# Music Library Management System

A COURSE PROJECT REPORT
By

**Atandrit Chatterjee** (RA2011031010042)
**Naman Saxena** (RA2011031010043)
**Arsh Bhatia** (RA2011031010053)

Under the guidance of

**Dr. S. Thenmalar**

In partial fulfilment for the Course

Of

18CSC303J – DATABASE MANAGEMENT SYSTEM


Faculty of Engineering and Technology, SRM Institute of Science and Technology
Kattankulathur, Tamil Nadu – 603203

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

**(Under Section 3 of UGC Act, 1956)**

# BONAFIDE CERTIFICATE

Certified that this mini project report "**Music Library Management System**" is the bonafide work of **Atandrit Chatterjee** (RA2011031010042)**, Naman Saxena** (RA2011031010043) and **Arsh Bhatia** (RA2011031010053) who carried out the project work under my supervision.

**SIGNATURE**

**Dr. S. Thenmalar**
**Networking and Communications**
SRM Institute of Science and Technology

# ABSTRACT


- A Music Library Management System is a software solution for managing and organizing a music library for music lovers.
- The Music Library Management System provides a centralized platform for managing a music library.
- The system also offers a range of functionalities such as adding new albums, editing existing ones, deleting albums, searching for specific songs, and generating playlists.
- The system helps users maintain and update their music library effortlessly.
- It offers a convenient and organized way to manage a music library.

# INTRODUCTION

- The Music Library Management System is a software solution for music lovers to organize and manage their music collection.
- The system provides a centralized platform to access the entire music library from a single location.
- The system's functionalities include adding new albums, editing existing ones, deleting albums, searching for specific songs, and generating playlists.
- The system is essential for individuals who want to maintain and update their music library efficiently.
- Benefits of the Music Library Management System include easy management of large music libraries, a user-friendly interface, and the ability to search for specific songs, create custom playlists, and rate favourite albums and songs.
- Users can add new albums to the music library from digital sources, and the system automatically organizes them.
- The system allows users to update album information, add album artwork, and delete unwanted albums.
- Searching for specific songs is easy with the system's search functionalities.
- The playlist generation feature allows users to create custom playlists and provides personalized recommendations based on their listening history.
- The Music Library Management System is a must-have tool for anyone who loves music.

# PROBLEM STATEMENT

- Music enthusiasts who manage their music library manually can face challenges in keeping track of their collection, especially if they have many albums and songs. This task can be time-consuming and burdensome.

- Maintaining organization and updating information for many musical pieces can be overwhelming and challenging for users.

- Manual management can result in various problems, including duplicate entries, missing information, and an unorganized library. These issues can make it difficult for users to efficiently manage their music collection.

- The lack of organization and accuracy can lead to inefficiencies, making it challenging for users to find the songs they want to listen to or create custom playlists.

- Manual management can also result in inaccuracies in album and song information, causing confusion and making it challenging for users to keep their collection up-to-date.

- Without a centralized platform for managing and updating the collection, it can be challenging to keep up with new album releases and additions to the music library.

# OBJECTIVE

The Music Library Management System has several objectives that make it an essential tool for music lovers. These objectives include:

- The Music Library Management System aims to provide a centralized platform for managing a music library, which enables users to access their entire music collection from a single location.
- The system offers various functionalities, including adding new albums, editing existing ones, deleting albums, searching for specific songs, and generating playlists, to help users maintain and update their music library efficiently.
- One of the system's primary objectives is to handle large music libraries with ease, as managing extensive music collections can be challenging.
- The Music Library Management System's user-friendly interface is designed to be intuitive and easy to use, allowing users to navigate through their music library effortlessly.
- The system enables users to import new albums from a digital source, such as a CD or an online music store, and automatically organizes them to make them easy to find and access.
- Users can edit the information associated with these albums, such as artist, album title, and genre, and rate their favourite songs and albums, providing a convenient way to keep track of their preferred music.
- The system's rating feature enables it to make personalized recommendations based on the user's listening history.

In summary, the Music Library Management System's objectives include providing a centralized platform for managing a music library, handling large music libraries with ease, having a user-friendly interface, allowing users to import and edit albums, removing albums, searching for specific songs, creating custom playlists, and rating favourite songs and albums.

# SCOPE AND APPLICATIONS

- The Music Library Management System has a broad scope and can be used by anyone with a music collection, including music enthusiasts, professionals, and music organizations.

- The system is scalable and can handle music collections of any size, making it suitable for both personal and commercial use.

- The project has various applications in different fields, including the music industry, music venues and event organizers, music schools and conservatories, libraries, museums, and archives, and music streaming services.

- In the music industry, the system can be used by music producers, DJs, and music labels to manage their music catalog and organize their tracks. Music venues and event organizers can also use it to create playlists for events and ensure the right music is played at the right time.

- In educational settings, music teachers can use the system to create playlists for their students based on their level and musical interests. Libraries, museums, and archives can also use it to organize and manage their music libraries.

- Streaming services can use the system to offer personalized recommendations to their users and improve their user experience. The system's ability to rate and review songs can also be used to generate user-generated content and provide feedback on new releases.

- In personal settings, music enthusiasts can use the system to manage their music collections, create playlists, and discover new music. The system's ease of use and ability to handle large music libraries make it a valuable tool for anyone who loves music.

# LIST OF GENERAL AND UNIQUE SERVICES IN THE DATABASE APPLICATION

- User authentication and authorization to control access to the library and playlists.
- Integration with music recognition software to automatically populate metadata for new albums and songs.
- Data analytics and visualization tools to track listening habits and generate recommendations.
- Social features such as the ability to share playlists or collaborate on a playlist with friends.
- Automatic backup and synchronization of the library across multiple devices.
- Integration with third-party APIs to pull in additional data about artists, albums, and songs.
- Advanced search capabilities, including searching by lyrics or mood.

# SOFTWARE REQUIREMENT SPECIFICATION

## *Functional Requirements*

The functional requirements for the Music Library Management System:

- User registration and login functionality to allow users to access and manage their music library and playlists.
- Advanced search functionality, including the ability to search by artist, album, genre, year, lyrics, or mood.
- Customizable playlist creation, including the ability to add, remove, and reorder songs in a playlist.
- User authentication and authorization to control access to the music library and playlists.
- Integration with third-party APIs to pull in additional data about artists, albums, and songs.
- Automatic backup and synchronization of the music library across multiple devices.

These functional requirements are essential for providing users with a comprehensive and user-friendly solution for managing a music library. By allowing users to easily search, create and manage playlists, and backup their music library across multiple devices, the Music Library Management System provides a convenient and organized way to manage a music collection. Additionally, user authentication and authorization help to ensure that users have complete control over who has access to their music library and playlists. Integration with third-party APIs also provides users with additional information and data about their favorite artists and songs.

## *Non Functional Requirements*

The non-functional requirements for the Music Library Management System:

- Usability: The system should be easy to use and navigate, with an intuitive user interface that enables users to quickly perform tasks.
- Performance: The system should be fast and responsive, providing users with quick access to their music library and playlists.
- Security: The system should be secure, protecting user data from unauthorized access or breaches.

- Compatibility: The system should be compatible with various operating systems and devices, enabling users to access their music library on a range of devices.
- Maintainability: The system should be easy to maintain and update, minimizing downtime and disruption to users.
- Accessibility: The system should be accessible to users with disabilities, including support for assistive technologies and compliance with accessibility standards.

By focusing on these non-functional requirements, the Music Library Management System can provide users with a high-quality and reliable experience that is easy to use and accessible to a wide range of users.

### *Hardware and Software Requirements*

The Music library Management System requires a computer system with an internet connection. The minimum hardware and software requirements are as follows:

● Operating System: Windows, Linux or MacOS
● Web Server: Apache or IIS
● Database: MySQL or PostgreSQL
● Programming Languages: HTML, CSS, JavaScript, PHP
● Browser: Google Chrome, Mozilla Firefox, or Microsoft Edge

### *System Constraints*

● The system must comply with all applicable laws and regulations related to data privacy and security.
● The system must be able to handle large amounts of data and user traffic.
● The system should have a backup and disaster recovery plan in place to ensure continuity of service in case of system failure.

# LITERATURE SURVEY

There are several existing music library management systems available in the market today, each with its own unique features and capabilities. A literature survey of these systems reveals that there are several key factors to consider when choosing a music library management system, including ease of use, compatibility with various devices, and support for advanced features such as social sharing and analytics.

- **iTunes:** Developed by Apple Inc., is a popular music library management system that provides users with a centralized platform for managing their music library, as well as a built-in media player for listening to music.
- **Windows Media Player:** Developed by Microsoft, is another popular music library management system that provides users with similar functionalities to iTunes, but is only available on Microsoft devices.
- **Spotify:** Developed by Spotify AB, is a cloud-based system that provides users with a centralized platform for managing their music library, as well as a built-in media player for listening to music. In addition to advanced search capabilities, it also offers social sharing features that allow users to share their playlists with friends and collaborate on playlists together.

In conclusion, there are several existing music library management systems available in the market today, each with its own unique features and capabilities. By conducting a literature survey of these systems, users can gain a better understanding of the key factors to consider when choosing a music library management system, including ease of use, compatibility with various devices, and support for advanced features such as social sharing and analytics.

# COMPARISON OF EXISTING VS PROPOSED SYSTEM

The existing music library management systems have certain limitations that the proposed music library management system aims to address. The proposed system is designed to offer a more comprehensive and user-friendly solution for managing a music library.

- One of the limitations of the existing music library management systems is the lack of advanced features such as automatic metadata population, data analytics, and social features. The proposed system aims to incorporate these features to offer a more efficient solution for music lovers. With the user authentication and authorization feature, users can control access to the library and playlists, ensuring security and privacy.

- Another limitation of the existing music library management systems is limited search capabilities and the lack of integration with third-party APIs. The proposed system offers advanced search capabilities, including searching by lyrics or mood. Additionally, it has integration with third-party APIs to pull in additional data about artists, albums, and songs. This feature is particularly useful for music enthusiasts who want to explore new music and discover new artists.

- In terms of user experience, the proposed system offers a more streamlined and organized interface for managing a music library. The system is capable of handling large music libraries with ease, and users can easily add new albums, edit existing ones, delete albums, search for specific songs, and generate playlists. Additionally, the system allows users to rate their favorite songs and albums and provides recommendations based on their listening history.

- Furthermore, the proposed music library management system has a more extensive scope, with features such as social features for sharing and collaborating on playlists, automatic backup, and synchronization across devices. These features make it easy for users to maintain and update their music collection effortlessly.

In conclusion, the proposed music library management system offers a more comprehensive and user-friendly solution for managing a music library than existing systems. Its advanced features, scope, and user experience make it a valuable tool for music enthusiasts who want to maintain and update their music collection effortlessly. With the proposed system, music lovers can enjoy their music collection to the fullest potential.

# SYSTEM ARCHITECTURE AND DESIGN

The architecture and design of a Music Library Management System can vary based on the specific requirements and functionalities of the system. However, here is a general overview of the system architecture and design:

- **Front-end:** The front-end of the system is the user interface that allows users to interact with the system. It can be a web-based interface or a mobile application. The front-end should be designed to be user-friendly, responsive, and accessible on different devices.

- **Back-end:** The back-end of the system is responsible for processing user requests and managing the system's data. It consists of servers, databases, and APIs. The back-end should be scalable, secure, and reliable.

- **Database:** The database is where all the music data related to the library is stored. It should be designed in a way that allows efficient data retrieval and manipulation. The database should also be scalable, secure, and reliable.

- **Functionality:** The Music Library Management System should have various functionalities to support different aspects of music library management, including adding new albums, editing existing ones, deleting albums, searching for specific songs, and generating playlists. The system should also support rating songs and albums, recommending music based on user preferences, and customizing playlists based on different criteria.

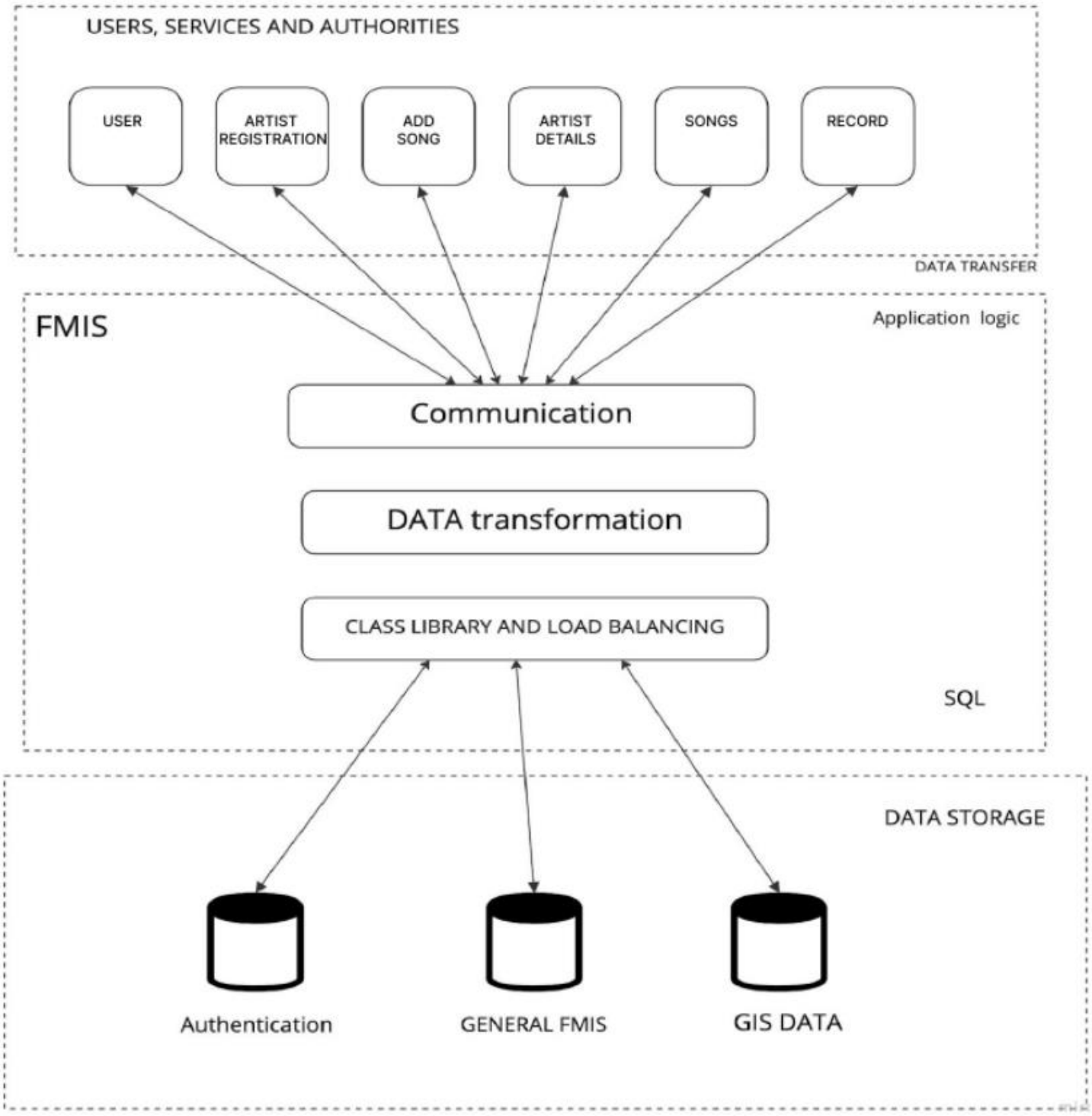
- **Integrations:** The system should be able to integrate with third-party APIs to pull in additional data about artists, albums, and songs. Integration can help automate some of the data input processes and provide more accurate information.

- **Security:** The system should have robust security measures to protect the data and prevent unauthorized access. This can include encryption, user authentication, access controls, and regular security audits.

- **Deployment:** The system should be designed to be deployed on different environments, including cloud-based or on-premise. The deployment process should be automated to reduce errors and ensure consistency.

Overall, the architecture and design of the Music Library Management System should prioritize ease of use, scalability, and security to provide a centralized platform for managing a music library efficiently and effortlessly.

# ARCHITECTURE DIAGRAM

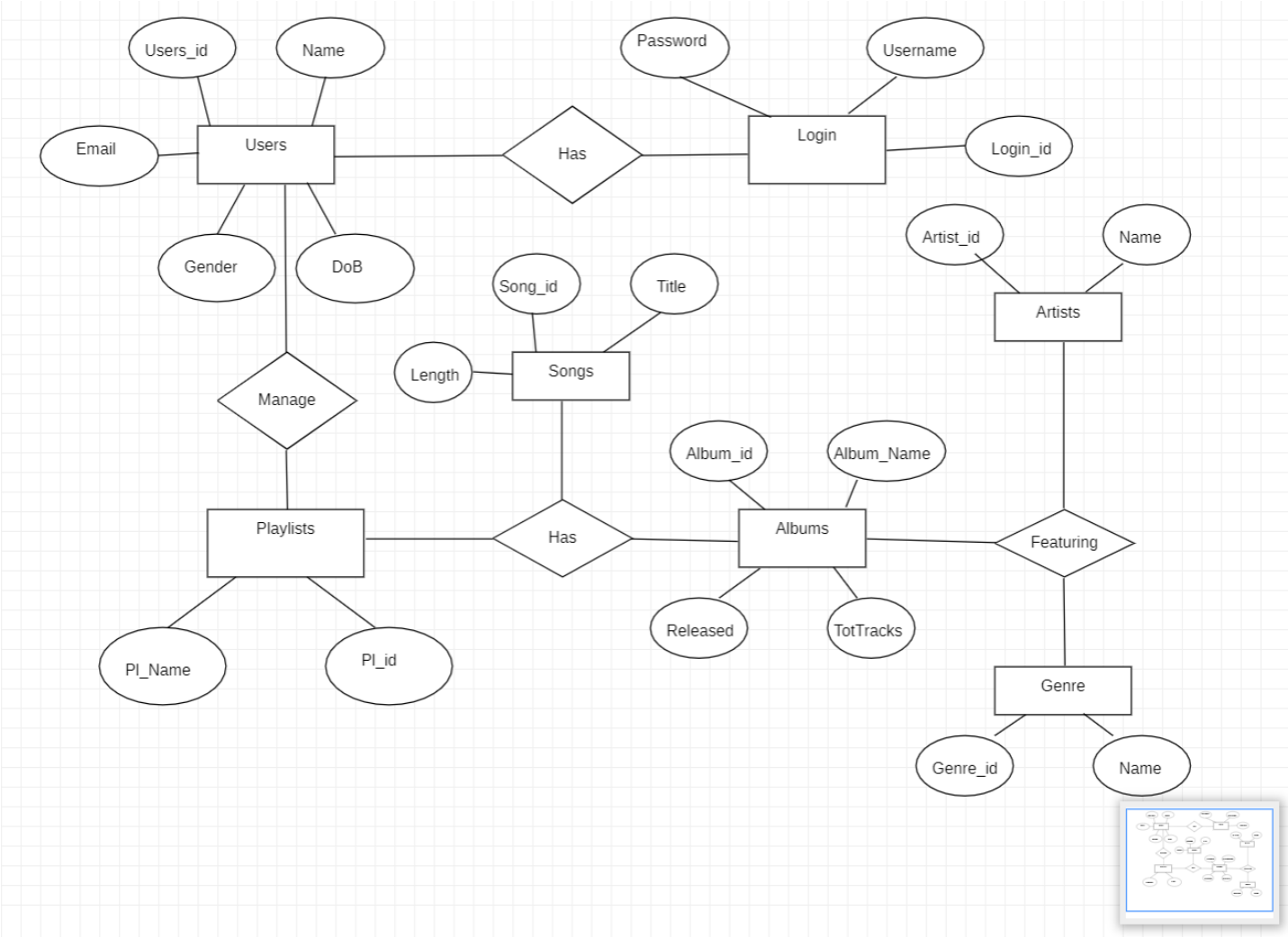


An architecture diagram for a music library management system is a visual representation of the components, modules, and relationships within the system and how they interact to achieve the desired functionality. It provides an overview of the system's architecture and communicates the design to stakeholders.

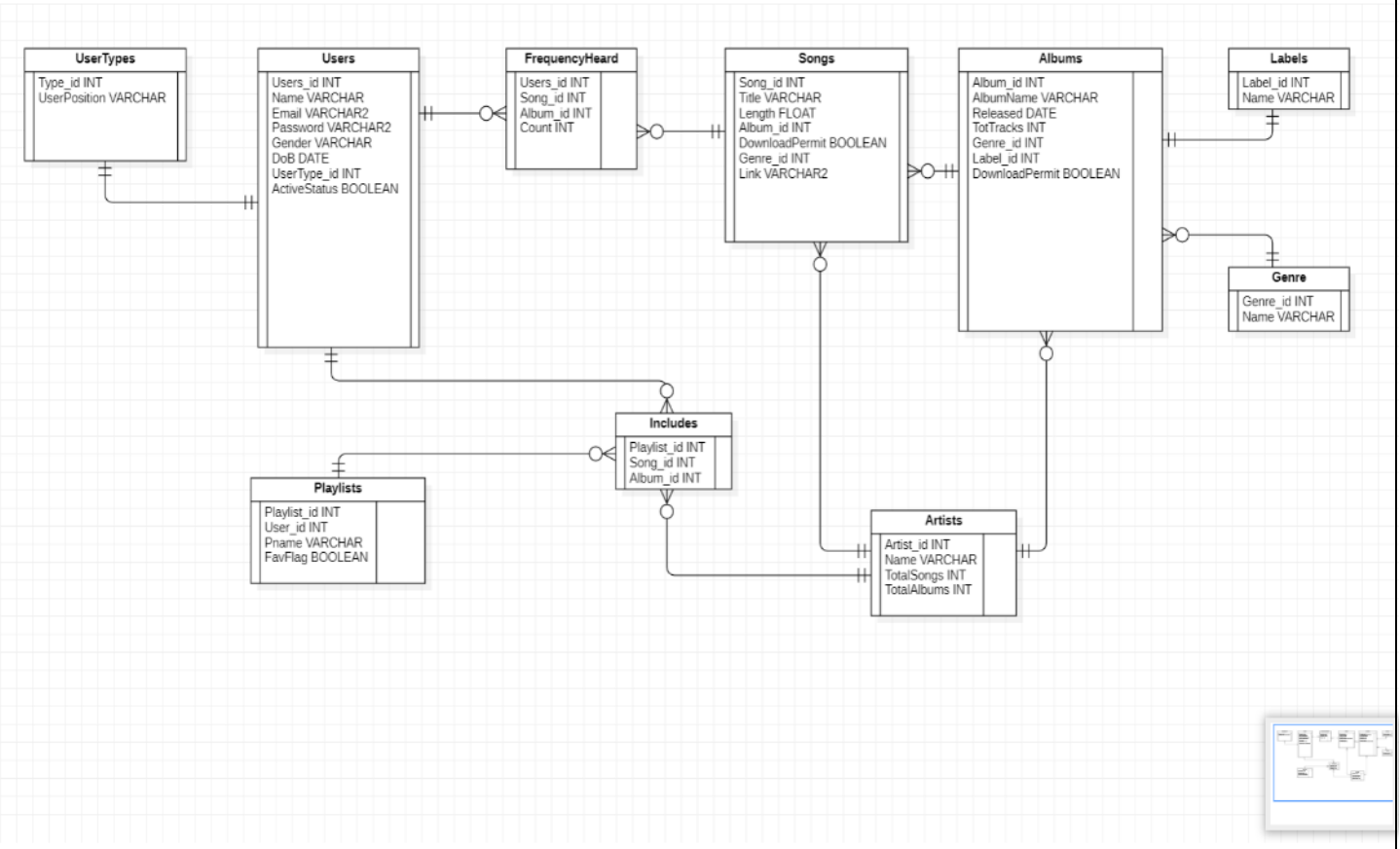
- The overall diagram is divided into three sections: user services and authorities, the music database, and data storage units.
- The first section encompasses all the user services and authorities that the management system provides, such as artist details and registration, songs, and adding songs.
- The second section is dedicated to the music database, which contains information on songs, artists, albums, and genres.
- The third section includes the data storage units, which may include authentication for user login and data backup.
- All the components of the system are connected via the internet to the application logic of the database system.
- The communication section of the system facilitates the connection between the components, where data is transformed and sent to the class library and load balancing.
- All these components are connected with SQL and perform tasks, such as retrieving or updating information in the music database.
- Lastly, all these components are connected to a data storage unit, where the information is stored securely.

Overall, the architecture diagram for a music library management system project would need to be tailored to the specific needs and requirements of the music library and its users. It should also take into account scalability, reliability, and security considerations, as well as any regulatory requirements that may apply.

# ER DIAGRAM - 1


Users_id
Name
Password
Username
Email
Users
Has
Login
Login_id
Gender
DoB
Artist_id
Name
Song_id
Title
Artists
Length
Songs
Manage
Album_id
Album_Name
Playlists
Has
Albums
Featuring
Released
TotTracks
Pl_Name
Pl_id
Genre
Genre_id
Name

# ER DIAGRAM – 2


UserTypes
Type_id INT
UserPosition VARCHAR
Users
Users_id INT
Name VARCHAR
Email VARCHAR2
Password VARCHAR2
Gender VARCHAR
DoB DATE
UserType_id INT
ActiveStatus BOOLEAN
FrequencyHeard
Users_id INT
Song_id INT
Album_id INT
Count INT
Songs
Song_id INT
Title VARCHAR
Length FLOAT
Album_id INT
DownloadPermit BOOLEAN
Genre_id INT
Link VARCHAR2
Albums
Album_id INT
AlbumName VARCHAR
Released DATE
TotTracks INT
Genre_id INT
Label_id INT
DownloadPermit BOOLEAN
Labels
Label_id INT
Name VARCHAR
Genre
Genre_id INT
Name VARCHAR
Includes
Playlist_id INT
Song_id INT
Album_id INT
Playlists
Playlist_id INT
User_id INT
Pname VARCHAR
FavFlag BOOLEAN
Artists
Artist_id INT
Name VARCHAR
TotalSongs INT
TotalAlbums INT

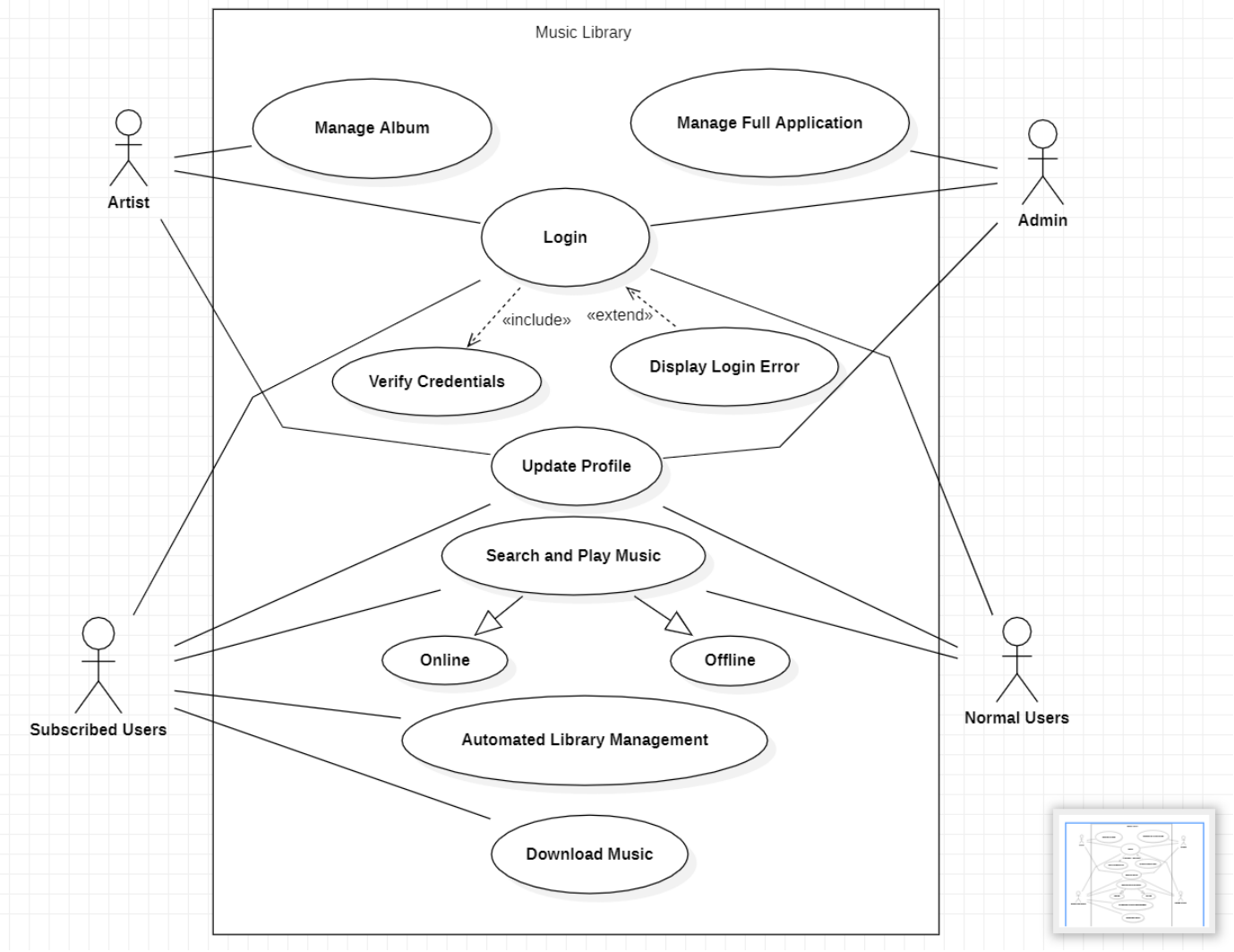
The ER Diagram for the Music Library Management System includes the following entities:

- **Album:** This entity stores information about each album in the music library, such as album title, artist name, release year, genre, and number of tracks.
- **Song:** This entity stores information about each track in the music library, such as song title, artist name, album title, release year, genre, and song length.
- **Playlist:** This entity stores information about each playlist in the music library, such as playlist title, creator, number of tracks, and duration.
- **User:** This entity would store information about each user, such as user ID, username, password, email, and date of registration.

The relationships between these entities:

- An album can have many songs, but each track belongs to only one album (one-to-many).
- A playlist can have many songs, and each track can belong to many playlists (many-to-many).
- A user can create many playlists, but each playlist is created by only one user (one-to-many).

# USE CASE DIAGRAM



The Use Case Diagram for the Music Library Management System includes the following use cases:

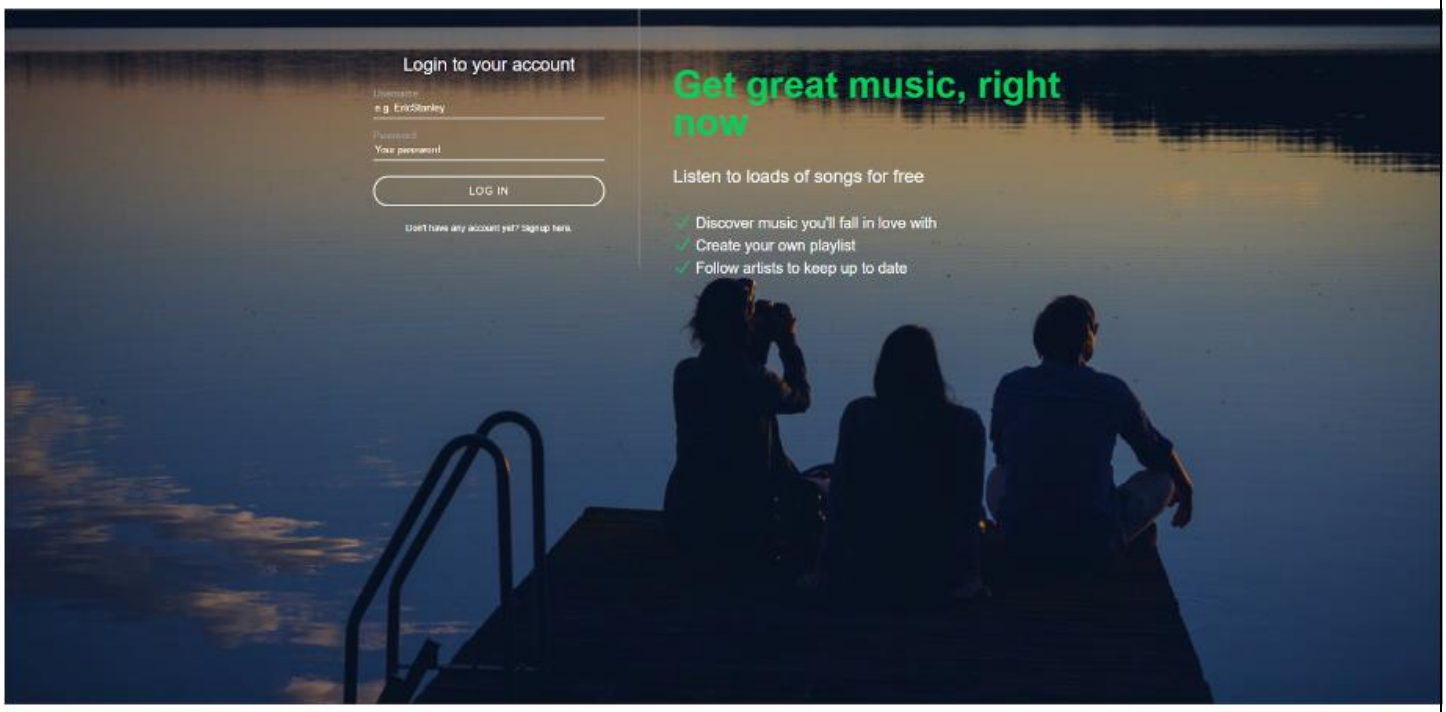
- **Search Song:** This use case allows users to search for a specific song in the music library by title, artist name, or album name.
- **Generate Playlist:** This use case allows users to create a new playlist by selecting songs from the music library.
- **Edit Playlist:** This use case allows users to edit the information of an existing playlist in the music library.
- **Delete Playlist:** This use case allows users to delete a playlist from the music library.
- **Register User:** This use case allows new users to register for an account in the music library system.
- **Login User:** This use case allows registered users to log in to the music library system to access their music collection and playlists.

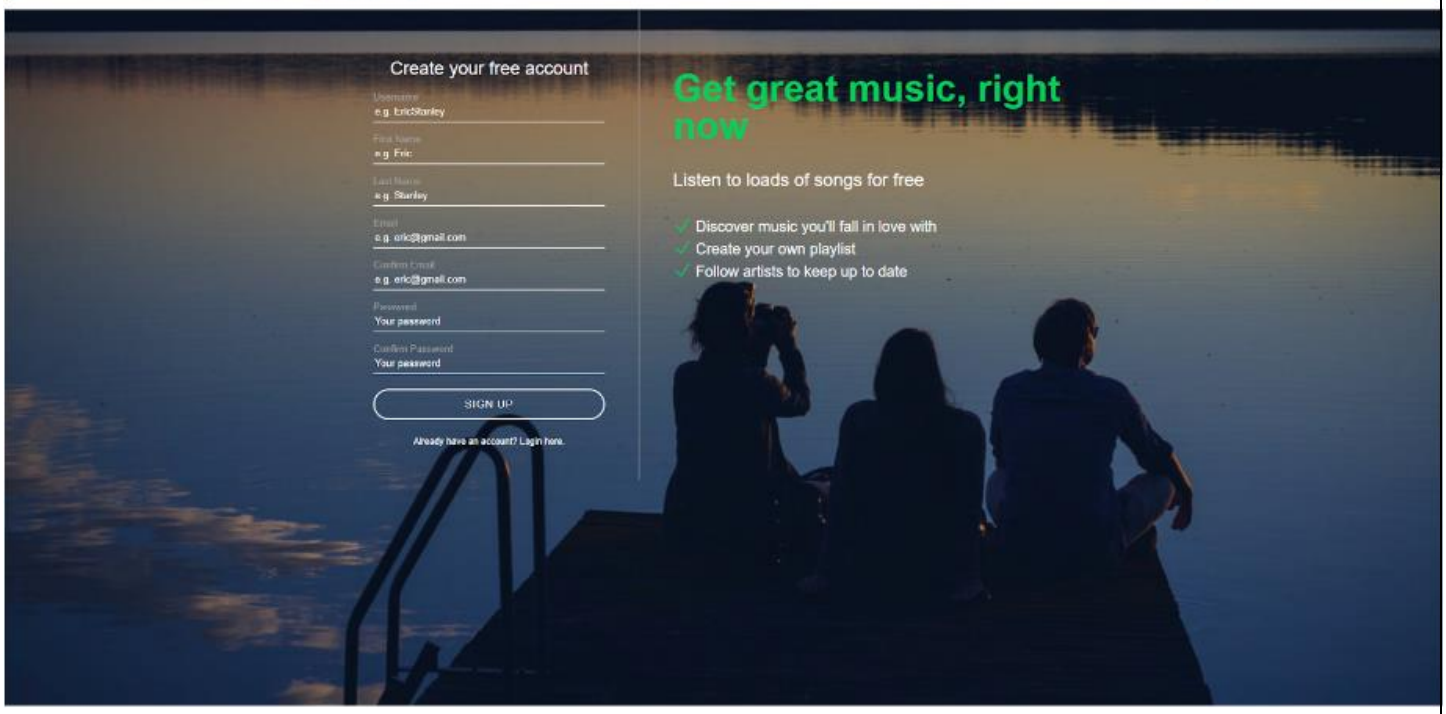
The relationships between these use cases:

- Search Song, Generate Playlist, Edit Playlist, and Delete Playlist are related to Track and Playlist entities.
- Register User and Login User are related to User entity.

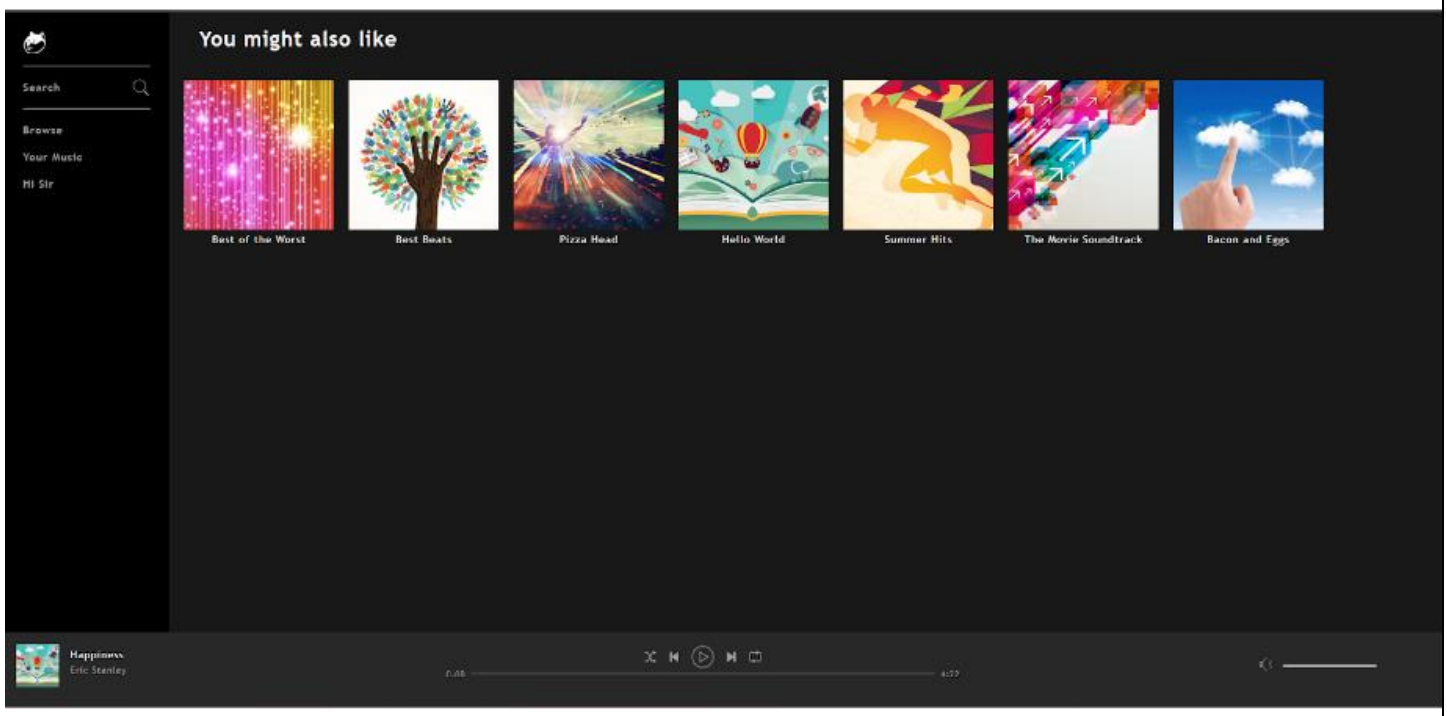
# FRONT END (UI) DESIGN

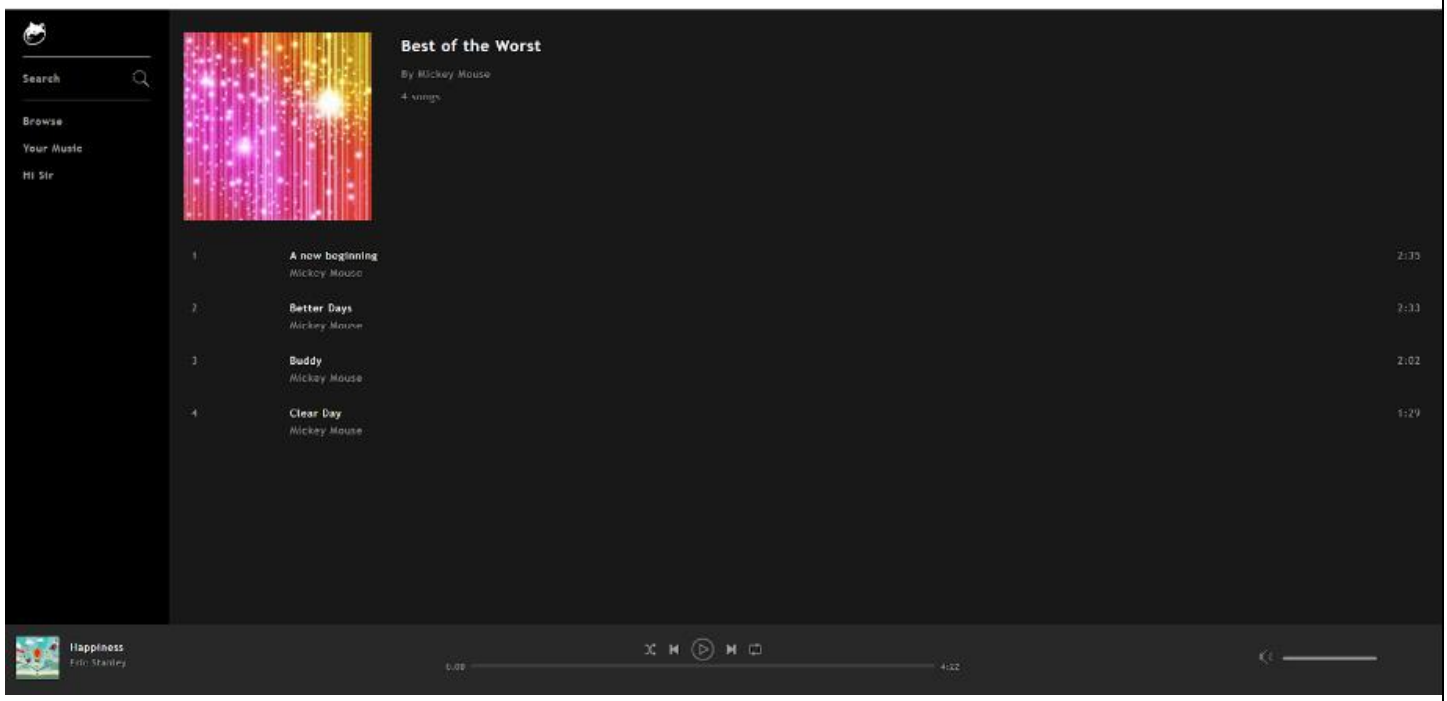
Login Page:

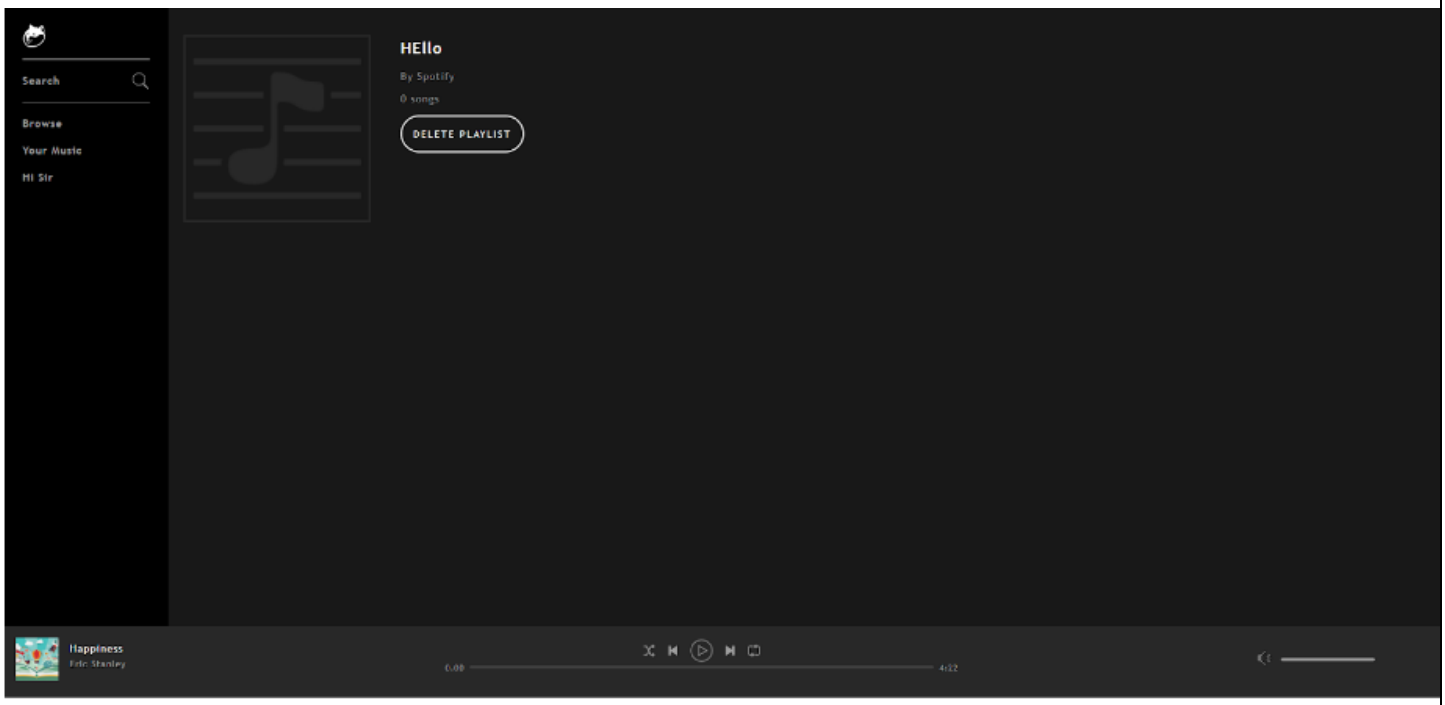


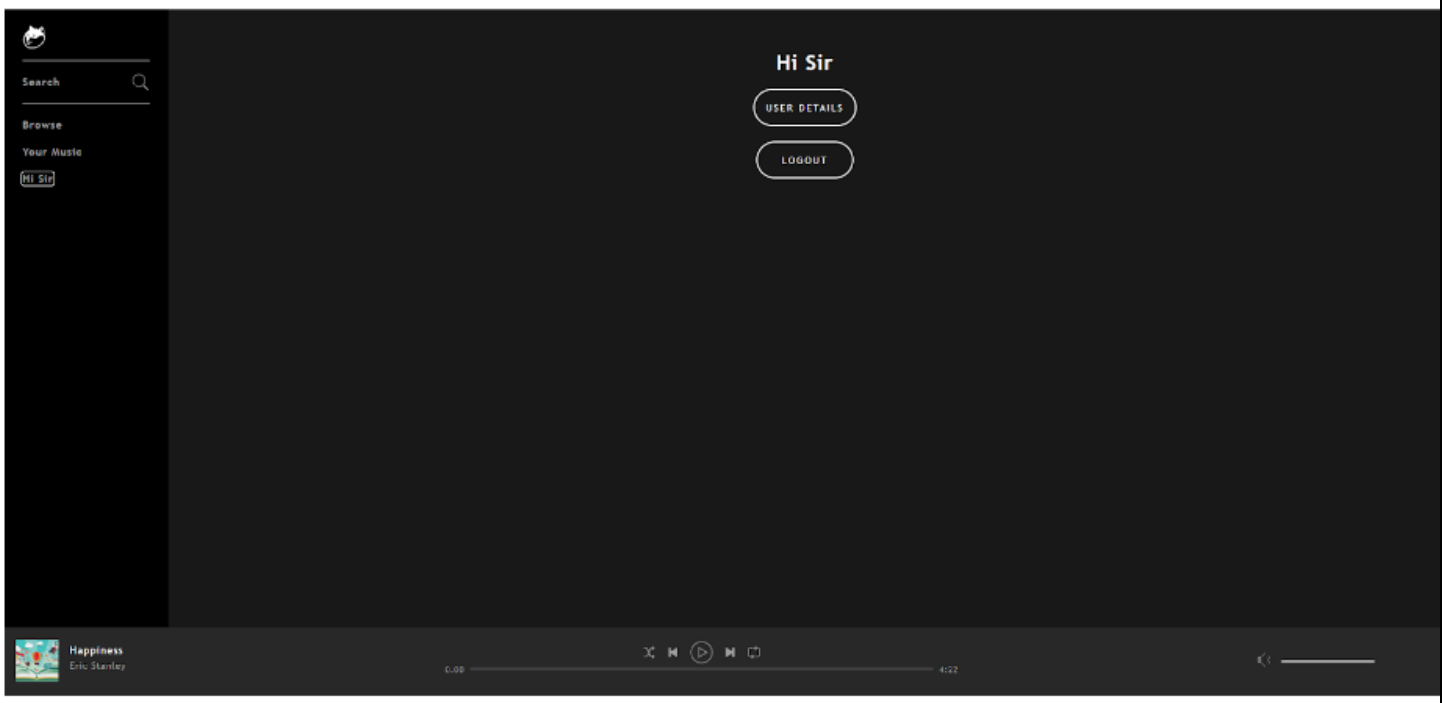



Home Page:


You might also like
Search
Browse
Your Music
Hi Sir
Best of the Worst
Best Beats
Pizza Head
Hello World
Summer Hits
The Movie Soundtrack
Bacon and Eggs
Happiness
Eric Stanley


Search
Browse
Your Music
Hi Sir
Best of the Worst
By Mickey Mouse
4 songs
1 A new beginning Mickey Mouse 2:35
2 Better Days Mickey Mouse 2:33
3 Buddy Mickey Mouse 2:02
4 Clear Day Mickey Mouse 1:29
Happiness
Eric Stanley


Search
Browse
Your Music
Hi Sir
HEllo
By Spotify
0 songs
DELETE PLAYLIST
Happiness
Eric Stanley
0:00
4:22


Search
Browse
Your Music
Hi Sir
Hi Sir
USER DETAILS
LOGOUT
Happiness
Eric Stanley
0:00
4:22

• The frontend of the Music Library Management System was designed using Figma, a cloud-based design and prototyping tool.

• Figma is a popular software used for designing user interfaces and creating interactive prototypes.

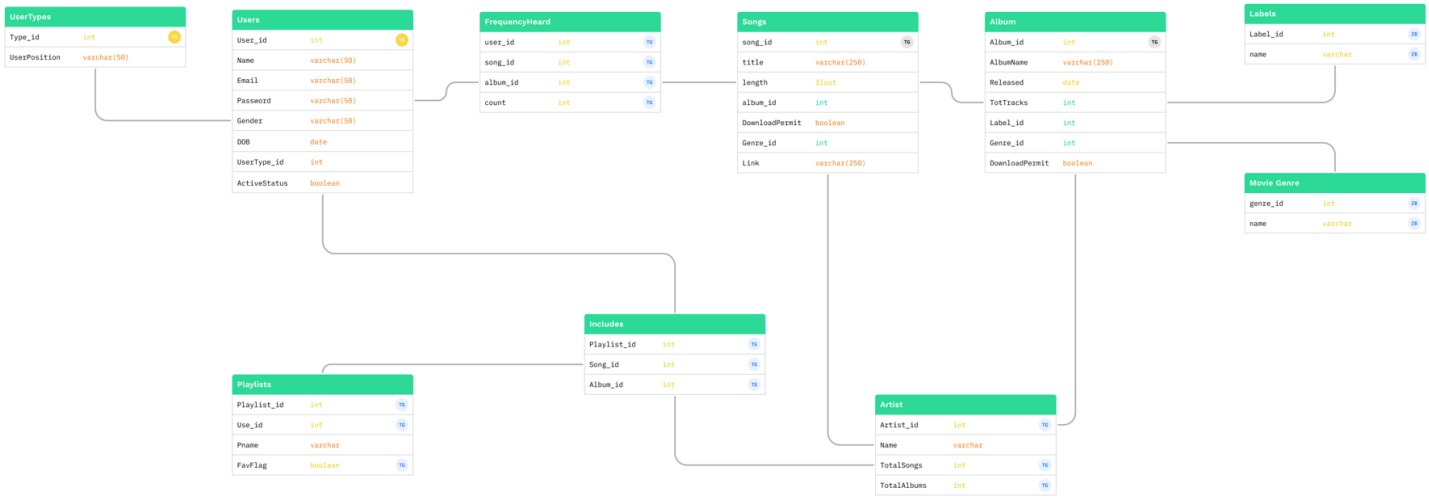
• Figma offers a range of features, such as collaborative design, design systems, and plugins, making it a versatile tool for designing UIs.

• The UI design focuses on providing an intuitive and user-friendly interface for managing music collections.

• The design includes features such as album covers, search bars, and menu options to make navigation easy.

• The UI was designed to be responsive, meaning it adjusts to different screen sizes, including mobile devices.

# BACK END (DATABASE) DESIGN




• The Music Library Management System's backend was designed using FigJam, a collaborative whiteboard software.

• FigJam is a visual collaboration tool used for designing and brainstorming ideas.

• It offers features such as sticky notes, shapes, and connectors, making it an ideal tool for designing database schemas.

• The backend database design includes tables for storing information about albums, songs, artists, and playlists.

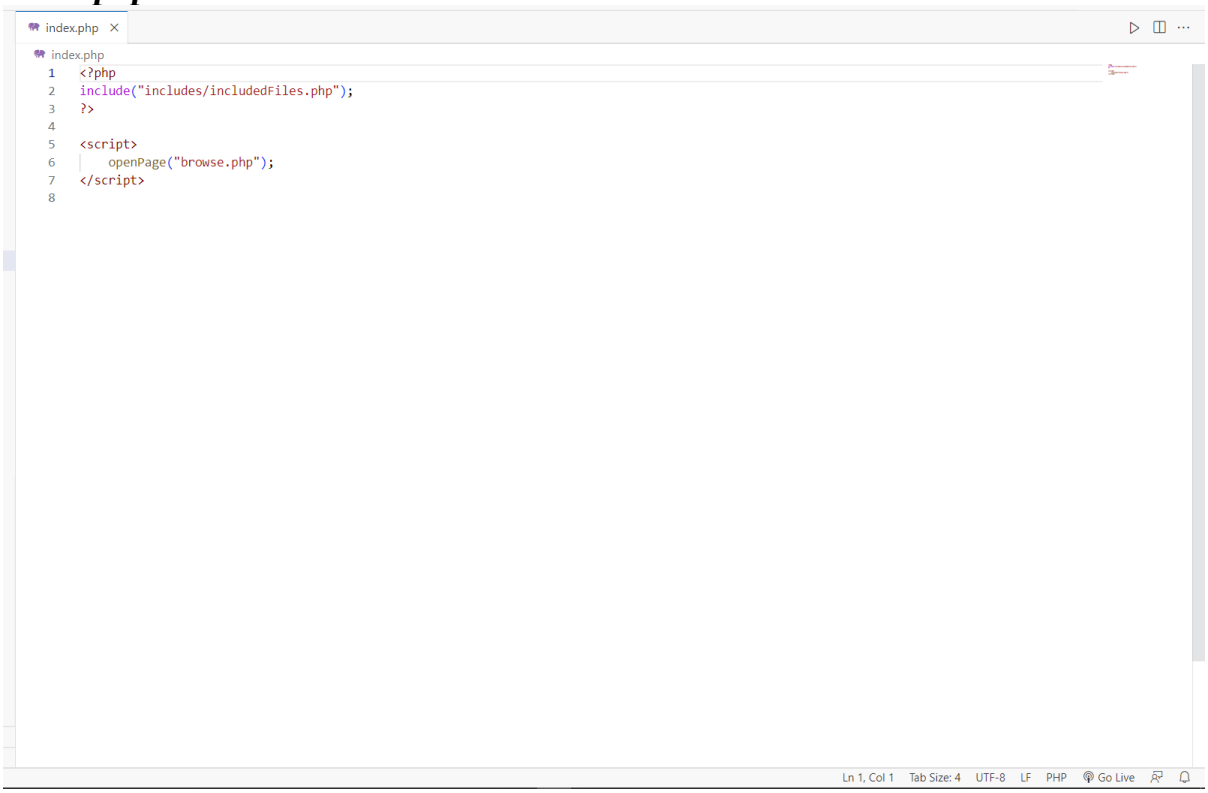
• The design follows the principles of normalization to ensure data consistency and accuracy.

• The database schema was designed to be scalable and flexible, allowing for easy modification in the future.

• The system uses a relational database management system (RDBMS) such as MySQL to store and manage data efficiently.

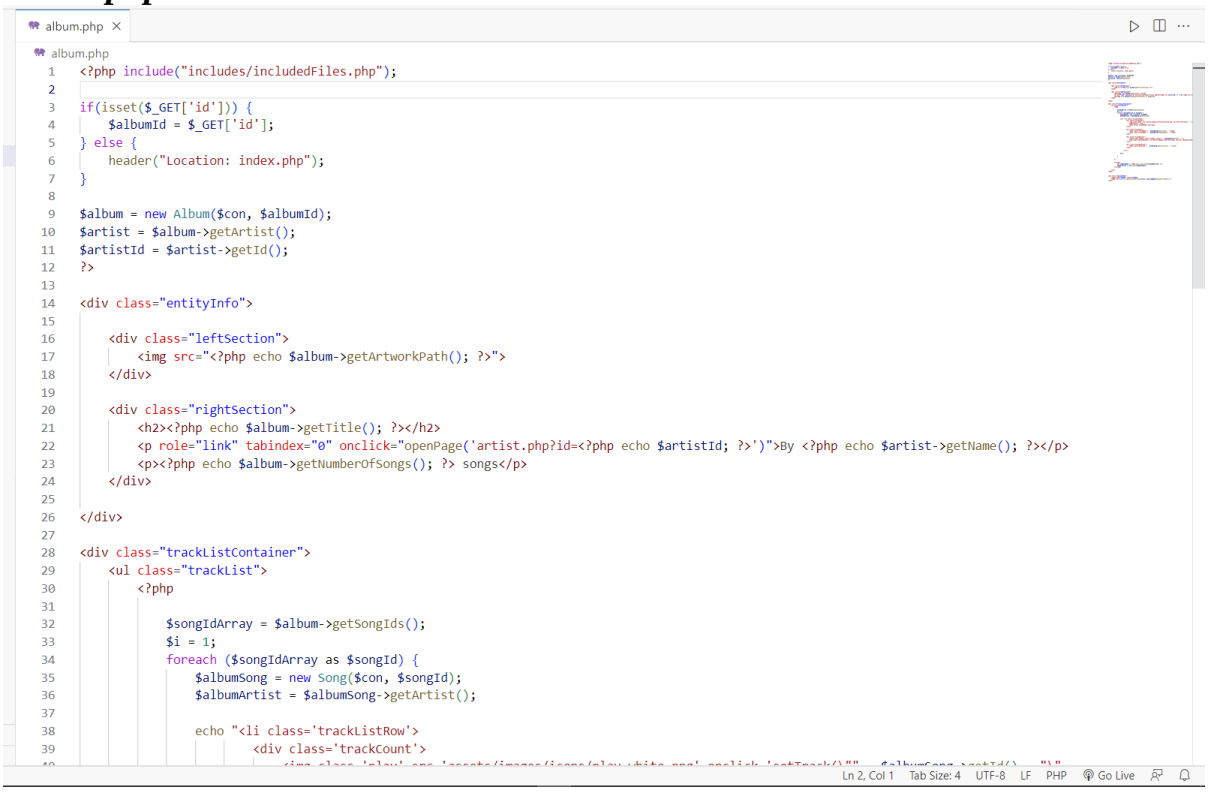
# CODE AND SCREENSHOTS

## *index.php*



```php
<?php
include("includes/includedFiles.php");
?>

<script>
    openPage("browse.php");
</script>
```

## *album.php*



```php
<?php include("includes/includedFiles.php");

if(isset($_GET['id'])) {
    $albumId = $_GET['id'];
} else {
    header("Location: index.php");
}

$album = new Album($con, $albumId);
$artist = $album->getArtist();
$artistId = $artist->getId();
?>

<div class="entityInfo">

    <div class="leftSection">
        <img src="<?php echo $album->getArtworkPath(); ?>">
    </div>

    <div class="rightSection">
        <h2><?php echo $album->getTitle(); ?></h2>
        <p role="link" tabindex="0" onclick="openPage('artist.php?id=<?php echo $artistId; ?>')">By <?php echo $artist->getName(); ?></p>
        <p><?php echo $album->getNumberOfSongs(); ?> songs</p>
    </div>

</div>

<div class="trackListContainer">
    <ul class="trackList">
        <?php

            $songIdArray = $album->getSongIds();
            $i = 1;
            foreach ($songIdArray as $songId) {
                $albumSong = new Song($con, $songId);
                $albumArtist = $albumSong->getArtist();

                echo "<li class='trackListRow'>
                        <div class='trackCount'>
```

album.php

```
                        tempPlaylist, true)'>
                        <span class='trackNumber'>$i</span>
                    </div>

                    <div class='trackInfo'>
                        <span class='trackName'>" . $albumSong->getTitle() . "</span>
                        <span class'artistName'>" . $albumArtist->getName() . "</span>
                    </div>

                    <div class='trackOptions'>
                        <input type='hidden' class='songId' value='" . $albumSong->getId() . "'/>
                        <img class='optionsButton' src='assets/images/icons/more.png' onclick='showOptionsMenu(this)'>
                    </div>

                    <div class='trackDuration'>
                        <span class='duration'>" . $albumSong->getDuration() . "</span>
                    </div>

                </li>";

                $i++;

            }
        ?>

        <script>
            var tempSongIds = '<?php echo json_encode($songIdArray); ?>';
            tempPlaylist = JSON.parse(tempSongIds);
        </script>

    </ul>
</div>


<nav class="optionsMenu">
    <input type="hidden" class="songId">
    <?php echo Playlist::getPlaylistDropdown($con, $userLoggedIn->getUsername()); ?>
</nav>
```

Ln 2, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

## *artist.php*

artist.php

```
<?php

include("includes/includedFiles.php");

if(isset($_GET['id'])) {
    $artistId = $_GET['id'];
} else {
    header("Location: index.php");
}

$artist = new Artist($con, $artistId);

?>


<div class="entityInfo borderBottom">

    <div class="centerSection">

        <div class="artistInfo">

            <h1 class="artistName"><?php echo $artist->getName(); ?></h1>

            <div class="headerButtons">
                <button class="button green" onclick="playFirstSong();">PLAY</button>
            </div>

        </div>

    </div>

</div>


<div class="trackListContainer borderBottom">
    <h2>SONGS</h2>
    <ul class="trackList">
        <?php
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

artist.php

```
            $songIdArray = $artist->getSongIds();
            $i = 1;
            foreach ($songIdArray as $songId) {

                if ($i > 5) {
                    break;
                }

                $albumSong = new Song($con, $songId);
                $albumArtist = $albumSong->getArtist();

                echo "<li class='trackListRow'>
                        <div class='trackCount'>
                            <img class='play' src='assets/images/icons/play-white.png' onclick='setTrack(\"" . $albumSong->getId() . "\",
                            tempPlaylist, true)'>
                            <span class='trackNumber'>$i</span>
                        </div>

                        <div class='trackInfo'>
                            <span class='trackName'>" . $albumSong->getTitle() . "</span>
                            <span class'artistName'>" . $albumArtist->getName() . "</span>
                        </div>

                        <div class='trackOptions'>
                            <input type='hidden' class='songId' value='" . $albumSong->getId() . "'/>
                            <img class='optionsButton' src='assets/images/icons/more.png' onclick='showOptionsMenu(this)'>
                        </div>

                        <div class='trackDuration'>
                            <span class='duration'>" . $albumSong->getDuration() . "</span>
                        </div>

                    </li>";

                $i++;

            }
        ?>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

artist.php

```
        <script>
            var tempSongIds = '<?php echo json_encode($songIdArray); ?>';
            tempPlaylist = JSON.parse(tempSongIds);
        </script>

    </ul>
</div>


<div class="gridViewContainer">

    <h2>ALBUMS</h2>

    <?php
        $albumQuery = mysqli_query($con, "SELECT * FROM albums WHERE artist='$artistId'");

        while($row = mysqli_fetch_array($albumQuery)) {

            echo "<div class='gridViewItem'>
                    <span role='link' tabindex='0' onclick='openPage(\"album.php?id=" . $row['id'] . "\")'>
                        <img src='" .  $row['artworkPath'] . "'>
                        <div class='gridViewInfo'>"
                            . $row['title'] .
                        "</div>
                    </span>
                </div>";
        }
    ?>

</div>

<nav class="optionsMenu">
    <input type="hidden" class="songId">
    <?php echo Playlist::getPlaylistDropdown($con, $userLoggedIn->getUsername()); ?>
</nav>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

# *browse.php*

browse.php

```php
<?php
include("includes/includedFiles.php");
?>

<h1 class="pageHeadingBig">You might also like</h1>

<div class="gridViewContainer">

    <?php
        $albumQuery = mysqli_query($con, "SELECT * FROM albums ORDER BY RAND() LIMIT 10");

        while($row = mysqli_fetch_array($albumQuery)) {

            echo "<div class='gridViewItem'>
                    <span role='link' tabindex='0' onclick='openPage(\"album.php?id=" . $row['id'] . "\")'>
                        <img src='" .  $row['artworkPath'] . "'>
                        <div class='gridViewInfo'>"
                            . $row['title'] .
                        "</div>
                    </span>
                </div>";
        }
    ?>

</div>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

# *playlist.php*

playlist.php

```php
<?php include("includes/includedFiles.php");

if(isset($_GET['id'])) {
    $playlistId = $_GET['id'];
} else {
    header("Location: index.php");
}

$playlist = new Playlist($con, $playlistId);
$owner = new User($con, $playlist->getOwner());

?>

<div class="entityInfo">

    <div class="leftSection">
        <div class="playlistImage">
            <img src="assets/images/icons/playlist.png">
        </div>
    </div>

    <div class="rightSection">
        <h2><?php echo $playlist->getName(); ?></h2>
        <p>By <?php echo $playlist->getOwner(); ?></p>
        <p><?php echo $playlist->getNumberOfSongs(); ?> songs</p>

        <button class="button" onclick="deletePlaylist('<?php echo $playlistId; ?>')">DELETE PLAYLIST</button>
    </div>

</div>

<div class="trackListContainer">
    <ul class="trackList">
        <?php

            $songIdArray = $playlist->getSongIds();
            $i = 1;
            foreach ($songIdArray as $songId) {
                $playlistSong = new Song($con, $songId);
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

playlist.php

```
                        tempPlaylist, true)'>
                        <span class='trackNumber'>$i</span>
                    </div>

                    <div class='trackInfo'>
                        <span class='trackName'>" . $playlistSong->getTitle() . "</span>
                        <span class'artistName'>" . $songArtist->getName() . "</span>
                    </div>

                    <div class='trackOptions'>
                        <input type='hidden' class='songId' value='" . $playlistSong->getId() . "'/>
                        <img class='optionsButton' src='assets/images/icons/more.png' onclick='showOptionsMenu(this)'>
                    </div>

                    <div class='trackDuration'>
                        <span class='duration'>" . $playlistSong->getDuration() . "</span>
                    </div>

                </li>";

            $i++;

        }

    ?>

    <script>
        var tempSongIds = '<?php echo json_encode($songIdArray); ?>';
        tempPlaylist = JSON.parse(tempSongIds);
    </script>

</ul>
</div>

<nav class="optionsMenu">
    <input type="hidden" class="songId">
    <?php echo Playlist::getPlaylistDropdown($con, $userLoggedIn->getUsername()); ?>
    <div class="item" onclick="removeFromPlaylist(this, '<?php echo $playlistId; ?>')">Remove from playlist</div>
</nav>
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

# *register.php*

register.php

```
<?php

    include("includes/config.php");
    include("includes/classes/Account.php");
    include("includes/classes/Constants.php");

    $account = new Account($con);

    include("includes/handlers/register-handler.php");
    include("includes/handlers/login-handler.php");

    function getInputValue($name) {
        if (isset($_POST[$name])) {
            echo $_POST[$name];
        }
    }
?>

<!DOCTYPE html>
<html>
<head>
    <title>Welcome to Spotify!</title>
    <link rel="stylesheet" type="text/css" href="assets/css/register.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    <script src="assets/js/register.js"></script>
</head>
<body>
    <?php
        if (isset($_POST['registerButton'])) {
            echo '<script>
                    $(document).ready(function() {
                        $("#loginForm").hide();
                        $("#registerForm").show();
                    });
                </script>';
        } else {
            echo '<script>
                    $(document).ready(function() {
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

register.php

```
                    <label for="username">Username</label>
                    <input id="username" name="username" type="text" placeholder="e.g. EricStanley" value="<?php getInputValue
                    ('username'); ?>" required>
                </p>

                <p>
                    <?php echo $account -> getError(Constants::$firstNameCharacters); ?>
                    <label for="firstName">First Name</label>
                    <input id="firstName" name="firstName" type="text" placeholder="e.g. Eric" value="<?php getInputValue('firstName'); ?
                    >" required>
                </p>

                <p>
                    <?php echo $account -> getError(Constants::$lastNameCharacters); ?>
                    <label for="lastName">Last Name</label>
                    <input id="lastName" name="lastName" type="text" placeholder="e.g. Stanley" value="<?php getInputValue('lastName'); ?
                    >" required>
                </p>

                <p>
                    <?php echo $account -> getError(Constants::$emailInvalid); ?>
                    <?php echo $account -> getError(Constants::$emailsDoNotMatch); ?>
                    <?php echo $account -> getError(Constants::$emailTaken); ?>
                    <label for="email">Email</label>
                    <input id="email" name="email" type="email" placeholder="e.g. eric@gmail.com" value="<?php getInputValue('email'); ?
                    >" required>
                </p>

                <p>
                    <label for="emailConfirm">Confirm Email</label>
                    <input id="emailConfirm" name="emailConfirm" type="email" placeholder="e.g. eric@gmail.com" value="<?php
                    getInputValue('emailConfirm'); ?>" required>
                </p>

                <p>
                    <?php echo $account -> getError(Constants::$passwordsDoNotMatch); ?>
                    <?php echo $account -> getError(Constants::$passwordNotAlphanumeric); ?>
                    <?php echo $account -> getError(Constants::$passwordCharacters); ?>
                    <label for="password">Password</label>
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

register.php

```
                </p>

                <p>
                    <label for="passwordConfirm">Confirm Password</label>
                    <input id="passwordConfirm" name="passwordConfirm" type="password" placeholder="Your password" required>
                </p>

                <button type="submit" name="registerButton">SIGN UP</button>

                <div class="hasAccountText">
                    <span id="hideRegister">Already have an account? Login here.</span>
                </div>

            </form>

        </div>

        <div id="loginText">
            <h1>Get great music, right now</h1>
            <h2>Listen to loads of songs for free</h2>
            <ul>
                <li>Discover music you'll fall in love with</li>
                <li>Create your own playlist</li>
                <li>Follow artists to keep up to date</li>
            </ul>
        </div>
    </div>
</div>

</body>
</html>
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

# *search.php*

search.php

```php
<?php

include("includes/includedFiles.php");

if (isset($_GET['term'])) {
    $term = urldecode($_GET['term']);
} else {
    $term = "";
}

?>

<div class="searchContainer">

    <h4>Search for an artist, album or song</h4>
    <input type="text" class="searchInput" value="<?php echo $term; ?>" placeholder="Start typing..."
        onfocus="moveCursorToEnd(this);">

</div>

<script>

    $(".searchInput").focus();

    $(function() {
        $(".searchInput").keyup(function() {
            clearTimeout(timer);

            timer = setTimeout(function() {
                var val = $(".searchInput").val();
                openPage("search.php?term=" + val);
            }, 2000);
        });
    });

    function moveCursorToEnd(el) {
        if (typeof el.selectionStart == "number") {
            el.selectionStart = el.selectionEnd = el.value.length;
        } else if (typeof el.createTextRange != "undefined") {
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

search.php

```php
    <h2>SONGS</h2>
    <ul class="trackList">
        <?php

            $songsQuery = mysqli_query($con, "SELECT id FROM songs WHERE title LIKE '%$term%' LIMIT 10");

            if (mysqli_num_rows($songsQuery) == 0) {
                echo "<span class='noResults'>No songs found matching " . $term . "</span>";
            }

            $songIdArray = array();
            $i = 1;
            while ($row = mysqli_fetch_array($songsQuery)) {

                if ($i > 15) {
                    break;
                }

                array_push($songIdArray, $row['id']);

                $albumSong = new Song($con, $row['id']);
                $albumArtist = $albumSong->getArtist();

                echo "<li class='trackListRow'>
                        <div class='trackCount'>
                            <img class='play' src='assets/images/icons/play-white.png' onclick='setTrack(\"" . $albumSong->getId() . "\",
                            tempPlaylist, true)'>
                            <span class='trackNumber'>$i</span>
                        </div>

                        <div class='trackInfo'>
                            <span class='trackName'>" . $albumSong->getTitle() . "</span>
                            <span class'artistName'>" . $albumArtist->getName() . "</span>
                        </div>

                        <div class='trackOptions'>
                            <input type='hidden' class='songId' value='" . $albumSong->getId() . "'/>
                            <img class='optionsButton' src='assets/images/icons/more.png' onclick='showOptionsMenu(this)'>
                        </div>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

search.php

```php
    ?>

</div>

<div class="gridViewContainer">

    <h2>ALBUMS</h2>

    <?php
        $albumQuery = mysqli_query($con, "SELECT * FROM albums WHERE title LIKE '%$term%' LIMIT 10");

        if (mysqli_num_rows($albumQuery) == 0) {
            echo "<span class='noResults'>No albums found matching " . $term . "</span>";
        }

        while($row = mysqli_fetch_array($albumQuery)) {

            echo "<div class='gridViewItem'>
                    <span role='link' tabindex='0' onclick='openPage(\"album.php?id=" . $row['id'] . "\")'>
                        <img src='" .  $row['artworkPath'] . "'>
                        <div class='gridViewInfo'>"
                            . $row['title'] .
                        "</div>
                    </span>
                </div>";
        }
    ?>

</div>


<nav class="optionsMenu">
    <input type="hidden" class="songId">
    <?php echo Playlist::getPlaylistDropdown($con, $userLoggedIn->getUsername()); ?>
</nav>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

# *settings.php*

settings.php

```php
<?php

include("includes/includedFiles.php");

?>

<div class="entityInfo">

    <div class="centerSection">
        <div class="userInfo">
            <h1><?php echo $userLoggedIn->getFirstAndLastName(); ?></h1>
        </div>
    </div>

    <div class="buttonItems">
        <button class="button" onclick="openPage('updateDetails.php')">USER DETAILS</button>
        <button class="button" onclick="logout()">LOGOUT</button>
    </div>

</div>
```

Ln 1, Col 1  Tab Size: 4  UTF-8  LF  PHP  Go Live

# *updateDetails.php*

updateDetails.php

```php
<?php
include("includes/includedFiles.php");
?>

<div class="userDetails">

    <div class="container borderBottom">
        <h2>EMAIL</h2>
        <input type="text" class="email" name="email" placeholder="Email address..." value="<?php echo $userLoggedIn->getEmail() ?>">
        <span class="message"></span>
        <button class="button" onclick="updateEmail('email')">SAVE</button>
    </div>

    <div class="container">
        <h2>PASSWORD</h2>
        <input type="password" class="oldPassword" name="oldPassword" placeholder="Current password">
        <input type="password" class="newPassword1" name="newPassword1" placeholder="New password">
        <input type="password" class="newPassword2" name="newPassword2" placeholder="Confirm password">
        <span class="message"></span>
        <button class="button" onclick="updatePassword('oldPassword', 'newPassword1', 'newPassword2')">SAVE</button>
    </div>

</div>

```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

# *yourMusic.php*

yourMusic.php

```php
<?php

include("includes/includedFiles.php");

?>

<div class="playlistContainer">

    <div class="gridViewContainer">
        <h2>PLAYLISTS</h2>

        <div class="buttonItems">
            <button class="button green" onclick="createPlaylist()">NEW PLAYLIST</button>
        </div>

        <?php
            $username = $userLoggedIn->getUsername();
            $playlistQuery = mysqli_query($con, "SELECT * FROM playlists WHERE owner='$username'");

            if (mysqli_num_rows($playlistQuery) == 0) {
                echo "<span class='noResults'>You don't have any playlists yet!</span>";
            }

            while($row = mysqli_fetch_array($playlistQuery)) {

                $playlist = new Playlist($con, $row);

                echo "<div class='gridViewItem' role='link' tabIndex='0'
                    onclick='openPage(\"playlist.php?id=" . $playlist->getId() . "\")'>

                        <div class='playlistImage'>
                            <img src='assets/images/icons/playlist.png'>
                        </div>

                        <div class='gridViewInfo'>"
                            . $playlist->getName() .
                        "</div>
                    </div>";
            }
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF PHP Go Live

# *music.sql*

```sql
Execute
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
Execute
SET AUTOCOMMIT = 0;
Execute
START TRANSACTION;
Execute
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `spotify`
--

-- --------------------------------------------------------

--
-- Table structure for table `albums`
--

Execute
CREATE TABLE `albums` (
  `id` int(11) NOT NULL,
  `title` varchar(250) NOT NULL,
  `artist` int(11) NOT NULL,
  `genre` int(11) NOT NULL,
  `artworkPath` varchar(500) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `albums`
--

Execute
INSERT INTO `albums` (`id`, `title`, `artist`, `genre`, `artworkPath`) VALUES
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF SQL SQLite: No database Go Live

```sql
Execute
CREATE TABLE `artists` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `artists`
--

Execute
INSERT INTO `artists` (`id`, `name`) VALUES
(1, 'Mickey Mouse'),
(2, 'Goofy'),
(3, 'Eric Stanley'),
(4, 'Homer'),
(5, 'Bruce Lee');

-- --------------------------------------------------------

--
-- Table structure for table `genres`
--

Execute
CREATE TABLE `genres` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `genres`
--

Execute
INSERT INTO `genres` (`id`, `name`) VALUES
(1, 'Rock'),
(2, 'Pop'),
(3, 'Hip Hop'),
(4, 'R & B'),
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF SQL SQLite: No database Go Live

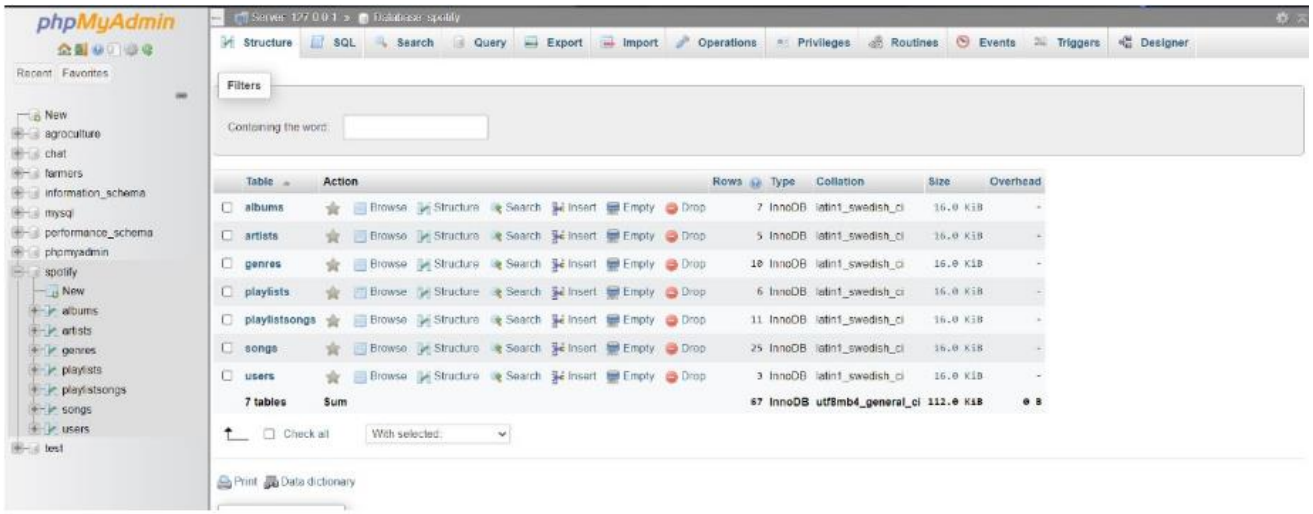
```sql
CREATE TABLE `playlists` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `owner` varchar(50) NOT NULL,
  `dateCreated` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `playlists`
--

INSERT INTO `playlists` (`id`, `name`, `owner`, `dateCreated`) VALUES
(2, 'Melodies', 'ericstanley', '2019-10-14 00:00:00'),
(3, 'Rock', 'ericstanley', '2019-10-14 00:00:00'),
(4, 'Beats', 'ericstanley', '2019-10-18 00:00:00'),
(5, 'Jazz', 'ericstanley', '2019-10-18 00:00:00'),
(6, 'Pop', 'ericstanley', '2019-10-18 00:00:00'),
(7, 'Hits', 'ericstanley', '2019-10-18 00:00:00');

-- --------------------------------------------------------

--
-- Table structure for table `playlistSongs`
--

CREATE TABLE `playlistSongs` (
  `id` int(11) NOT NULL,
  `songId` int(11) NOT NULL,
  `playlistId` int(11) NOT NULL,
  `playlistOrder` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `playlistSongs`
--
```

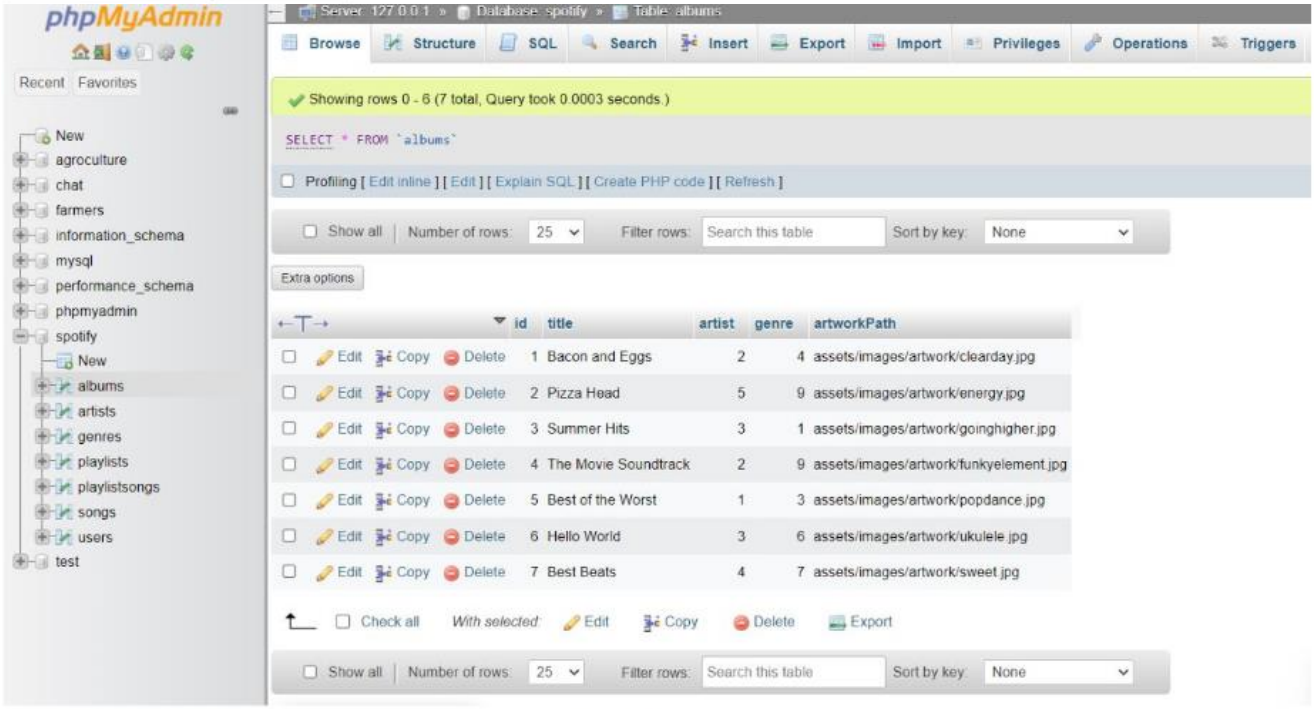
Ln 1, Col 1 Spaces: 2 UTF-8 LF SQL SQLite: No database Go Live

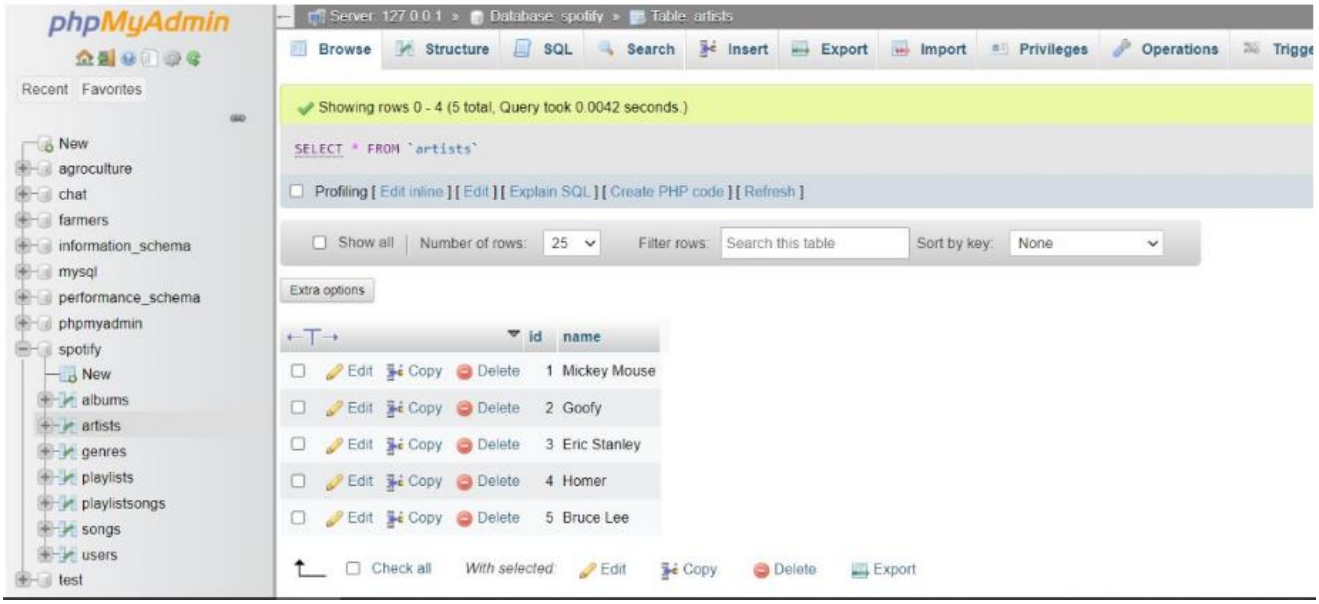
```sql
ALTER TABLE `artists`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;

--
-- AUTO_INCREMENT for table `genres`
--
ALTER TABLE `genres`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;

--
-- AUTO_INCREMENT for table `playlists`
--
ALTER TABLE `playlists`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;

--
-- AUTO_INCREMENT for table `playlistSongs`
--
ALTER TABLE `playlistSongs`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=18;

--
-- AUTO_INCREMENT for table `songs`
--
ALTER TABLE `songs`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=26;

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;
COMMIT;
```

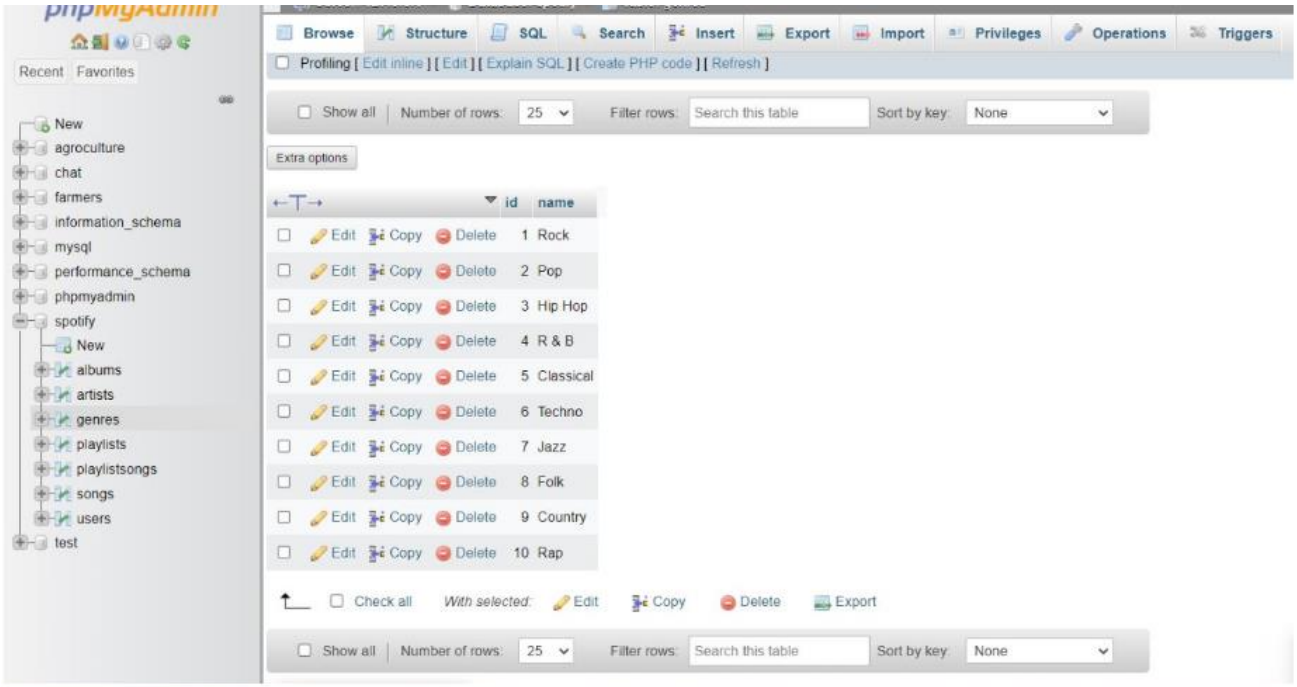
Ln 1, Col 1 Spaces: 2 UTF-8 LF SQL SQLite: No database Go Live

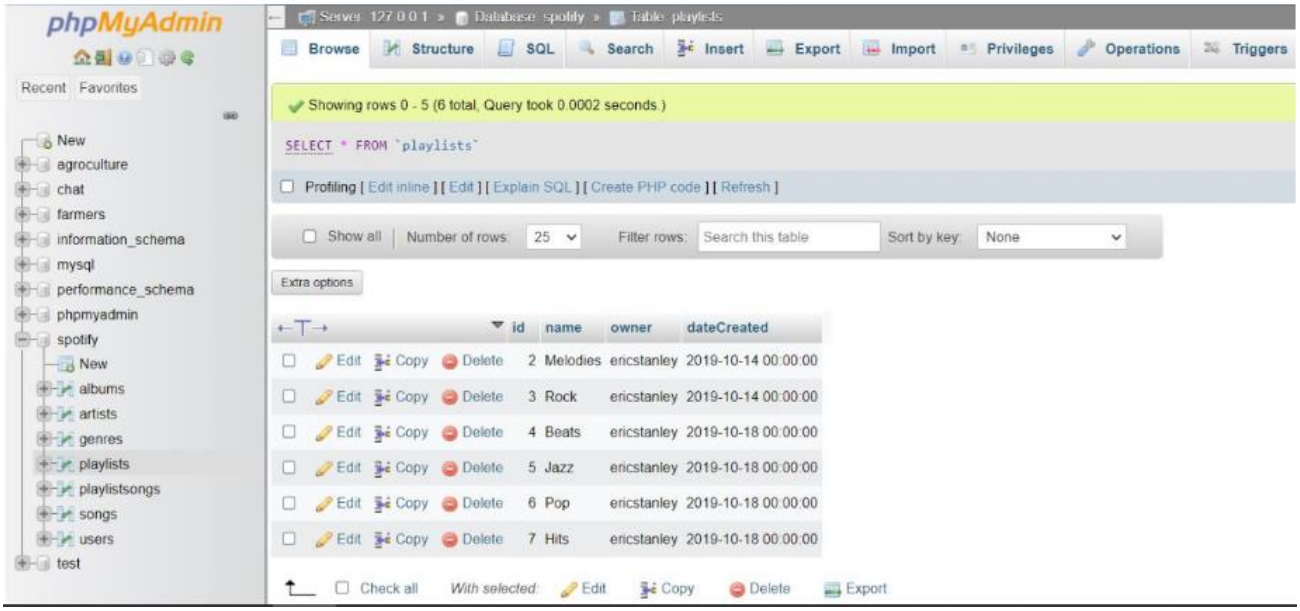
# *Database*

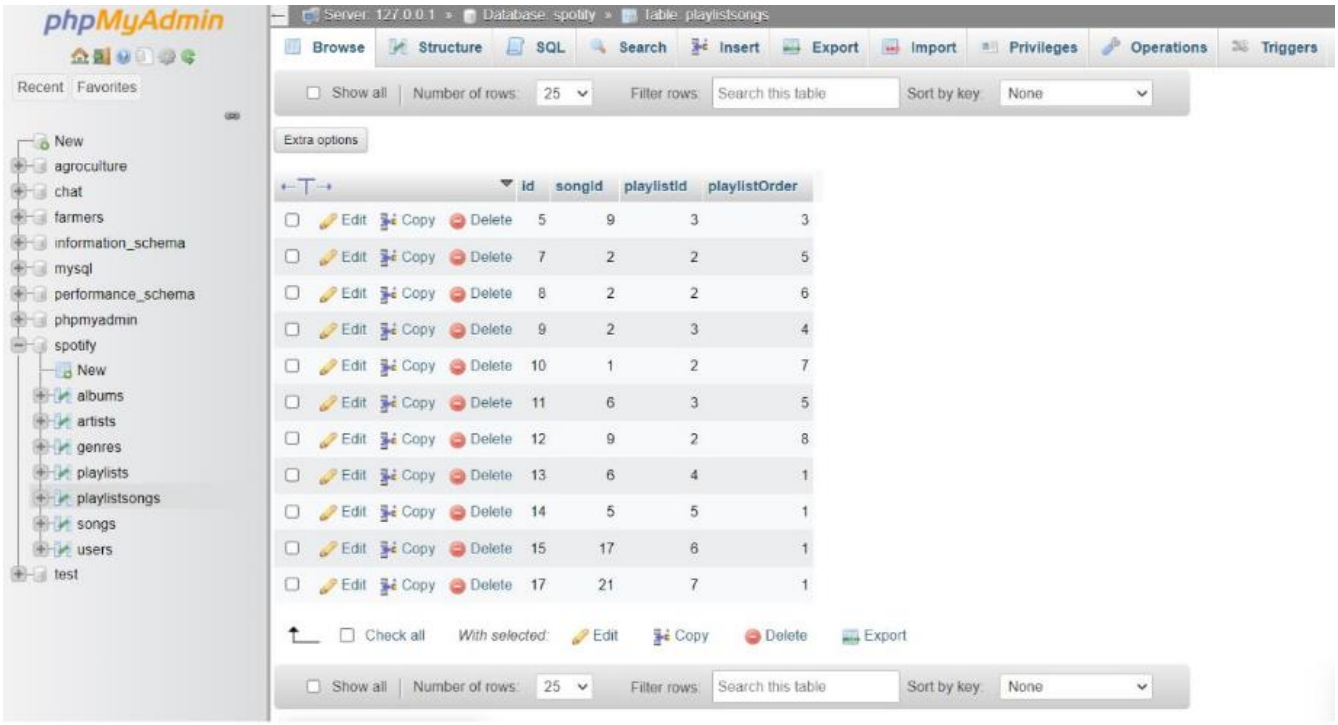

phpMyAdmin
Server: 127.0.0.1 » Database: spotify
Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers Designer
Filters
Containing the word:
Table Action Rows Type Collation Size Overhead
albums Browse Structure Search Insert Empty Drop 7 InnoDB latin1_swedish_ci 16.0 KiB -
artists Browse Structure Search Insert Empty Drop 5 InnoDB latin1_swedish_ci 16.0 KiB -
genres Browse Structure Search Insert Empty Drop 10 InnoDB latin1_swedish_ci 16.0 KiB -
playlists Browse Structure Search Insert Empty Drop 6 InnoDB latin1_swedish_ci 16.0 KiB -
playlistsongs Browse Structure Search Insert Empty Drop 11 InnoDB latin1_swedish_ci 16.0 KiB -
songs Browse Structure Search Insert Empty Drop 25 InnoDB latin1_swedish_ci 16.0 KiB -
users Browse Structure Search Insert Empty Drop 3 InnoDB latin1_swedish_ci 16.0 KiB -
7 tables Sum 67 InnoDB utf8mb4_general_ci 112.0 KiB 0 B
Check all With selected:
Print Data dictionary
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test

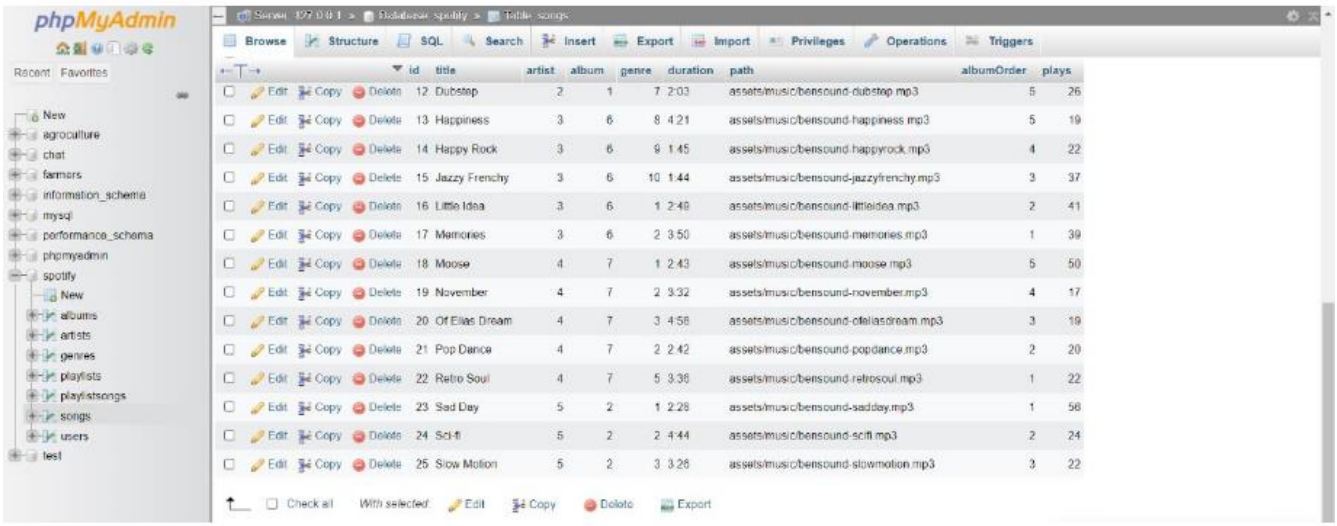

phpMyAdmin
Server: 127.0.0.1 » Database: spotify » Table: albums
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Showing rows 0 - 6 (7 total, Query took 0.0003 seconds.)
SELECT * FROM `albums`
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id title artist genre artworkPath
Edit Copy Delete 1 Bacon and Eggs 2 4 assets/images/artwork/clearday.jpg
Edit Copy Delete 2 Pizza Head 5 9 assets/images/artwork/energy.jpg
Edit Copy Delete 3 Summer Hits 3 1 assets/images/artwork/goinghigher.jpg
Edit Copy Delete 4 The Movie Soundtrack 2 9 assets/images/artwork/funkyelement.jpg
Edit Copy Delete 5 Best of the Worst 1 3 assets/images/artwork/popdance.jpg
Edit Copy Delete 6 Hello World 3 6 assets/images/artwork/ukulele.jpg
Edit Copy Delete 7 Best Beats 4 7 assets/images/artwork/sweet.jpg
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test


phpMyAdmin
Server: 127.0.0.1 » Database: spotify » Table: artists
Browse Structure SQL Search Insert Export Import Privileges Operations
Showing rows 0 - 4 (5 total, Query took 0.0042 seconds.)
SELECT * FROM `artists`
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id name
Edit Copy Delete 1 Mickey Mouse
Edit Copy Delete 2 Goofy
Edit Copy Delete 3 Eric Stanley
Edit Copy Delete 4 Homer
Edit Copy Delete 5 Bruce Lee
Check all With selected: Edit Copy Delete Export
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test


phpMyAdmin
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id name
Edit Copy Delete 1 Rock
Edit Copy Delete 2 Pop
Edit Copy Delete 3 Hip Hop
Edit Copy Delete 4 R & B
Edit Copy Delete 5 Classical
Edit Copy Delete 6 Techno
Edit Copy Delete 7 Jazz
Edit Copy Delete 8 Folk
Edit Copy Delete 9 Country
Edit Copy Delete 10 Rap
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None


phpMyAdmin
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test
Server: 127.0.0.1 » Database: spotify » Table: playlists
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Showing rows 0 - 5 (6 total, Query took 0.0002 seconds.)
SELECT * FROM `playlists`
Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id name owner dateCreated
Edit Copy Delete 2 Melodies ericstanley 2019-10-14 00:00:00
Edit Copy Delete 3 Rock ericstanley 2019-10-14 00:00:00
Edit Copy Delete 4 Beats ericstanley 2019-10-18 00:00:00
Edit Copy Delete 5 Jazz ericstanley 2019-10-18 00:00:00
Edit Copy Delete 6 Pop ericstanley 2019-10-18 00:00:00
Edit Copy Delete 7 Hits ericstanley 2019-10-18 00:00:00
Check all With selected: Edit Copy Delete Export


phpMyAdmin
Recent Favorites
New
agroculture
chat
farmers
information_schema
mysql
performance_schema
phpmyadmin
spotify
New
albums
artists
genres
playlists
playlistsongs
songs
users
test
Server: 127.0.0.1 » Database: spotify » Table: playlistsongs
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None
Extra options
id songId playlistId playlistOrder
Edit Copy Delete 5 9 3 3
Edit Copy Delete 7 2 2 5
Edit Copy Delete 8 2 2 6
Edit Copy Delete 9 2 3 4
Edit Copy Delete 10 1 2 7
Edit Copy Delete 11 6 3 5
Edit Copy Delete 12 9 2 8
Edit Copy Delete 13 6 4 1
Edit Copy Delete 14 5 5 1
Edit Copy Delete 15 17 6 1
Edit Copy Delete 17 21 7 1
Check all With selected: Edit Copy Delete Export
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None


phpMyAdmin
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
id title artist album genre duration path albumOrder plays
Edit Copy Delete 12 Dubstep 2 1 7 2:03 assets/music/bensound-dubstep.mp3 5 26
Edit Copy Delete 13 Happiness 3 6 8 4:21 assets/music/bensound-happiness.mp3 5 19
Edit Copy Delete 14 Happy Rock 3 6 9 1:45 assets/music/bensound-happyrock.mp3 4 22
Edit Copy Delete 15 Jazzy Frenchy 3 6 10 1:44 assets/music/bensound-jazzyfrenchy.mp3 3 37
Edit Copy Delete 16 Little Idea 3 6 1 2:49 assets/music/bensound-littleidea.mp3 2 41
Edit Copy Delete 17 Memories 3 6 2 3:50 assets/music/bensound-memories.mp3 1 39
Edit Copy Delete 18 Moose 4 7 1 2:43 assets/music/bensound-moose.mp3 5 50
Edit Copy Delete 19 November 4 7 2 3:32 assets/music/bensound-november.mp3 4 17
Edit Copy Delete 20 Of Elias Dream 4 7 3 4:58 assets/music/bensound-ofeliasdream.mp3 3 19
Edit Copy Delete 21 Pop Dance 4 7 2 2:42 assets/music/bensound-popdance.mp3 2 20
Edit Copy Delete 22 Retro Soul 4 7 5 3:36 assets/music/bensound-retrosoul.mp3 1 22
Edit Copy Delete 23 Sad Day 5 2 1 2:28 assets/music/bensound-sadday.mp3 1 58
Edit Copy Delete 24 Sci-fi 5 2 2 4:44 assets/music/bensound-scifi.mp3 2 24
Edit Copy Delete 25 Slow Motion 5 2 3 3:26 assets/music/bensound-slowmotion.mp3 3 22
Check all With selected: Edit Copy Delete Export

# TYPE OF CONNECTIVITY

- Local connectivity can be used to backup and restore the music library management system's data to a local database. This can help ensure that the data is always available in case of a system failure or data loss.

- Local connectivity can also be used for testing and development purposes, allowing developers to work on the application and database on their local machines without the need for network connectivity. This can help to improve the development process and reduce the risk of errors.

- Remote connectivity can be used to access the music library management system from anywhere with an internet connection, making it more convenient for users to manage their music library from different locations.

- Cloud connectivity can be utilized to store the data in the cloud, enabling easy access to the data from anywhere and providing better data security.

- Mobile connectivity can be implemented to enable users to access the music library management system using their mobile devices, increasing the accessibility and convenience of the system.

- Wi-Fi connectivity can be utilized to connect the music library management system to a wireless network, enabling users to access and manage their music library from any device connected to the network.

- Bluetooth connectivity can be implemented to allow users to connect their mobile devices to the music library management system wirelessly, enabling them to stream music directly from the system.

- Wired connectivity can be used to connect the music library management system to a wired network, providing a reliable and stable connection for managing the data.

# LIST OF MODULES AND FUNCTIONALITIES

- **Album Management** - This module allows users to add new albums, edit existing ones, and delete albums from the music library.
- **Song Management** - This module allows users to manage songs by adding, editing, and deleting them from the music library.
- **Search** - This module enables users to search for specific songs in the music library by artist, album, or song name.
- **Playlist Generation** - This module enables users to generate playlists based on their music preferences and library.
- **User Management** - This module allows administrators to manage user accounts, including adding new users, managing permissions, and handling user requests.

# RESULT AND DISCUSSION

For a music library management system, here are some possible results and discussions:

***Results:***

- **Improved organization and accessibility:** A music library management system can help music libraries to organize and access their music collections more efficiently by storing and categorizing music tracks and metadata, such as artist, album, genre, and release date. This can result in better searchability and increased productivity.

- **Enhanced user experience:** By providing a user-friendly interface, a music library management system can improve the user experience for music library patrons, allowing them to easily search, stream, and download music tracks from the library's collection.

- **Better copyright compliance:** With the help of a music library management system, music libraries can track and manage copyright permissions and licenses, ensuring that they are in compliance with copyright laws and regulations.

- **Improved analytics:** By analysing data related to music track popularity, user behaviour, and library usage, a music library management system can help music libraries make informed decisions about music collection development and marketing.

***Discussion:***

- **Integration with existing systems:** A music library management system needs to integrate with other systems already in place in the library, such as music streaming platforms, licensing databases, and accounting software. Libraries need to ensure that the new system is compatible and can communicate with existing systems.

- **Cost-benefit analysis:** The implementation of a music library management system involves upfront costs such as software and hardware purchases, installation, and training. Libraries need to evaluate the potential benefits of the system against these costs to determine the ROI.

- **Data privacy and security:** With the increasing use of technology in music libraries, data privacy and security have become a significant concern. Libraries need to ensure that the music library management system they choose has adequate measures in place to protect their data from unauthorized access or theft.

- **User adoption:** The success of a music library management system depends on how well it is adopted by the users. Libraries need to ensure that the system is user-friendly, and they receive adequate training and support to use it effectively.

Overall, a music library management system can bring significant benefits to music libraries by improving organization and accessibility, enhancing user experience, ensuring copyright compliance, and providing improved analytics. However, libraries need to ensure compatibility with existing systems, evaluate the potential benefits against the upfront costs, address data privacy and security concerns, and ensure adequate user adoption.

# CONCLUSION AND FUTURE ENHANCEMENT

- A well-designed music library management system helps music lovers organize and manage their music collections more efficiently.
- It reduces clutter and duplicate files and helps make more informed decisions about what music to listen to.
- Stakeholders, including music enthusiasts, music librarians, and music experts, should be involved in the development process.
- Appropriate technology platforms should be selected, and the system should be user-friendly and easy to adopt.
- Future requirements may include incorporating advanced technologies such as machine learning and natural language processing to provide more accurate and personalized music recommendations.
- Integration with other systems such as online music stores and streaming platforms could enhance the system's value for music lovers.
- The project's success will depend on careful planning, stakeholder engagement, and a willingness to incorporate new technologies and adapt to changing conditions in the music industry.

# REFERENCES


- Python - https://docs.python.org/3/
- MySQL - https://dev.mysql.com/doc/
- Flask - https://flask.palletsprojects.com/en/2.3.x/installation/#python-version/
- SQLAlchemy - https://docs.sqlalchemy.org/en/20/
- Apache - https://httpd.apache.org/docs-project/
- XAMPP - https://www.apachefriends.org/docs/
- PasswordSecurity- https://werkzeug.palletsprojects.com/en/2.3.x/installation/#python-version
- phpMyAdmin- https://docs.phpmyadmin.net/en/latest/
- Youtube - https://youtu.be/00lskS-XBnE
- VSCode - https://code.visualstudio.com/docs
- MySQLClient- https://pypi.org/project/mysqlclient/
- jQuery - https://plugins.jquery.com/
- AOS - https://plugins.jquery.com/