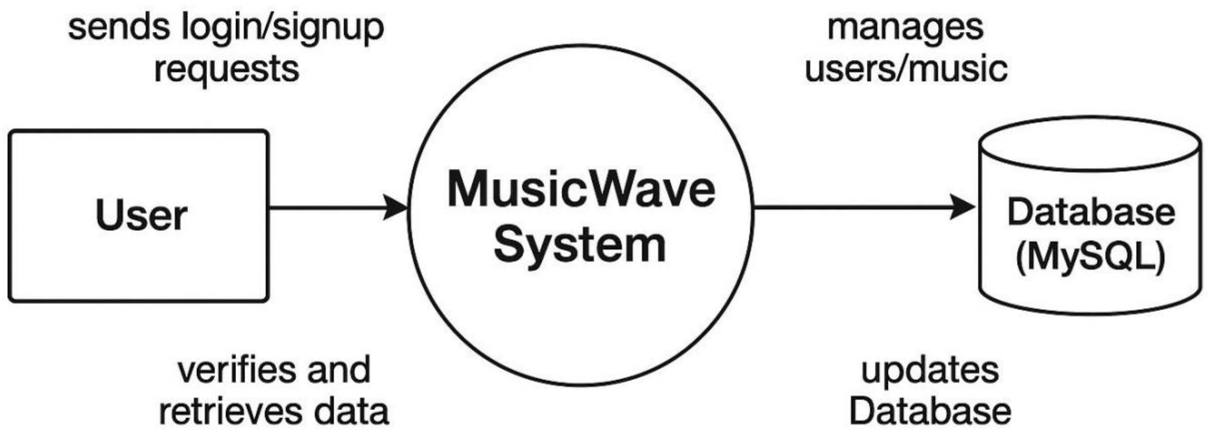
1. **Requirement Analysis & System Study**
   - Identifying project goals, challenges, and functional specifications.
   - Gathering stakeholder requirements and defining core functionalities.
2. **System Design**
   - Structuring the **database, modules, and architecture**.
   - Design **user interfaces** for optimal accessibility.
3. **Implementation (Coding)**
   - Backend development using **Python (Django)**.

   - Frontend design using **HTML, CSS, and Bootstrap**.
   - Database integration with **MySQL**.

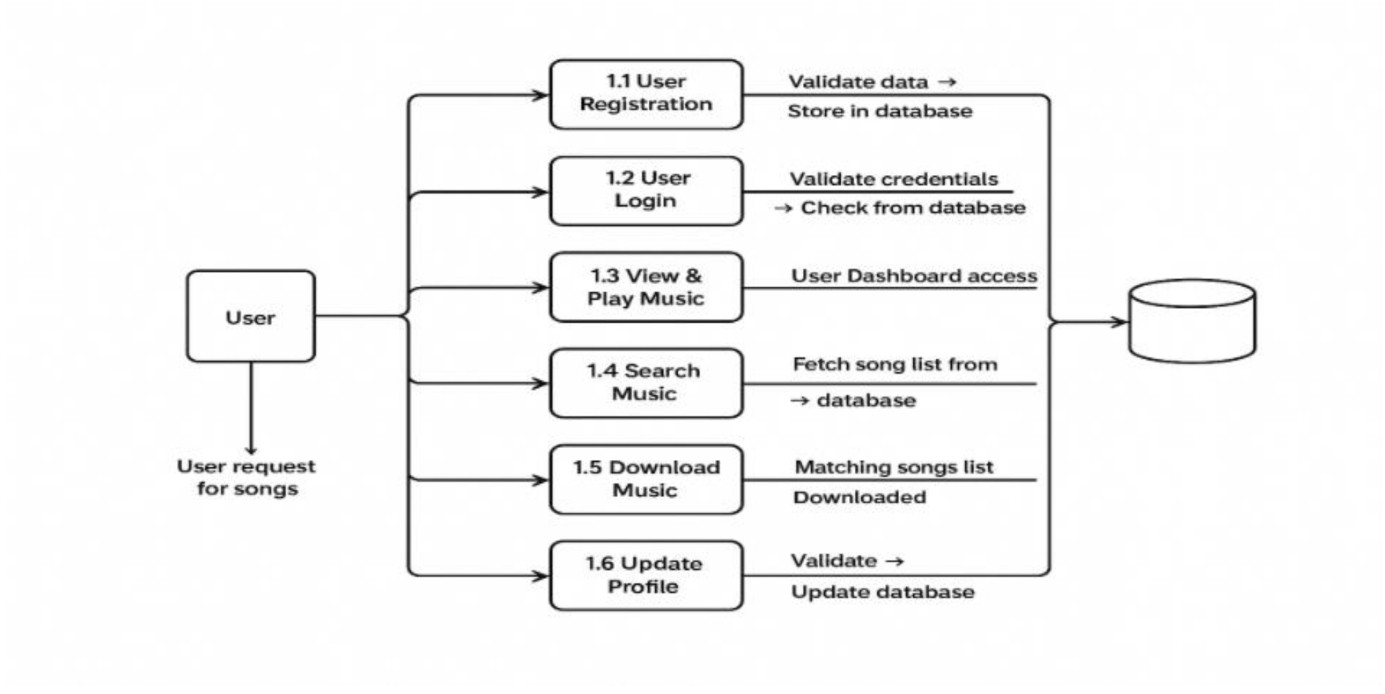
4. **Testing & Debugging**
   - Unit testing, integration testing, and usability checks.
   - Debugging for performance improvements.
5. **Deployment & Maintenance**
   - Hosting on a scalable environment.
   - Continuous updates for feature enhancements.

2.5 Data Flow Diagram

Zero Level

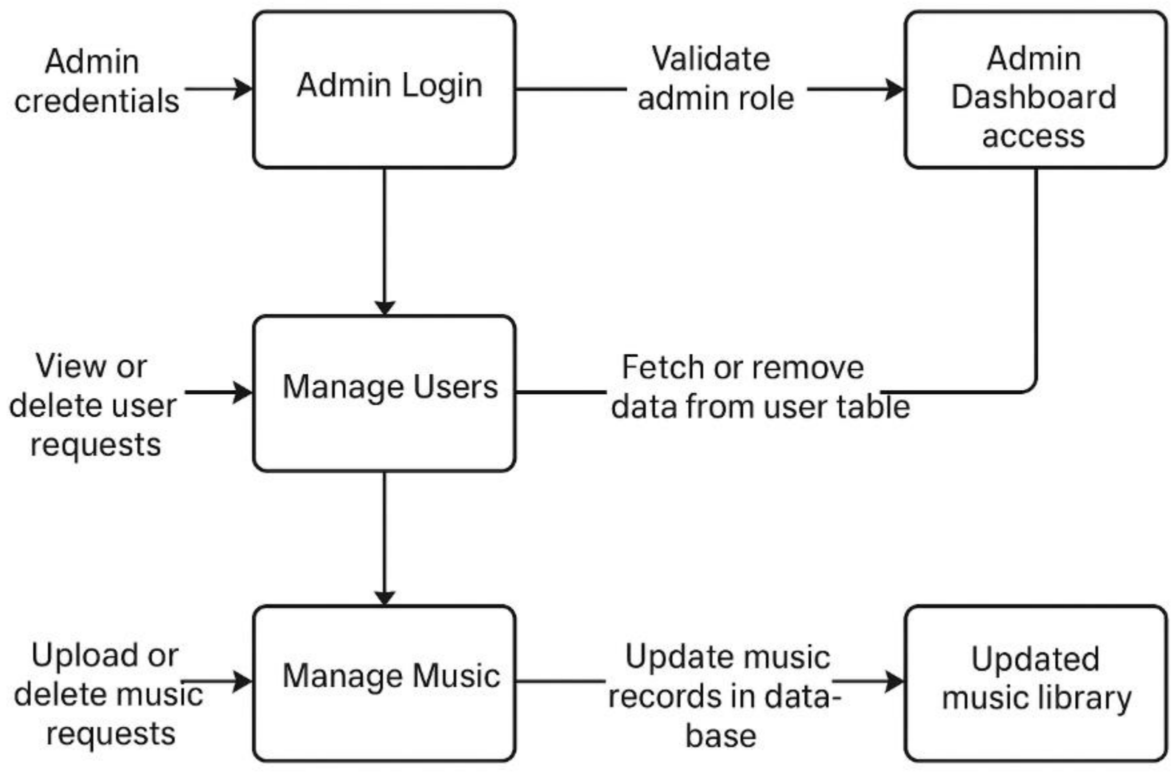


1st Level



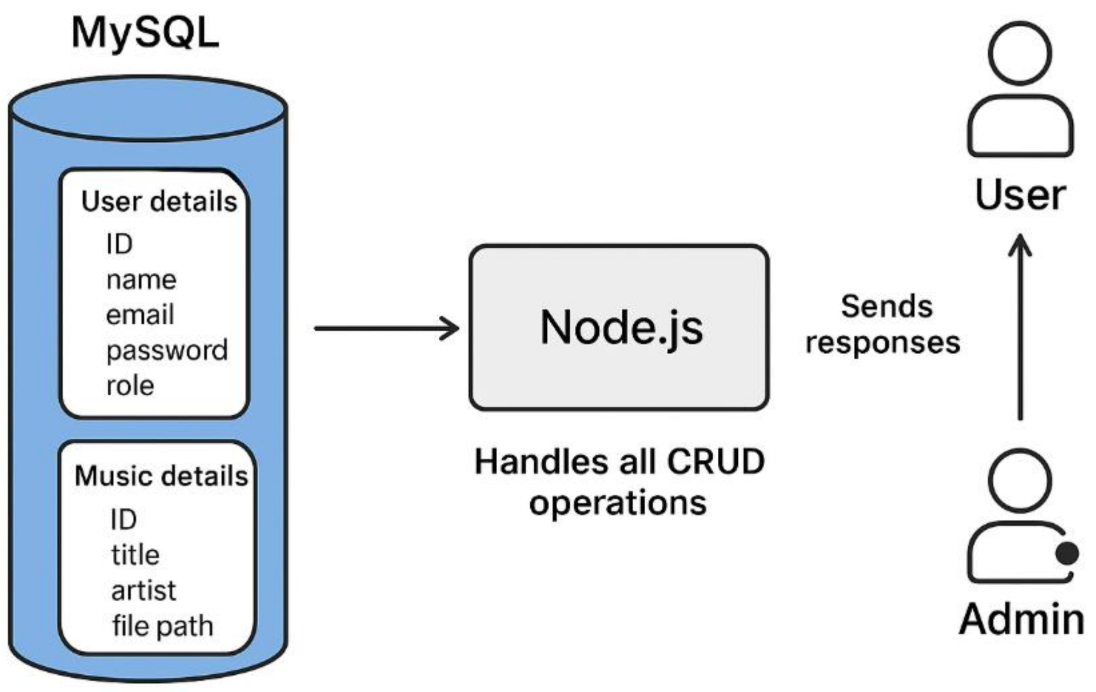
User
User request for songs
1.1 User Registration
Validate data → Store in database
1.2 User Login
Validate credentials → Check from database
1.3 View & Play Music
User Dashboard access
1.4 Search Music
Fetch song list from → database
1.5 Download Music
Matching songs list Downloaded
1.6 Update Profile
Validate → Update database

2nd Level



Admin credentials
Admin Login
Validate admin role
Admin Dashboard access
View or delete user requests
Manage Users
Fetch or remove data from user table
Upload or delete music requests
Manage Music
Update music records in data-base
Updated music library

Data Level


MySQL
User details
ID
name
email
password
role
Music details
ID
title
artist
file path
Node.js
Handles all CRUD operations
Sends responses
User
Admin

3. System Design

3.1 Modules

1. User Module

Purpose: Handles all functionalities for a standard user.

Features:

- User registration and login
- View music list
- Search music by title or artist
- Play music tracks
- Download music
- View and update profile
- Change password

2. Admin Module

Purpose: Allows administrators to manage users and music.

Features:

- Admin login
- View all registered users
- Delete users
- Upload new music
- Delete existing music
- View music library

3. Music Player Module

Purpose: Handles music playback functionalities.

Features:

- Play/Pause control
- Adjust playback speed (0.5x – 2x)
- Download music

- Single track play handling (only one track plays at a time)

4. Authentication Module

Purpose: Ensures secure login/signup and session management.

Features:

- User login/signup with validation
- Admin login
- Role-based access control (admin vs user)
- Session storage in browser (local Storage)

5. Music Management Module

Purpose: Handles storing, retrieving, and organizing music files.

Features:

- Upload music with title and artist details
- Fetch music list for users
- Search music library
- Delete music (admin only)

6. Search Module

Purpose: Enables users to easily find music.

Features:

- Search by song title
- Search by artist name
- Display matching results dynamically

7.Playlist Module

Purpose

- Allows users to create and manage personalized playlists.
- Enables adding, viewing, and removing songs dynamically.
- Enhances user experience by organizing favorite music.

Features

- Create new playlists with custom names.
- View all playlists of a logged-in user.
- Add songs from the music library to playlists.
- View songs contained in a selected playlist.
- Remove unwanted songs from playlists.
- Play songs directly from the playlist using the music player.

8. UI/Frontend Module

Purpose: Provides the interface for users and admin.

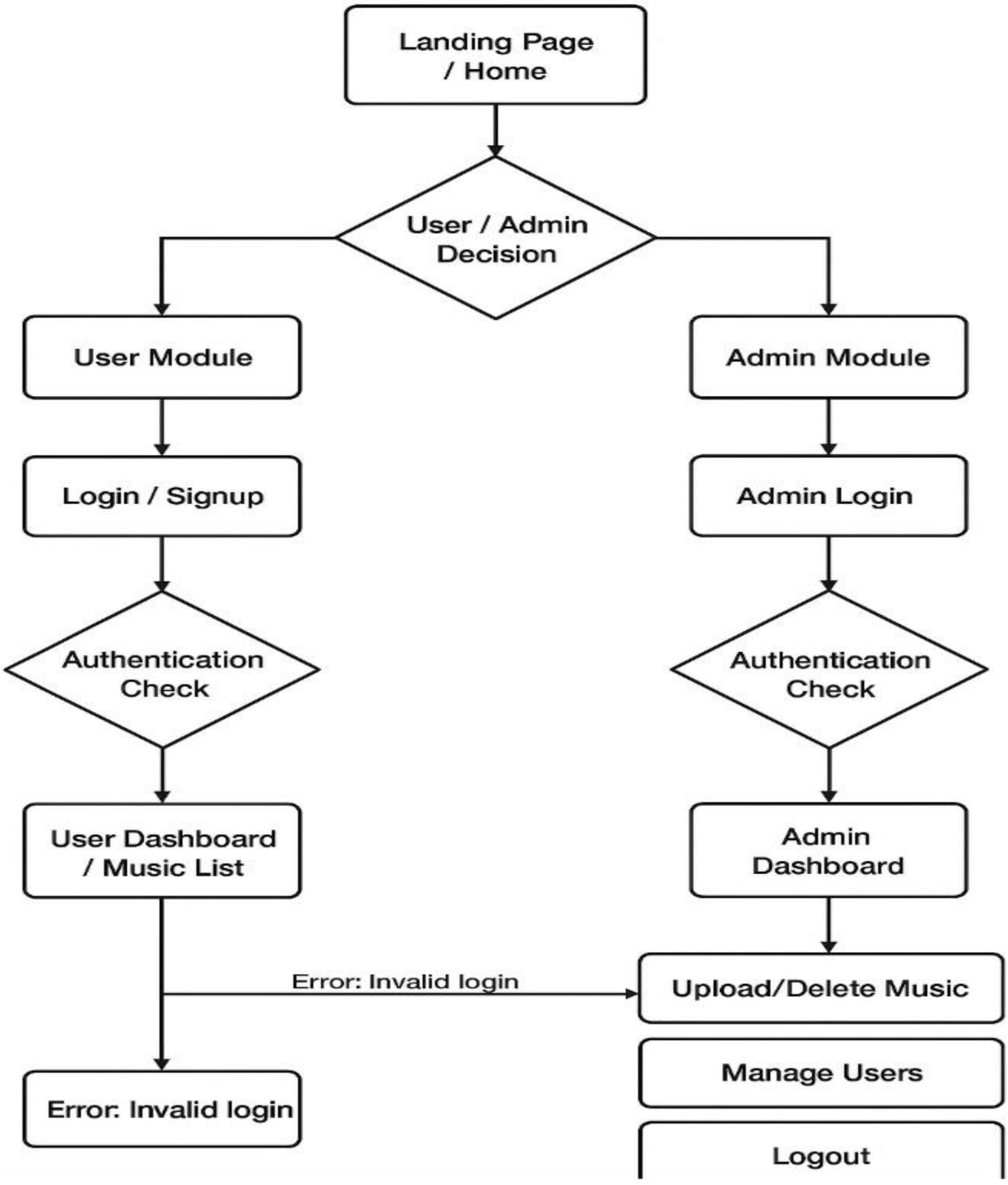
Features:

- Responsive navigation header
- Home page with introduction and explore music button
- Music list and player interface
- Forms for login, signup, and profile management
- Admin dashboard with tables for users and music

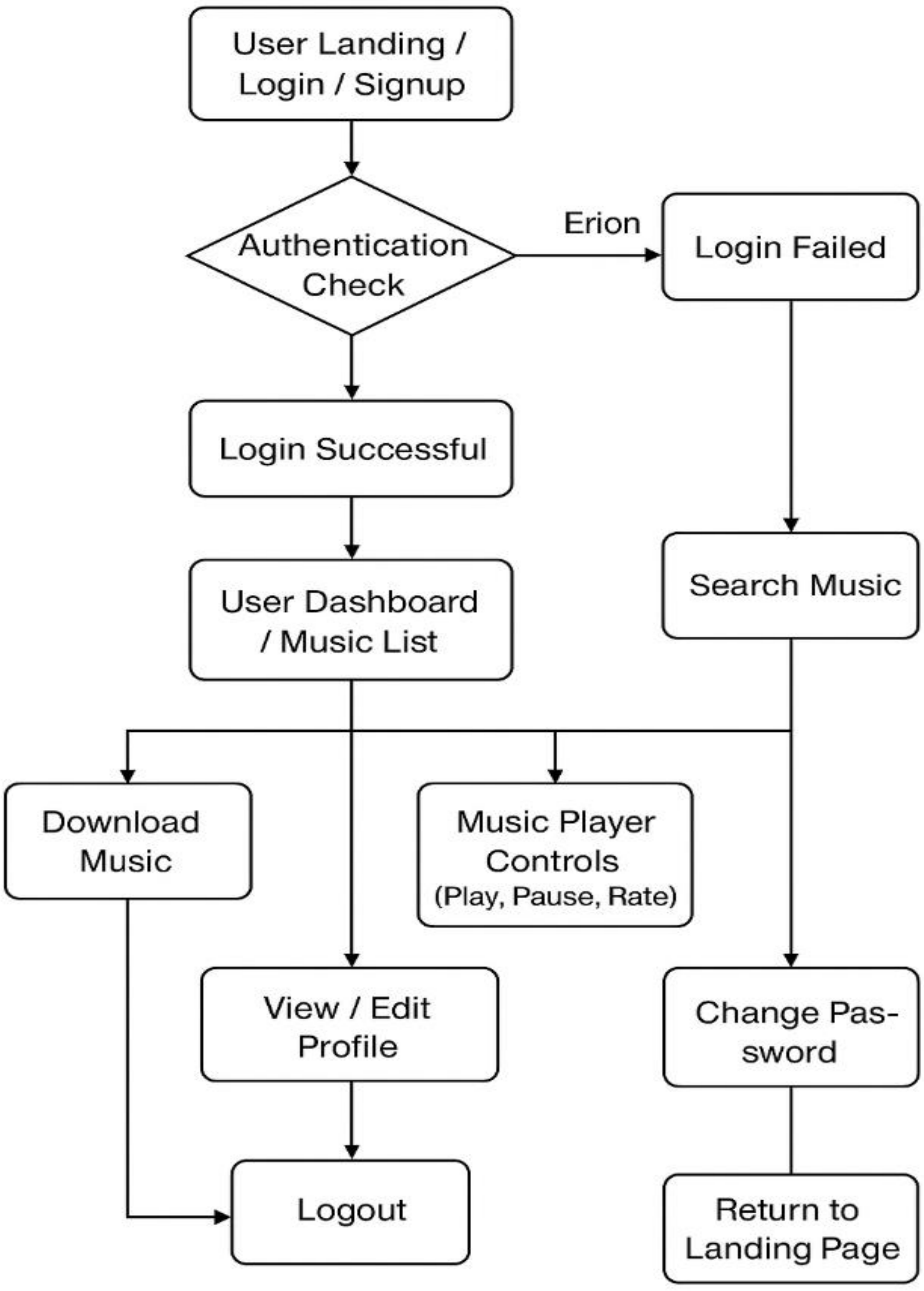
3.2 procedural design

1. flowchart

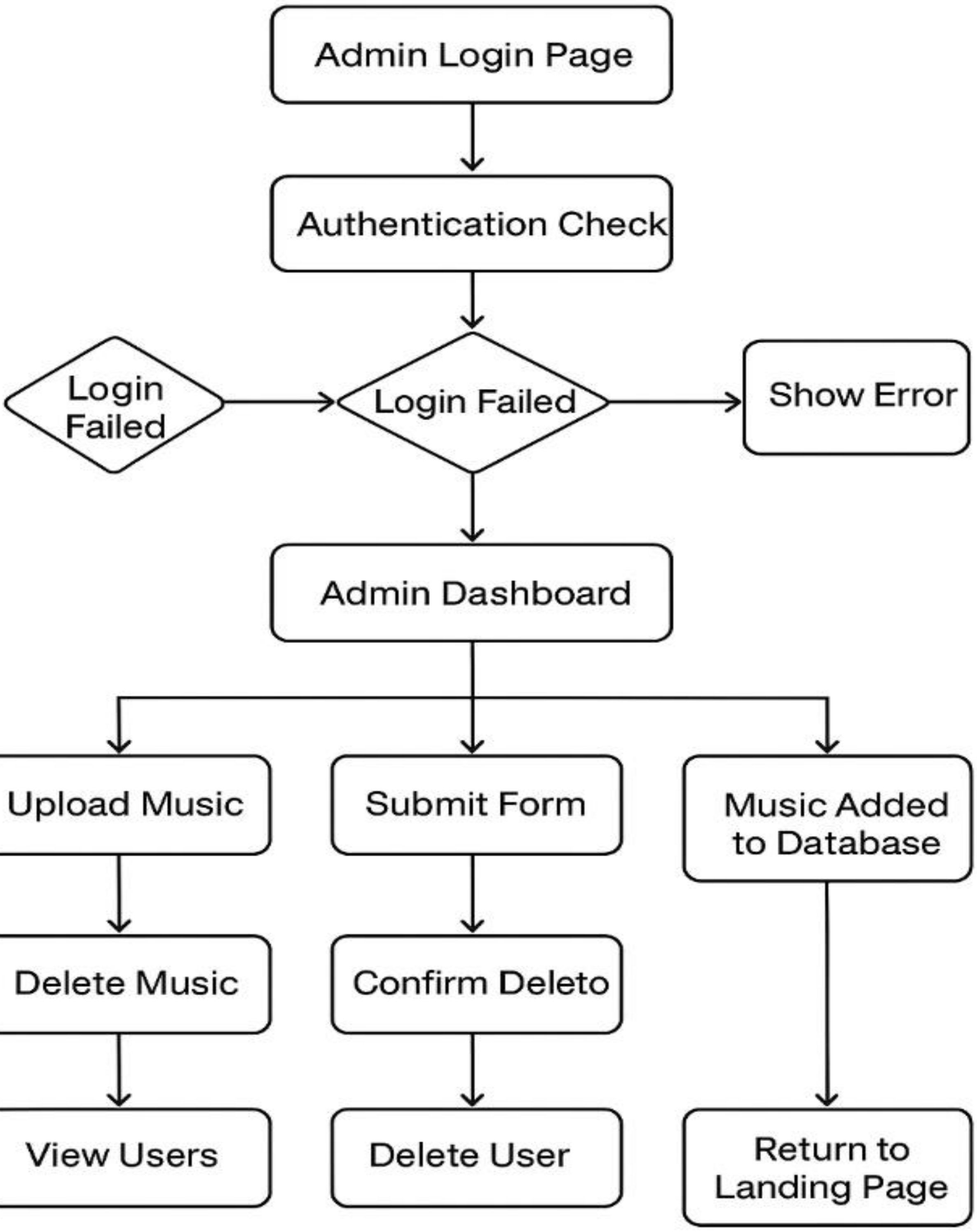
1.1 Overview



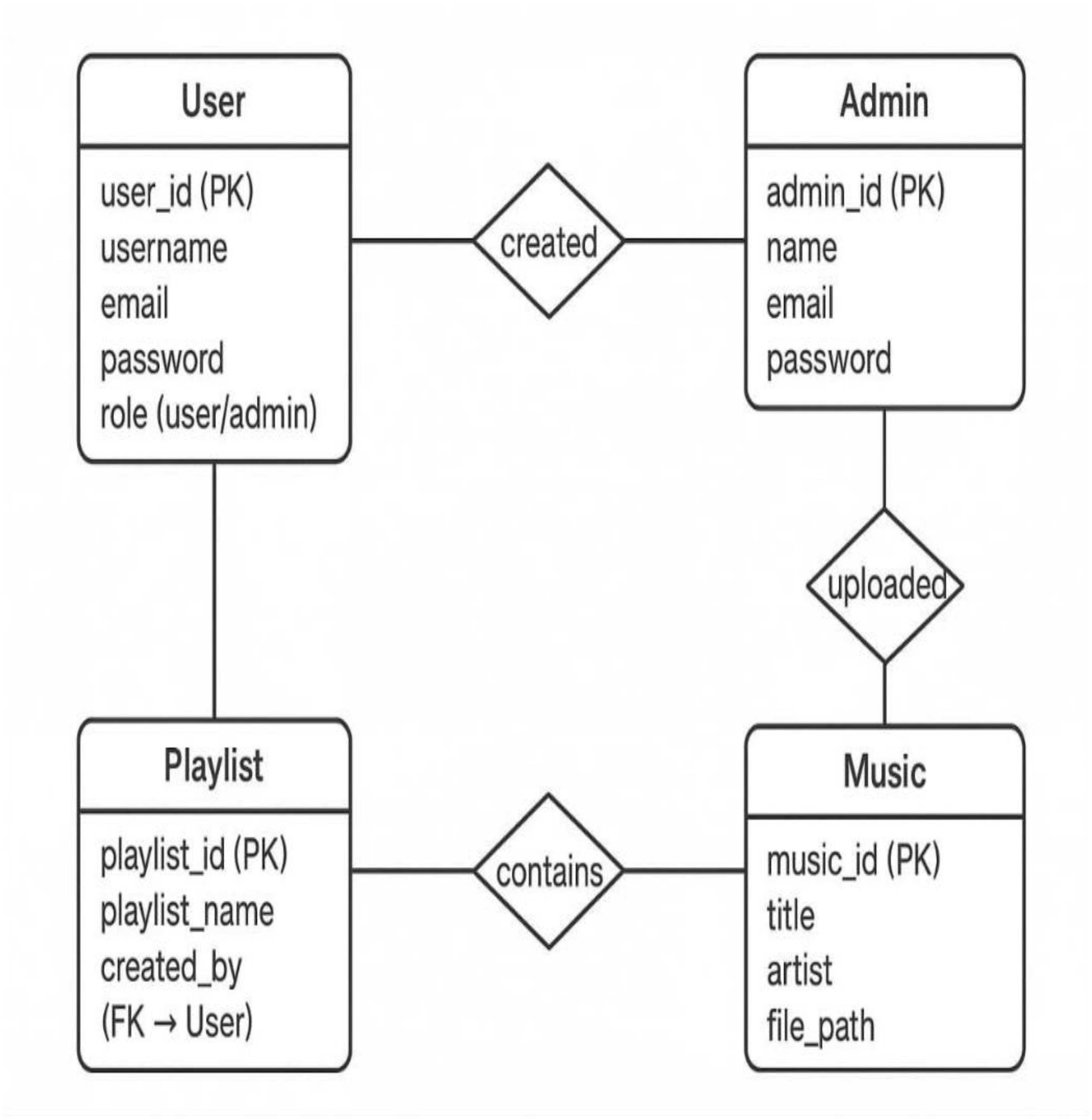
1.2.User


User Landing / Login / Signup
Authentication Check
Erion
Login Failed
Login Successful
User Dashboard / Music List
Search Music
Download Music
Music Player Controls (Play, Pause, Rate)
View / Edit Profile
Change Pas-sword
Logout
Return to Landing Page

1.3 Admin

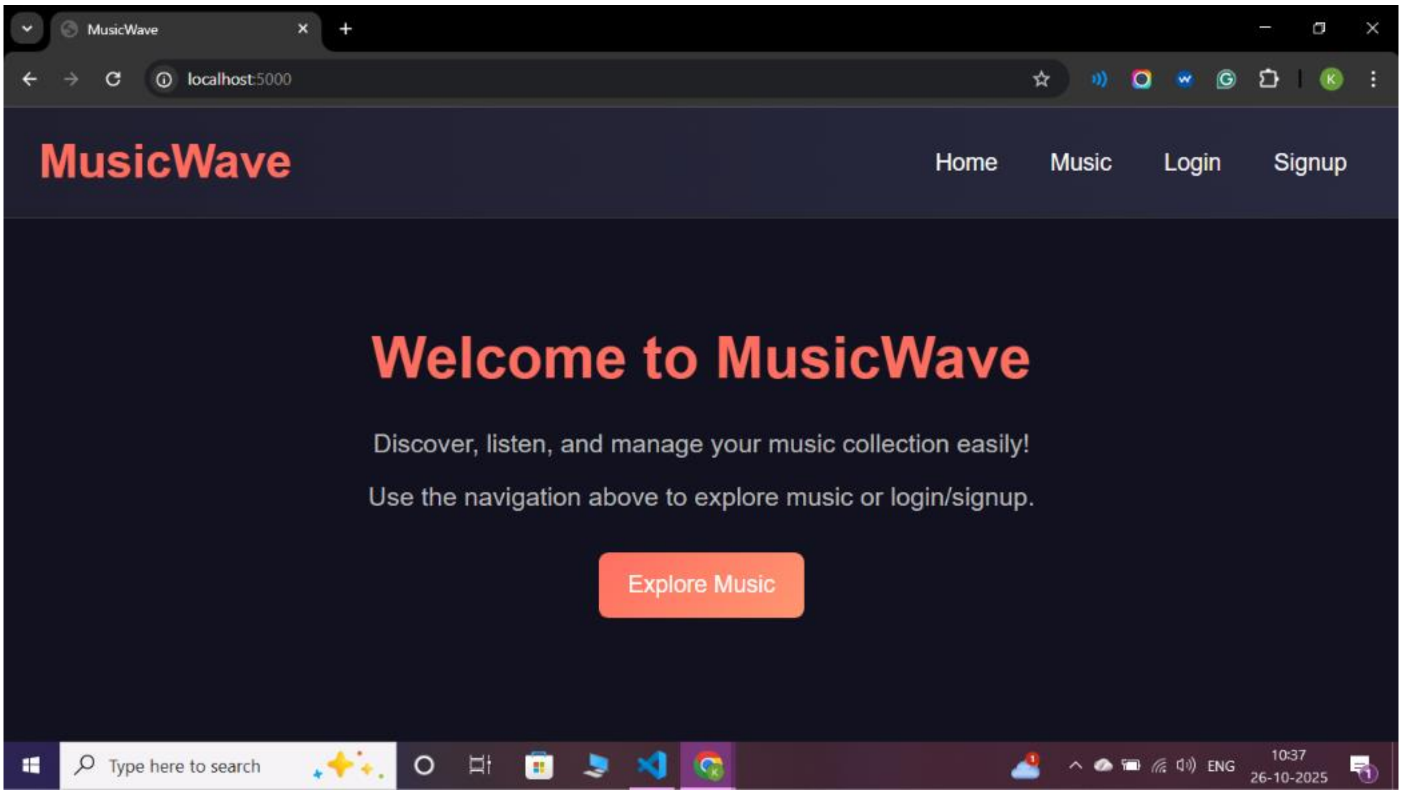

Admin Login Page
Authentication Check
Login Failed
Login Failed
Show Error
Admin Dashboard
Upload Music
Submit Form
Music Added to Database
Delete Music
Confirm Deleto
View Users
Delete User
Return to Landing Page

2.ER Diagram


User
user_id (PK)
username
email
password
role (user/admin)
created
Admin
admin_id (PK)
name
email
password
uploaded
Playlist
playlist_id (PK)
playlist_name
created_by
(FK → User)
contains
Music
music_id (PK)
title
artist
file_path

3.ScreenShots

1.Home Page

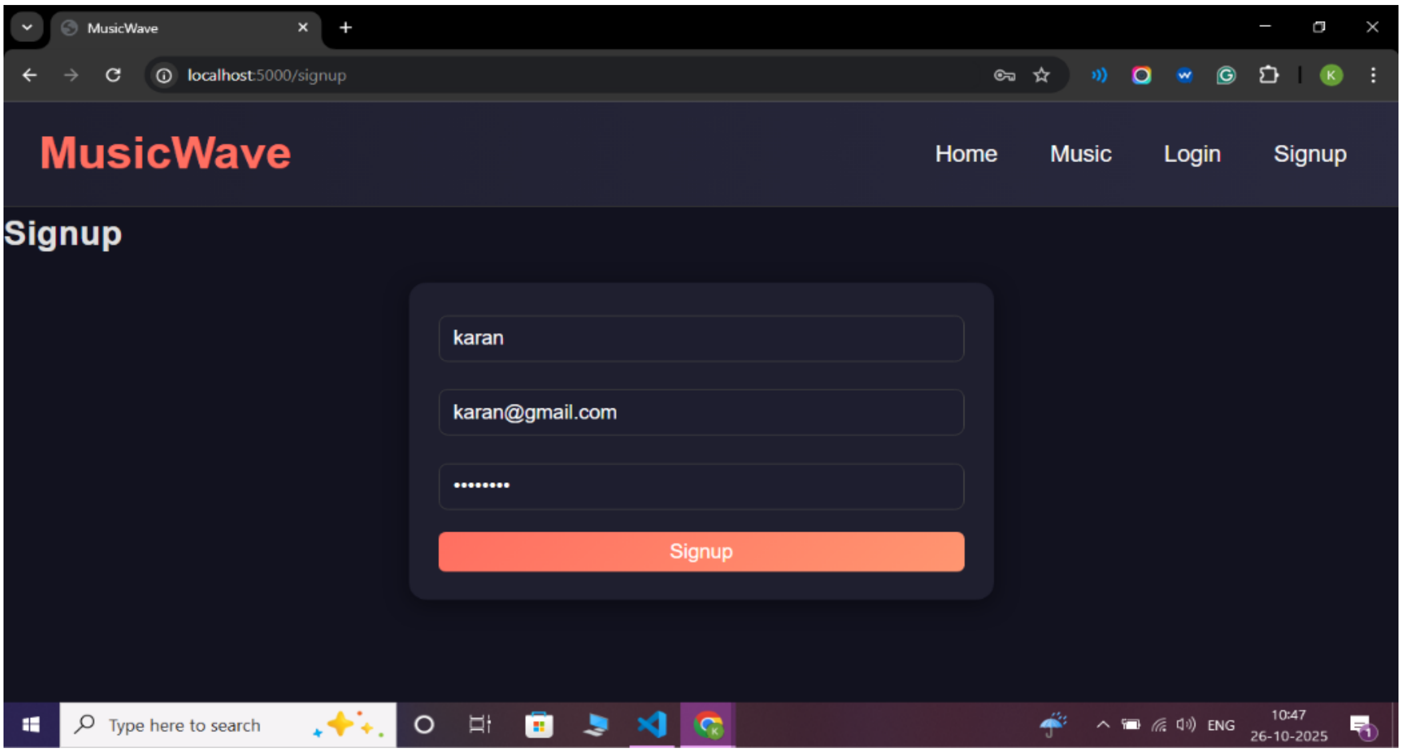


Welcome to MusicWave

Discover, listen, and manage your music collection easily!

Use the navigation above to explore music or login/signup.

2.SignUp Page



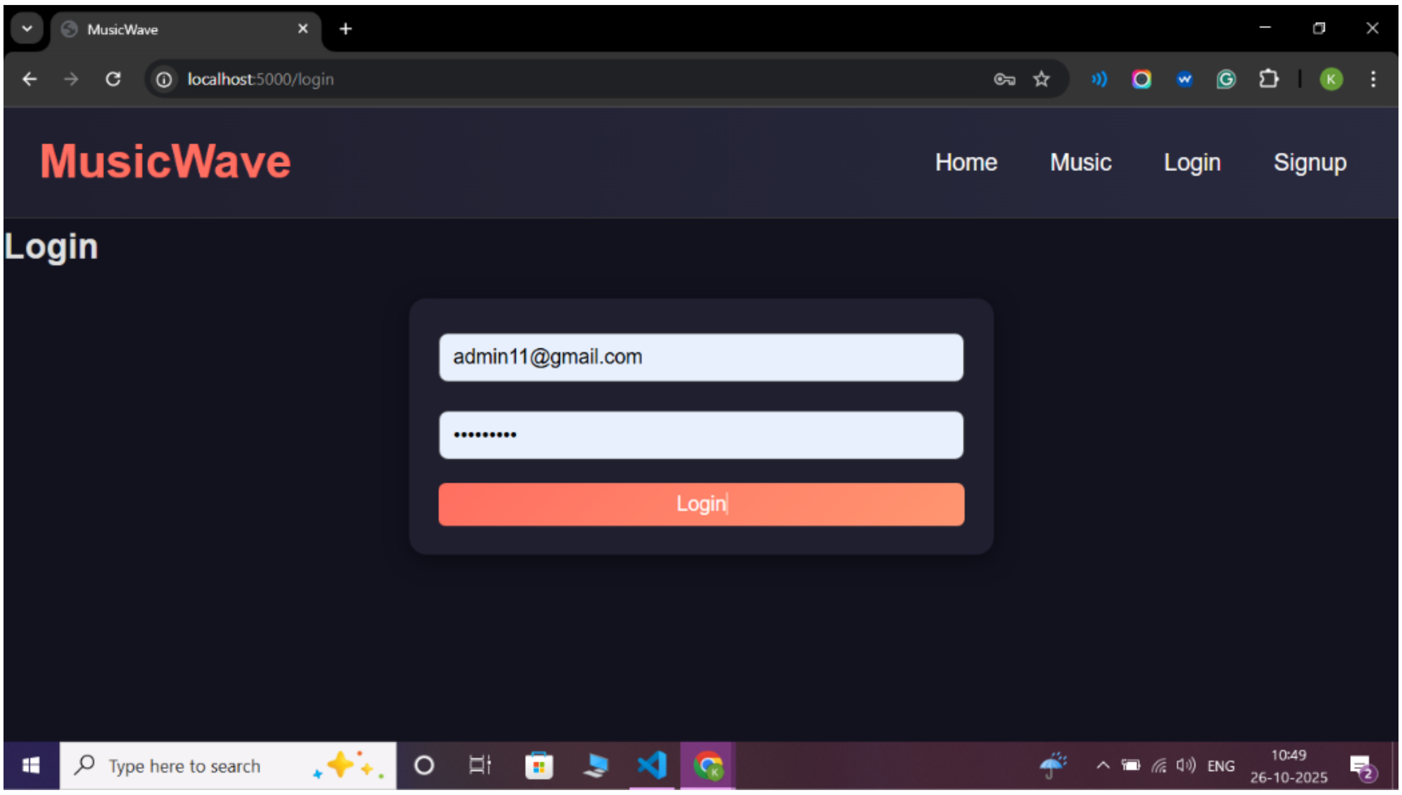
Signup

karan

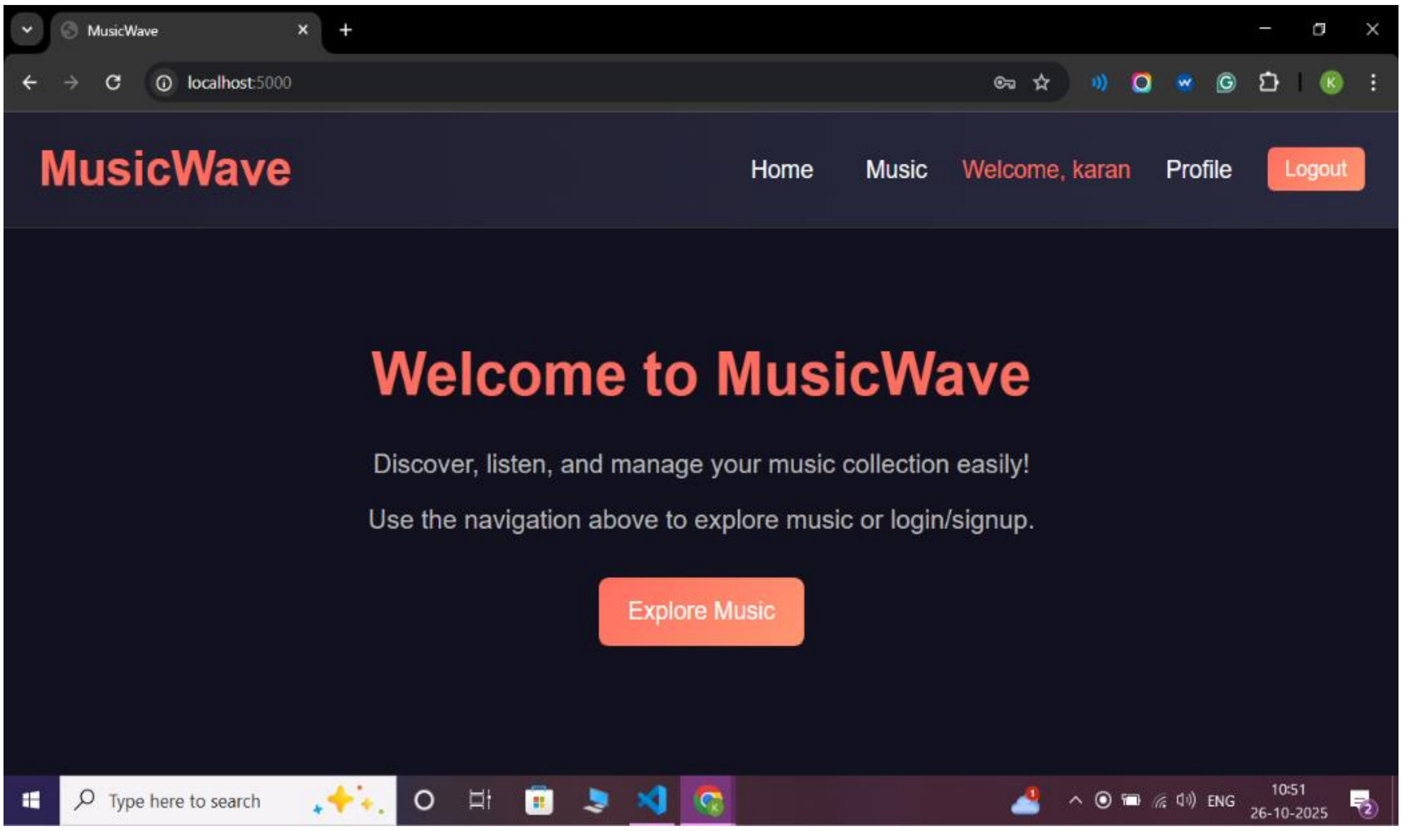
karan@gmail.com

••••••••

3.Login Page



4.User Login

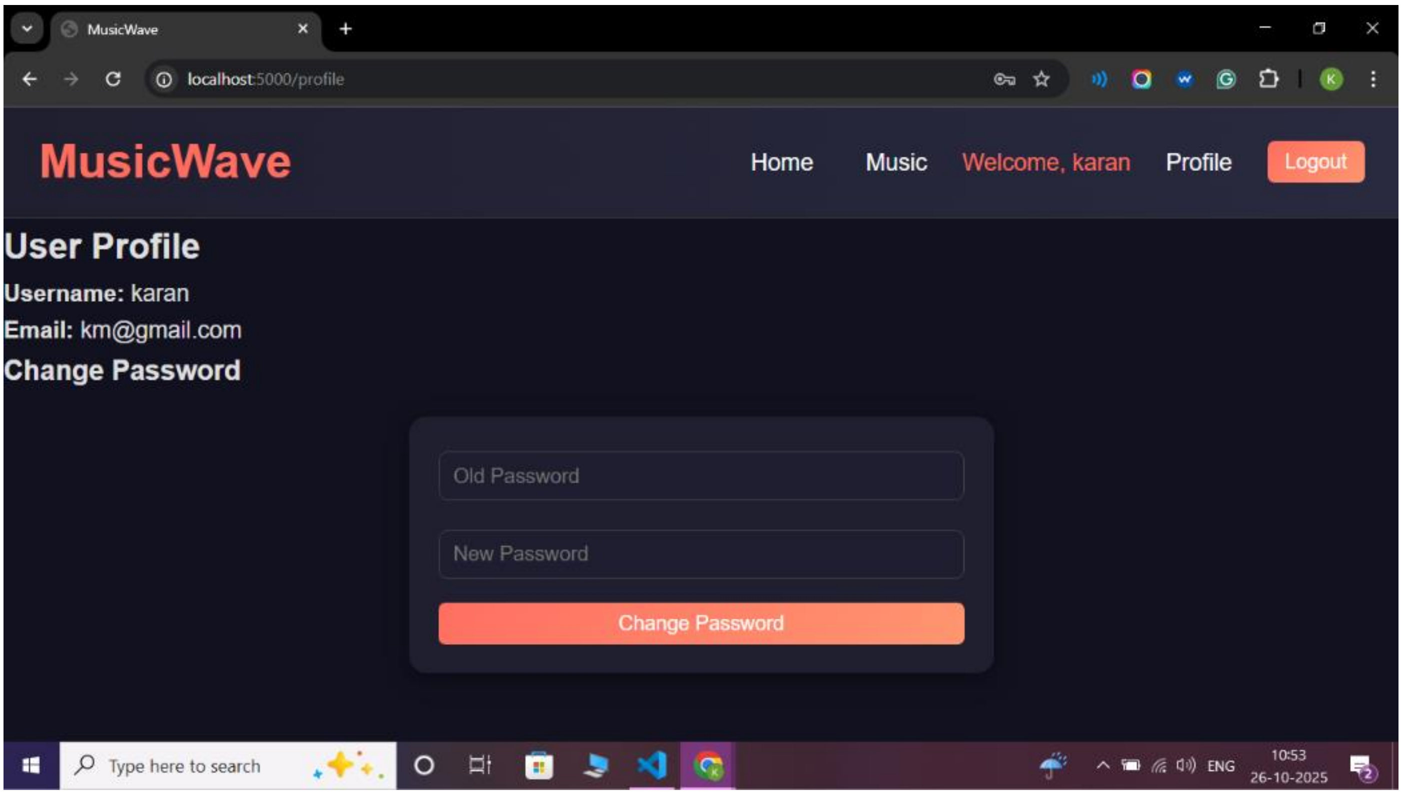


Welcome to MusicWave

Discover, listen, and manage your music collection easily!

Use the navigation above to explore music or login/signup.

5.User Profile Page



MusicWave

localhost:5000/profile

MusicWave | Home | Music | Welcome, karan | Profile | Logout

User Profile

Username: karan

Email: km@gmail.com

Change Password

Old Password

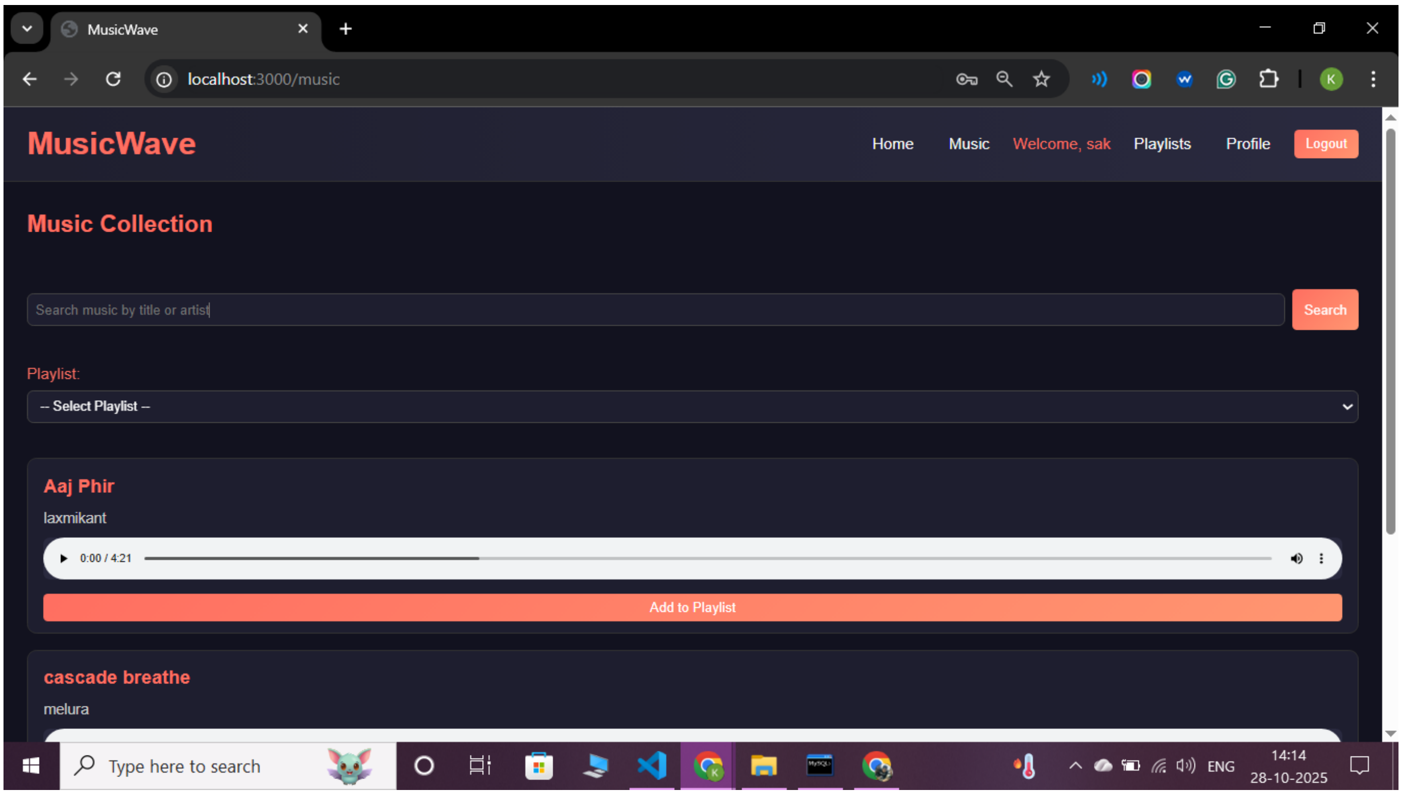
New Password

Change Password

Type here to search

10:53 26-10-2025

6.Music Page



MusicWave

localhost:3000/music

MusicWave | Home | Music | Welcome, sak | Playlists | Profile | Logout

Music Collection

Search music by title or artist | Search

Playlist:

-- Select Playlist --

Aaj Phir

laxmikant

0:00 / 4:21

Add to Playlist

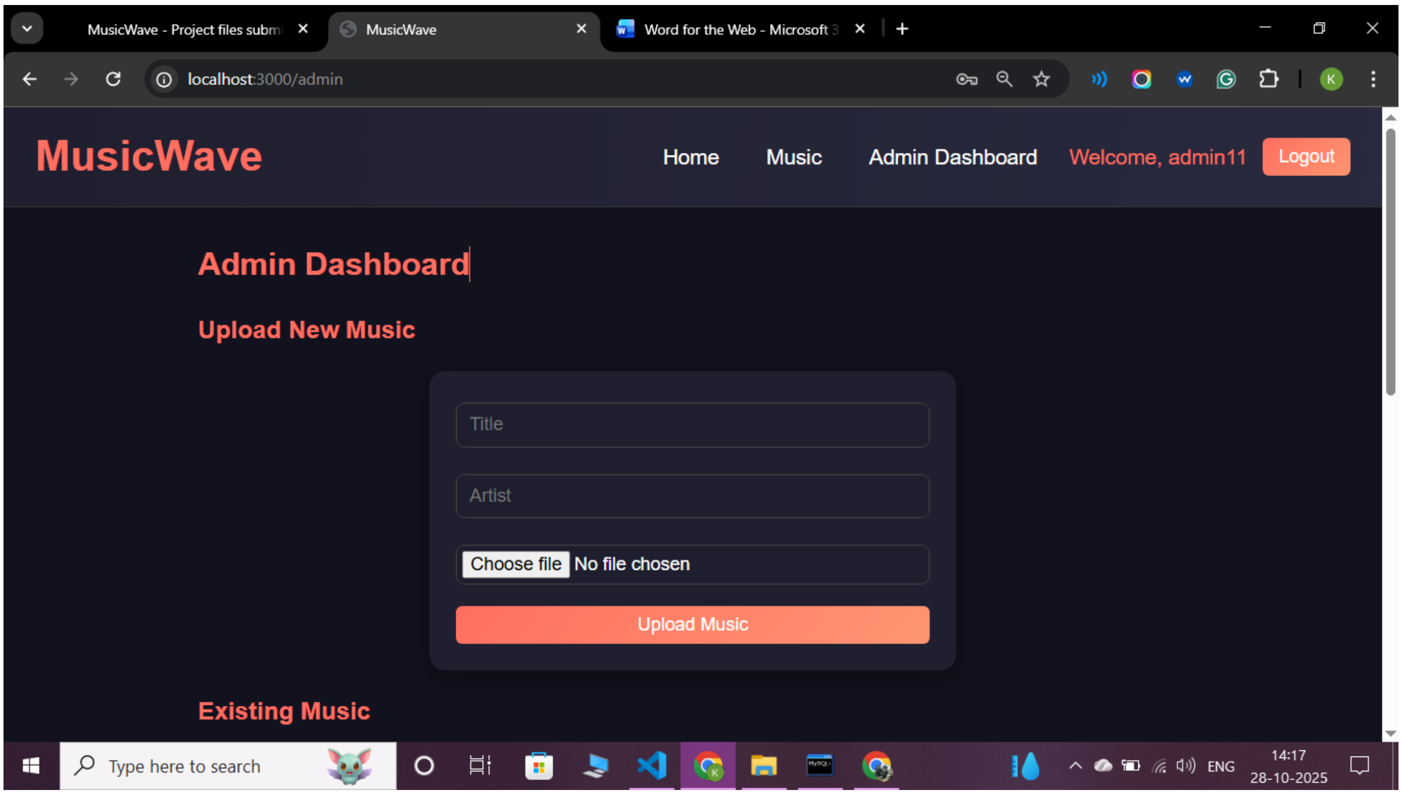
cascade breathe

melura

Type here to search

14:14 28-10-2025

7. Admin Dashboard Page



Admin Dashboard

Upload New Music

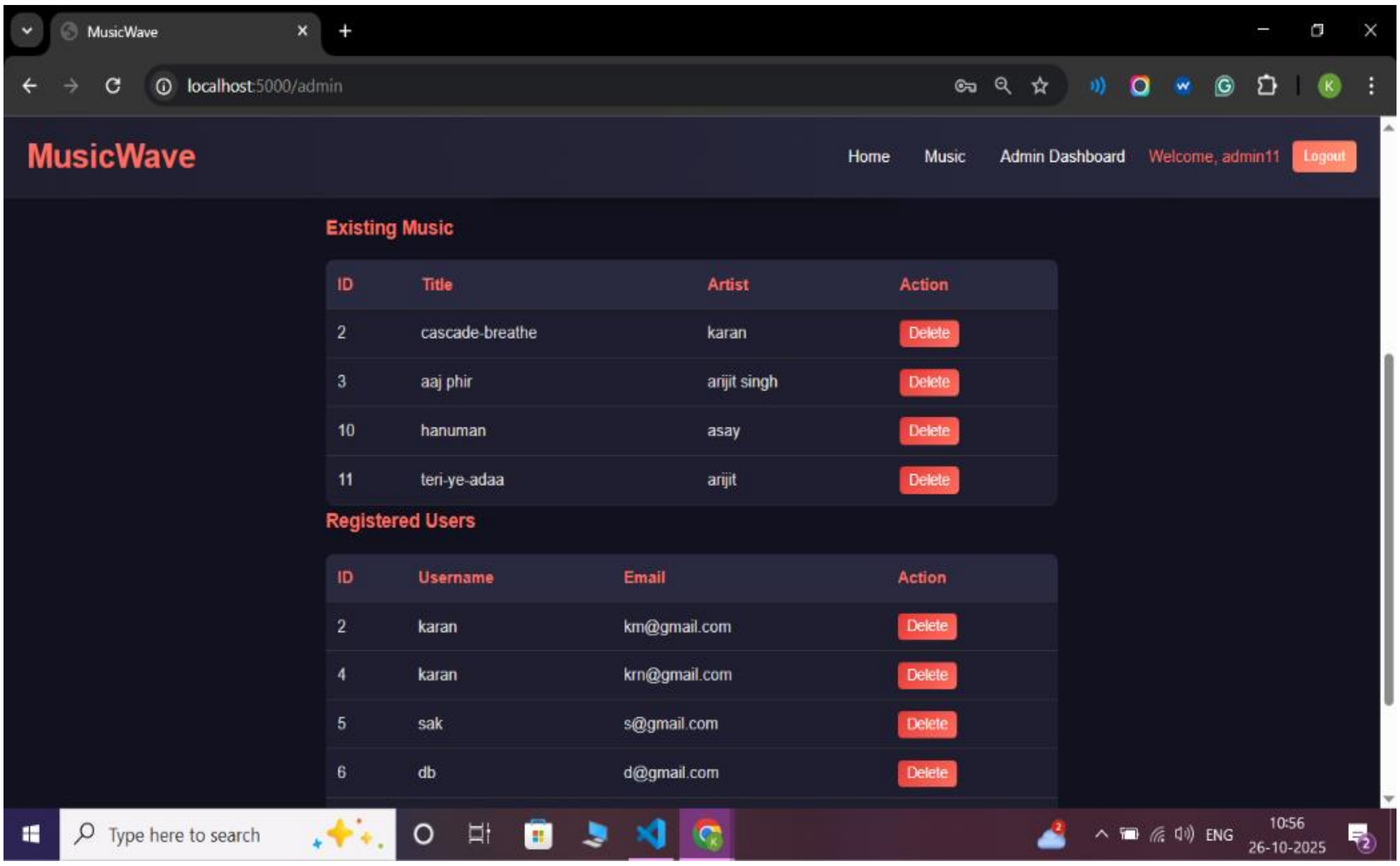
Title

Artist

Choose file No file chosen

Upload Music

Existing Music



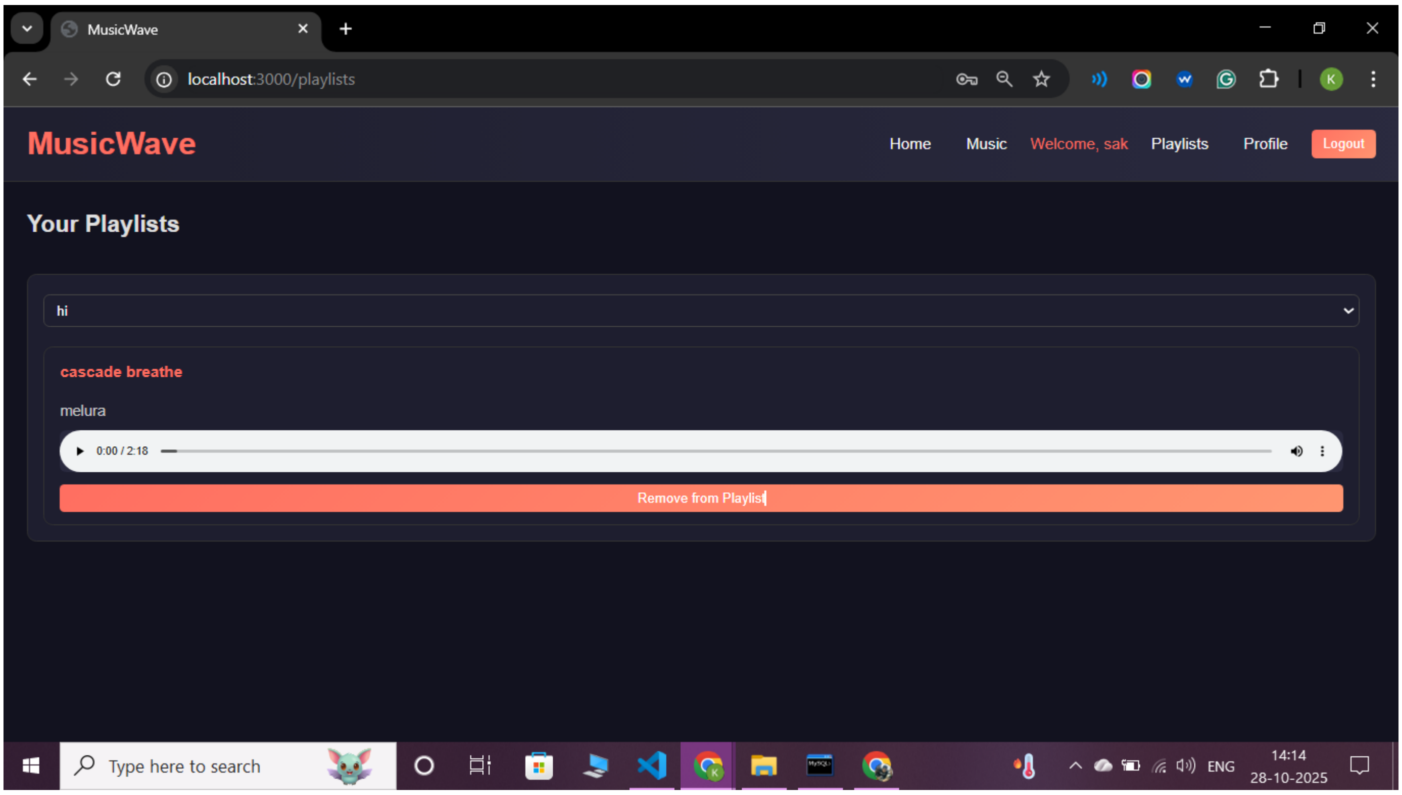
Existing Music

ID	Title	Artist	Action
2	cascade-breathe	karan	Delete
3	aaj phir	arijit singh	Delete
10	hanuman	asay	Delete
11	teri-ye-adaa	arijit	Delete

Registered Users

ID	Username	Email	Action
2	karan	km@gmail.com	Delete
4	karan	krn@gmail.com	Delete
5	sak	s@gmail.com	Delete
6	db	d@gmail.com	Delete

8. Playlist Page



Your Playlists

hi

cascade breathe

melura

0:00 / 2:18

Remove from Playlist

4.Implementation Approach

1. Tech Stack

The MusicWave project is developed using a modern full-stack web development approach.

- Frontend: React.js, HTML5, CSS3, JavaScript (ES6), Axios, React Router
- Backend: Node.js with Express.js for RESTful API development
- Database: MySQL for structured data storage and retrieval
- Other Tools: Vite (for frontend build), Multer (for file uploads), Bcrypt.js (for password hashing), and Dotenv (for environment configuration)

This combination provides high scalability, responsiveness, and security while maintaining a smooth user experience.

2. Backend Implementation

The backend of MusicWave is built using Node.js and Express.js to create a robust REST API for managing users, music, and administrative functions.

Key Features:

- Handles user registration, authentication, and profile management.
- Implements role-based access control, separating user and admin functionalities.
- Use Multer for uploading and managing music files.
- Protects passwords through Bcrypt.js hashing before storing them in the database.
- Provides well-defined API endpoints for all frontend operations.
- Utilizes dotenv for environment variable management.

Workflow:

1. The server connects to the MySQL database through mysql2/promise.

2. Incoming requests are routed through Express routes (userRoutes.js, adminRoutes.js).

3. Controllers handle business logic, interact with models, and return JSON responses.

4. The backend serves static music files and handles validation and error responses.

3. Frontend Implementation

The frontend is developed using React.js for dynamic rendering and single-page navigation. It provides an intuitive and responsive user interface that enhances the overall user experience.

Key Features:

- React Router handles navigation between pages (Home, Login, Signup, Profile, Admin Dashboard).
- Axios is used for communicating with the backend API.
- HTML5 input validation is applied, especially type="email" for correct email format checking.
- The design uses Flexbox, CSS transitions, and a dark-modern theme for visual appeal.
- Reusable components such as Header, MusicPlayer, and MusicList improve maintainability.
- The Admin Dashboard provides controls for uploading, deleting, and managing users and music.

Frontend Flow:

1. User interacts through forms and buttons (e.g., login, signup, upload music).

2. React sends API requests using Axios to the backend.

3. The UI updates dynamically based on backend responses, ensuring smooth interactivity.

4. Database Implementation

The database is implemented using MySQL to store and manage all persistent data, including user credentials, music metadata, and admin details.

Key Features:

Relational structure ensures data integrity through foreign key constraints.

The database contains two main tables:

- Users — stores username, email, password (hashed), and role (user/admin).
- Music — stores title, artist, file path, upload time, and uploader ID.

- Queries are executed using prepared statements to prevent SQL injection.
- Supports operations like insert, select, update, and delete for both users and music.

Workflow:

1. When a new user registers, details are validated, hashed, and stored in the users table.

2. When music is uploaded, the file is saved to the server, and its metadata is inserted into the music table.

3. Admins can view and manage both tables through backend routes.

5.Testing Approach

1. Testing Objective

The objective of testing in the MusicWave project is to ensure that the system functions correctly, securely, and efficiently according to the defined requirements.

The main goals include:

- Verify that all modules (User, Admin, Music Management) perform as expected.
- Ensuring seamless interaction between frontend, backend, and database.
- Detecting and fixing bugs related to UI behavior, API communication, and data handling.
- Validating input forms and security features such as password hashing and restricted access.
- Guaranteeing a smooth and responsive user experience across devices.

2. Types of Testing

a. Unit Testing

- Individual functions and modules are tested in isolation.

Example: Testing password hashing, API endpoints, and form validation logic.

b. Integration Testing

- Ensures that components like frontend forms, backend APIs, and the MySQL database work together correctly.

Example: Verifying that the signup form correctly sends data to the backend and that user details are stored in the database.

c. Functional Testing

- Validates that each function of the application meets the specified requirements.

Example: Checking user login, music upload, playback, and delete functionalities.

d. User Interface (UI) Testing

- Verifies layout, responsiveness, navigation flow, and visual design consistency.

Example: Testing the header, buttons, music cards, and responsive layout in different devices.

e. System Testing

- Complete system is tested as a whole to ensure all components operate in harmony.

Example: Full workflow from signup → login → play music → admin upload → logout.

f. Security Testing

- Focuses on password encryption, restricted access, and validation checks.

Example: Ensuring that only admins can upload or delete music files.

g. Performance Testing

- Verifies that the system performs well under normal user load.

Example: Checking page load speed and audio playback performance.

h. Validation Testing

- Ensures correctness of user inputs, especially email, password, and file uploads.

Example: HTML5 type="email" and required field validations.

3. Test Cases

Test Case ID	Module	Test Description	Expected Result	Actual Result	Status
TC01	Signup	Enter valid username, email, and password	User account created and stored in DB	Pass	✓
TC02	Signup	Enter invalid email (no “@”)	Validation error, form not submitted	Pass	✓
TC03	Login	Enter correct credentials	User logged in, redirected to dashboard	Pass	✓
TC04	Login	Enter incorrect password	Error message displayed	Pass	✓
TC05	Admin Login	Enter admin email and password	Redirect to admin dashboard	Pass	✓
TC06	Music Upload	Upload valid .mp3 file	File stored and displayed in music list	Pass	✓
TC07	Music Upload	Upload invalid file format	Upload blocked, error message shown	Pass	✓
TC08	Music Play	Click on play icon	Music plays; other tracks stop	Pass	✓
TC09	Music Search	Enter song name	Filtered list displayed	Pass	✓
TC10	Delete Music	Click delete (admin only)	File removed from server and DB	Pass	✓
TC11	User Profile	View and update details	Profile updated successfully	Pass	✓
TC12	Change Password	Enter old and new password	Password updated securely (bcrypt)	Pass	✓
TC13	Email Validation	Enter incorrect format	Browser blocks submission (HTML5 validation)	Pass	✓

4. Testing Tools

- **Postman:** Used for testing REST API endpoints (/signup, /login, /upload, /delete).
- **React Developer Tools:** For inspecting component state and props in the browser.
- **Browser DevTools:** For UI responsiveness, console logs, and network request monitoring.
- **MySQL Workbench:** For verifying database operations, table updates, and stored data.
- **VS Code Debugger:** For backend debugging and route testing.
- **Jest (Optional):** For unit testing individual backend functions (if configured).

5. Results

After performing multiple rounds of testing:

- All modules (User, Admin, Music Upload, Playback) worked as expected.
- HTML5 validation successfully prevented invalid email entries.
- Passwords were securely stored using bcrypt hashing.
- No unauthorized access was possible to admin routes without valid admin credentials.
- Audio files uploaded correctly and were playable through the frontend interface.
- The application maintained fast response times with minimal errors during testing.

6.Results And Discussions

Results

Functional Results

User Authentication:

Both users and admins can securely log in using bcrypt-hashed passwords. HTML5 form validation ensures only valid emails are accepted during signup and login.

User Module:

Users can sign up, log in, browse music, search for tracks by title or artist, play music, and change their passwords through an intuitive interface.

Admin Module:

Admins can log in securely, view all registered users, delete users if necessary, upload new songs, and manage existing music files from a single dashboard.

Music Management:

Songs uploaded by admins are automatically stored in the server directory (/uploads/music) and registered in the MySQL database. Users can play or download music without delay.

Profile Management:

Users can view their profiles and change their passwords after verifying their old one, ensuring privacy and account safety.

Search Functionality:

The search feature filters music dynamically using SQL LIKE queries for efficient and fast results.

Validation & Security:

HTML5 type=”email” ensures proper email format, while bcrypt ensures password encryption.

Admin routes are protected through backend role-based authentication (adminAuth middleware).

Performance Results

a. The system performed efficiently under moderate user load.
b. API requests for fetching and searching music returned responses quickly (<300ms in local environment).
c. Audio playback was smooth with minimal buffering due to optimized file handling and static serving from Express.
d. Database queries executed efficiently through connection pooling in db.js using mysql2/promise.

User Interface Results

- The frontend, designed using React.js, provided a modern and responsive UI.
- Dark mode styling and clean layout (as defined in styles.css) enhanced usability and user
- engagement.
- Navigation between components (e.g., Dashboard, Profile, Upload Page) was fast and smooth due to React Router integration.

Database Results

- The MySQL database successfully handled CRUD operations for users and music records.
- Referential integrity between the users and the music table was maintained through uploadedBy relationships.
- Data retrieval and deletion (via Admin Dashboard) reflected real-time updates in the system.

Discussion

- The MusicWave project demonstrates the successful integration of a multi-tier web application using React (Frontend), Node.js with Express (Backend), and MySQL (Database).
- The modular architecture made it easier to implement and test each feature independently before integrating them into the main system.
- The Admin Module enhanced system maintainability by allowing administrators to manage users and music directly.
- The Email Validation using HTML5 improved form reliability, ensuring correct data input and reducing server-side validation errors.
- Implementing bcrypt password hashing significantly strengthened system security and protected sensitive user data.
- File uploads handled by Multer were stored securely, and paths were dynamically updated in the database for consistent data management.
- Overall responsiveness was achieved through React state management and optimized API calls.

Challenges included managing file paths correctly across frontend and backend during deployment and maintaining consistent database relations between users and uploaded music. However, these were resolved through structured route design and parameterized SQL queries.

7.Conclusion and Future Scopes

Conclusion

The MusicWave project successfully provides a seamless platform for users to discover, stream, and manage music online while giving administrators tools to manage users and content efficiently.

It integrates a modern UI, secure authentication, and a well-structured backend using Node.js, Express, and MySQL. The frontend, built with React, ensures a smooth, responsive, and user-friendly experience.

Through its modular design and proper use of RESTful APIs, MusicWave achieves scalability, maintainability, and efficient data management.

Overall, the project demonstrates how modern web technologies can be combined to build an engaging and functional music streaming web application.

Future Scope

- **Mobile App Integration –** Extend the platform to Android and iOS for better accessibility.
- **AI-Based Song Recommendations –** Use machine learning to suggest personalized playlists based on user preferences and listening history.
- **Offline Music Support –** Enable users to download and play songs offline.
- **Live Streaming and Artist Interaction –** Allow artists to host live music sessions and interact with fans.
- **Social Sharing –** Add features to share playlists and songs on social media.
- **Advanced Analytics for Admins –** Provide insights into user activity, popular tracks, and system performance.
- **Cloud-Based Storage –** Integrate cloud platforms like AWS or Firebase for scalable song and data storage.
- **Subscription and Monetization –** Add premium plans with ad-free listening, exclusive tracks, and higher-quality streaming.

8.References

1. React Documentation – Official React.js documentation. Available: https://reactjs.org/docs/getting-started.html

2. Node.js Documentation – Official Node.js documentation. Available: https://nodejs.org/en/docs/

3. Express.js Guide – Official Express framework guide. Available: https://expressjs.com/en/starter/installing.html

4. MySQL Reference Manual – Official MySQL documentation. Available: .https://dev.mysql.com/doc/

5. Axios Documentation – HTTP client for making API requests. Available: https://axios-http.com/docs/intro

6. Vite.js Documentation – Build tool for frontend development. Available: https://vitejs.dev/guide/

7. W3Schools, "HTML Forms and Input Validation." Available: https://www.w3schools.com/html/html_forms.asp

8. ChatGPT:https://chatgpt.com/