

# **Software Requirements Specification**

**for**

# **Music Sharing System**

**Version <0.0.3>**

## **Group India**

<b>Chee Rui</b>	<b>1211112287</b>
<b>Tan Ee Hang</b>	<b>1211108082</b>
<b>Chan Ka Ken</b>	<b>1211109440</b>
<b>Mei Yong Peng</b>	<b>1211109159</b>

**Date: 12<sup>th</sup> February 2025**

# Contents

<b>CONTENTS .....</b>	<b>2</b>
<b>REVISIONS .....</b>	<b>4</b>
<b>1 PROJECT INTRODUCTION .....</b>	<b>5</b>
1.1 TEAM MEMBERS.....	5
1.2 PROBLEM STATEMENT.....	5
1.3 PROJECT PLAN.....	6
<b>2 SYSTEM OVERVIEW .....</b>	<b>7</b>
2.1 DESCRIPTION .....	7
2.2 ACTORS .....	7
2.3 ASSUMPTIONS AND DEPENDENCIES .....	8
2.4 USE CASE DIAGRAM .....	8
<b>3 SCENARIO – BASED MODELING.....</b>	<b>9</b>
3.1 MUSIC CREATOR.....	9
3.2 USER .....	13
<b>4 REQUIREMENTS MODELING.....</b>	<b>16</b>
4.1 CLASS DIAGRAMS / ERD .....	16
4.2 CLASSES / ENTITIES.....	18
<b>5 BEHAVIORAL &amp; FLOW MODELING.....</b>	<b>21</b>
5.1 SEQUENCE DIAGRAMS .....	21
<b>6 ARCHITECTURE DESIGN.....</b>	<b>24</b>
6.1 SOFTWARE ARCHITECTURE.....	24
<b>7 INTERFACE DESIGN .....</b>	<b>27</b>
7.1 MAIN SCREENS .....	27
7.2 PLAYING MUSIC SCREEN .....	29
7.3 LOGIN / SIGN-IN SCREEN .....	31
7.4 PROFILE / PLAYLIST SCREEN.....	33
7.5 MUSIC ANALYTIC SCREEN .....	34
<b>8 COMPONENT DESIGN .....</b>	<b>35</b>
8.1 MAIN LAYERS .....	35
<b>9 DEPLOYMENT DIAGRAM.....</b>	<b>39</b>
9.1 DEPLOYMENT DIAGRAM .....	39
9.2 DATA FLOW:.....	40
<b>10 IMPLEMENTATION .....</b>	<b>41</b>
10.1 DEVELOPMENT ENVIRONMENT .....	41
10.2 SOFTWARE INTEGRATION.....	43
10.3 DATABASE .....	46
<b>11 TESTING.....</b>	<b>47</b>
11.1 TESTING STRATEGY .....	47
11.2 TEST DATA.....	48
11.3 ACCEPTANCE TESTING .....	48

<b>12</b>	<b>SAMPLE SCREENS .....</b>	<b>49</b>
12.1	MAIN SCREEN .....	49
<b>13</b>	<b>CONCLUSION .....</b>	<b>53</b>
<b>14</b>	<b>USER GUIDE.....</b>	<b>55</b>

## Revisions

Version	Primary Author(s)	Description of Version	Date Completed
V.0.0.1	Chee Rui Chan Ka Ken Mei Yong Peng Tan Ee Hang	<p>This is the initial version of the project report, detailing the abstract, system overview, assumptions and dependencies</p> <p><b>Details:</b></p> <ul style="list-style-type: none"><li>• Listed out the roles of users, music creators and moderators</li><li>• Created the use case and and class diagrams</li></ul> <p><b>Notes:</b> This version creates the foundation for future alterations and provides a brief explanation for the overall project</p>	8/12/2024
V.0.0.2	Chee Rui Chan Ka Ken Mei Yong Peng Tan Ee Hang	<p>Updated certain functionalities and design systems.</p> <p><b>Notable Additions:</b></p> <ul style="list-style-type: none"><li>• Architecture Design</li><li>• Interface Design</li><li>• Component Design</li><li>• Deployment Diagram</li></ul> <p><b>Note:</b> This version designed the components and the overall architecture for a better visualisation of the system</p>	14/1/2025

Version	Primary Author(s)	Description of Version	Date Completed
V.0.0.3	Chee Rui Chan Ka Ken Mei Yong Peng Tan Ee Hang	<p>This is the final version of the report, detailing the sample screens, testing process, implementation of the final product and a conclusion for this project with a user guide so readers will have a better time understanding</p> <p><b>Notes:</b> This version provides a final tie for the whole project, while documenting how the software was implemented and how to use, with final remarks accompanying</p>	12/2/2025

## 1 Project Introduction

### 1.1 Team Members

Name	Actor/Processes
Chee Rui	User Interface and general system
Tan Ee Hang	Features and Analytics
Chan Ka Ken	Database and Error Handling
Mei Yong Peng	Website Design

### 1.2 Problem statement

Internet users tend to prefer straightforward, simple ways to share content, typically through links, whether with others or across different devices. However, there is a noticeable lack of such platforms for sharing music files in this field. To address this gap, this project aims to develop a music-sharing application that allows users to easily share music via links, while incorporating extra features such as playlists, subscription options, and anonymity.

The application serves two primary user groups, Music Creators and Listeners. Music creators can upload their tracks, generate unique sharing links, and have the option to mark their content as premium or shareable. Listeners can access these links to stream or download music, with premium content requiring payment.

The primary objective of this project is to create a functional application for file sharing within a controlled and intuitive environment. Key features include file upload/download capabilities, subscription-based premium access, and a feedback system for user interaction. Developed as an assignment, this project demonstrates core application development principles and processes in a practical, real-world scenario.

### 1.3 Project Plan

Task   Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
Communication														
Planning														
Modelling														
Construction														
Deployment														

## 2 System Overview

### 2.1 Description

BoopHub, the music-sharing application we designed, aims to provide users with a simple, worry-free platform for sharing and engaging with music. By eliminating moderators and search functions, the app ensures a straightforward and non-intrusive user experience.

Music creators can upload music files, configure access settings, such as marking content as premium or shareable, and generate unique links for direct sharing. Music metrics such as number of plays, likes and dislikes are also displayed in a simplistic statistics dashboard to help the creator gain direct insight of their music performance. Monetization is also supported through premium track options, enabling creators to earn revenue directly through this application.

As the system architecture is designed for simplicity and scalability, it leverages MySQL for secure storage and effective link management, paired with a user-friendly front-end for seamless navigation. Direct sharing links serve as the primary discovery mechanisms for the public, requiring the creator to spread out the music on their own, fostering a straightforward and individual-centered environment. Playlists also serves as the only way of video storing, which minimizes the burden of the server by deducting the need of saving user's previous seen videos.

This system prioritizes simplicity, autonomy, and user empowerment, making it an ideal platform for music creators, listeners and anyone looking to share their music without unnecessary complexity.

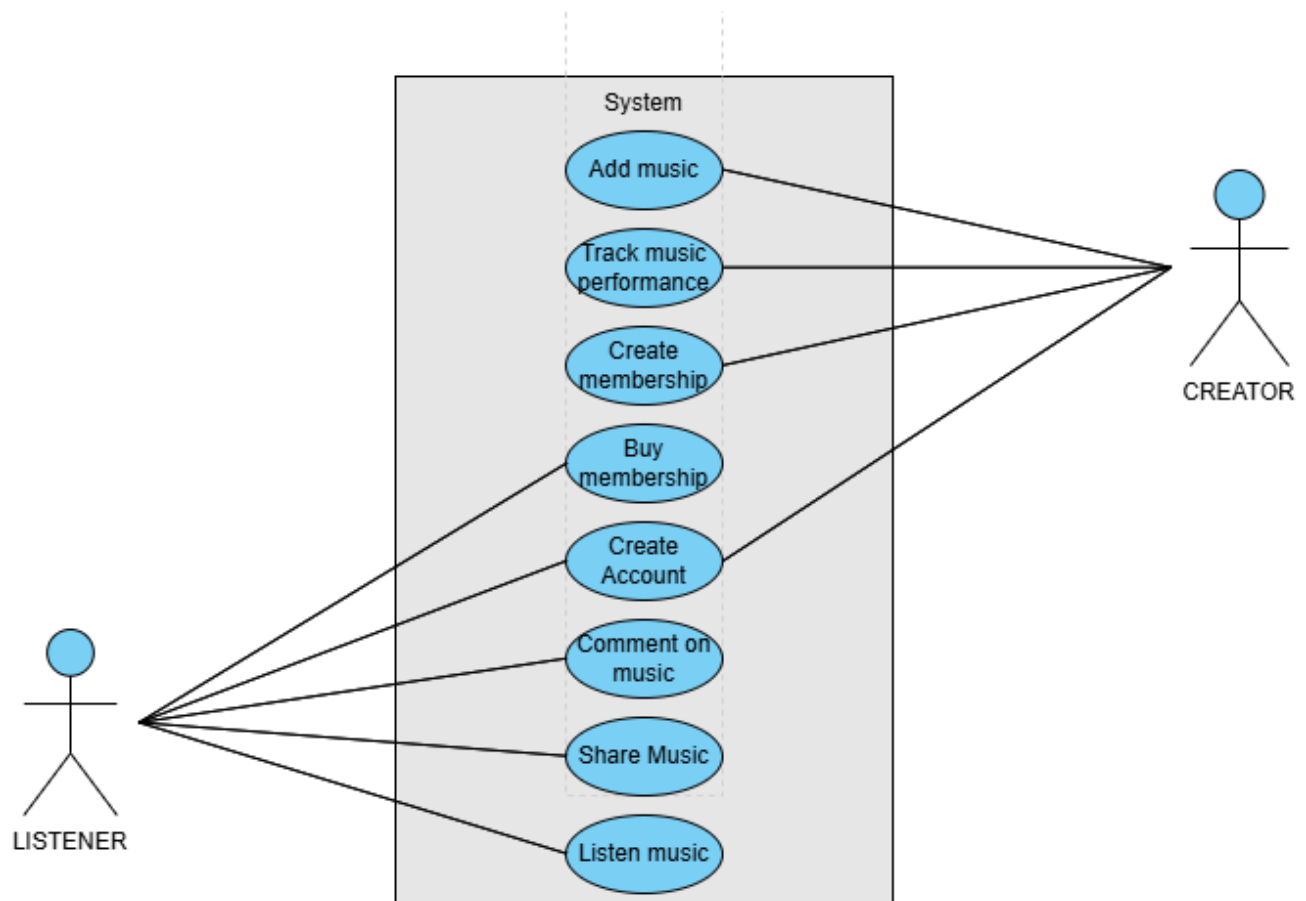
### 2.2 Actors

Actor	Use Cases
Music Creator	<ol style="list-style-type: none"><li>1. Uploads and Manage Contents</li><li>2. View Analytics</li><li>3. Create Membership</li></ol>
User	<ol style="list-style-type: none"><li>1. Rate content (like, comment and share)</li><li>2. Listen to music</li><li>3. Store music</li><li>4. Create Playlist</li></ol>

## 2.3 Assumptions and Dependencies

Assumptions	Dependencies
Users will access the app primarily on computers.	The platform relies on mySql to store user data and music file.
Users will have access to a stable internet connection to interact with the platform for sharing, storing or playing the music	Flask framework is used for frontend and backend.
Uploaded files will be limited to 100 MB per file.	Pythonanywhere.com is used to host the music sharing application.
The client's database is able to store all the data created	URL shortening service (e.g., Bitly) to create concise, shareable links.

## 2.4 Use Case Diagram

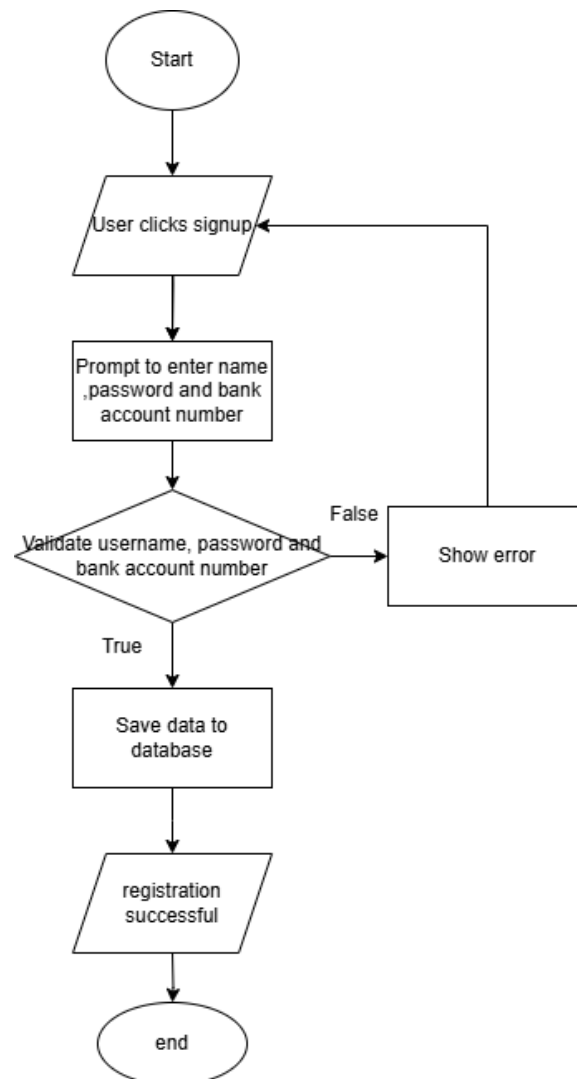




## 3 Scenario – Based Modeling

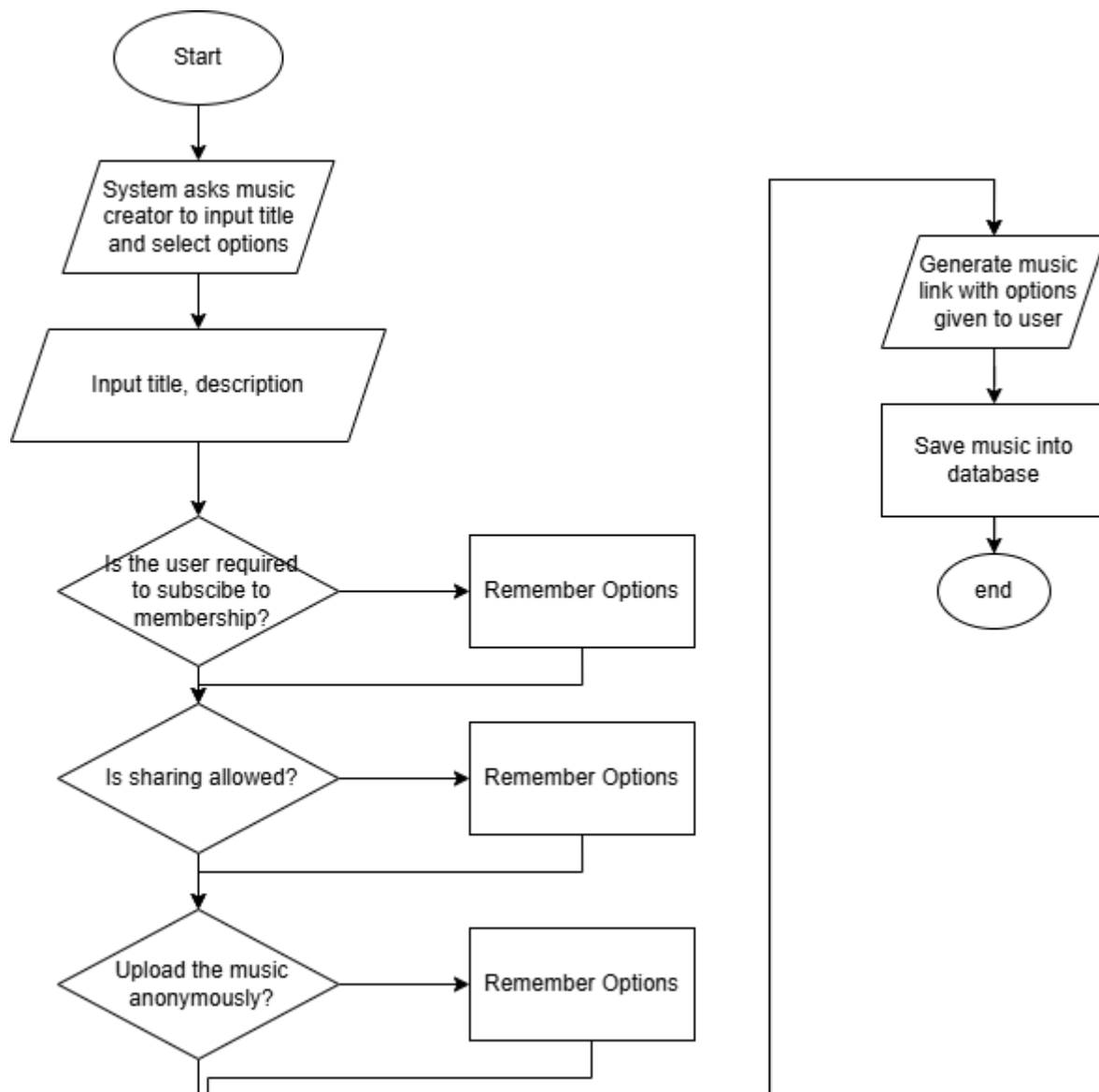
### 3.1 Music Creator

#### 3.1.1 Create Account



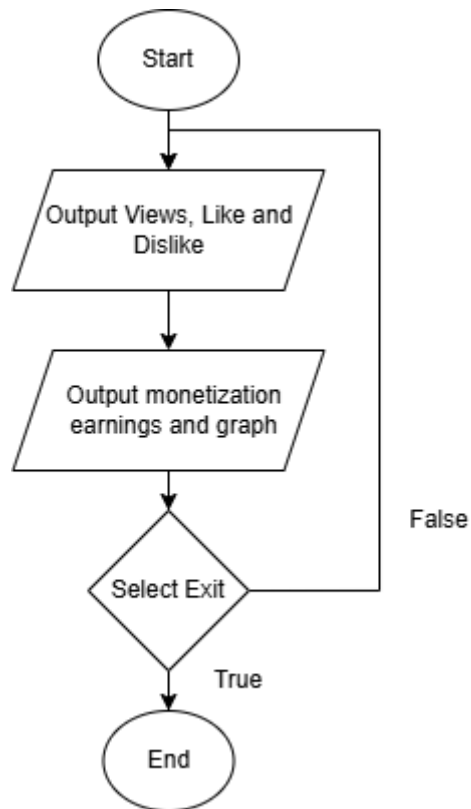
The music creator is required to enter their name, password and bank account number (for monetization purposes). The system will validate if name is available and if all pass the requirements. If yes, all data would be entered into the system database.

### 3.1.2 Add Music



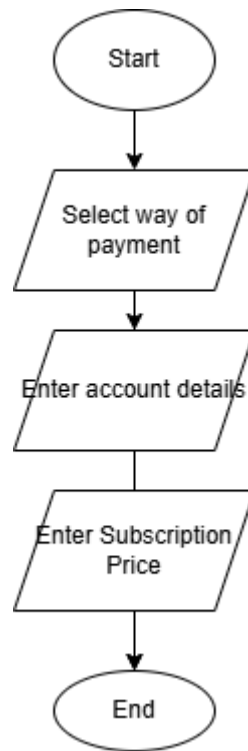
Music creator is able to add music. Music creator is required to input music title and description while selecting whether the music requires the user to subscribe to their membership, if sharing is allowed by another user or if they want to upload the music anonymously.

### 3.1.3 Track music performance



*Music Creator are able to view and track music performance of their specific video link. Likes, dislikes and views will be visualized with earnings in graph form.*

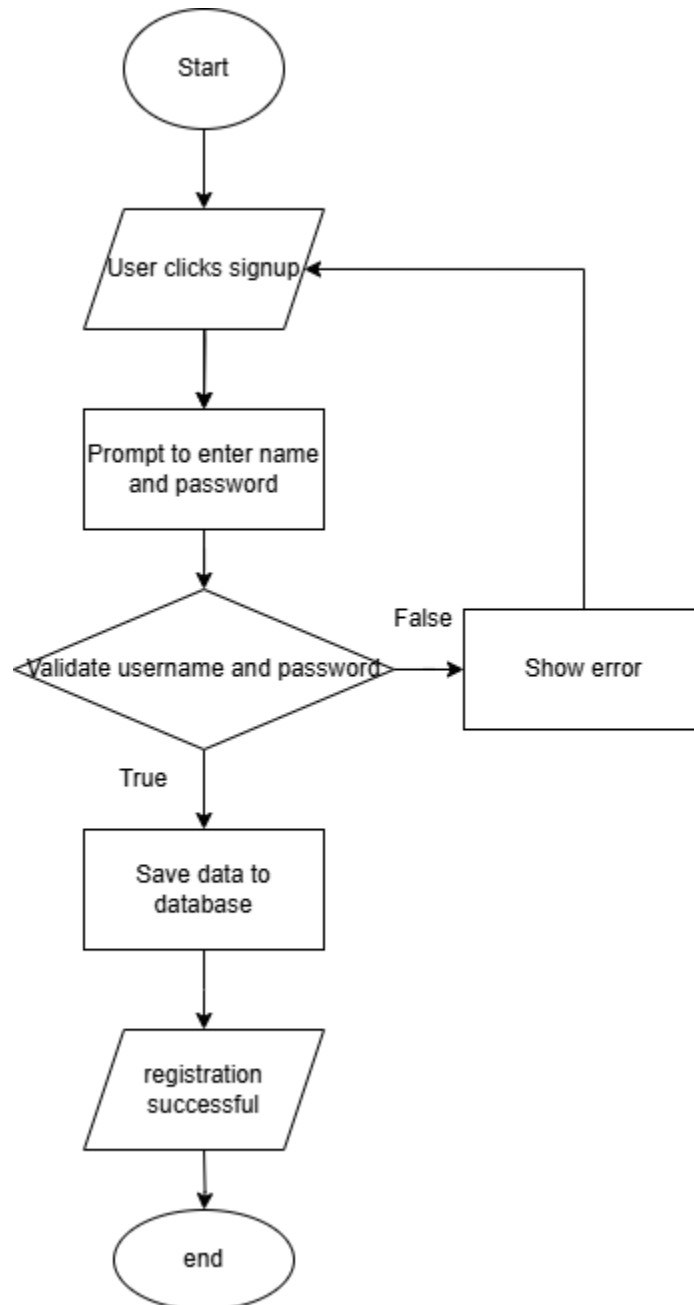
### **3.1.4 Create Membership**



Music creator can create their own membership by stating what methods they're using to accept their payment and what is the subscription price.

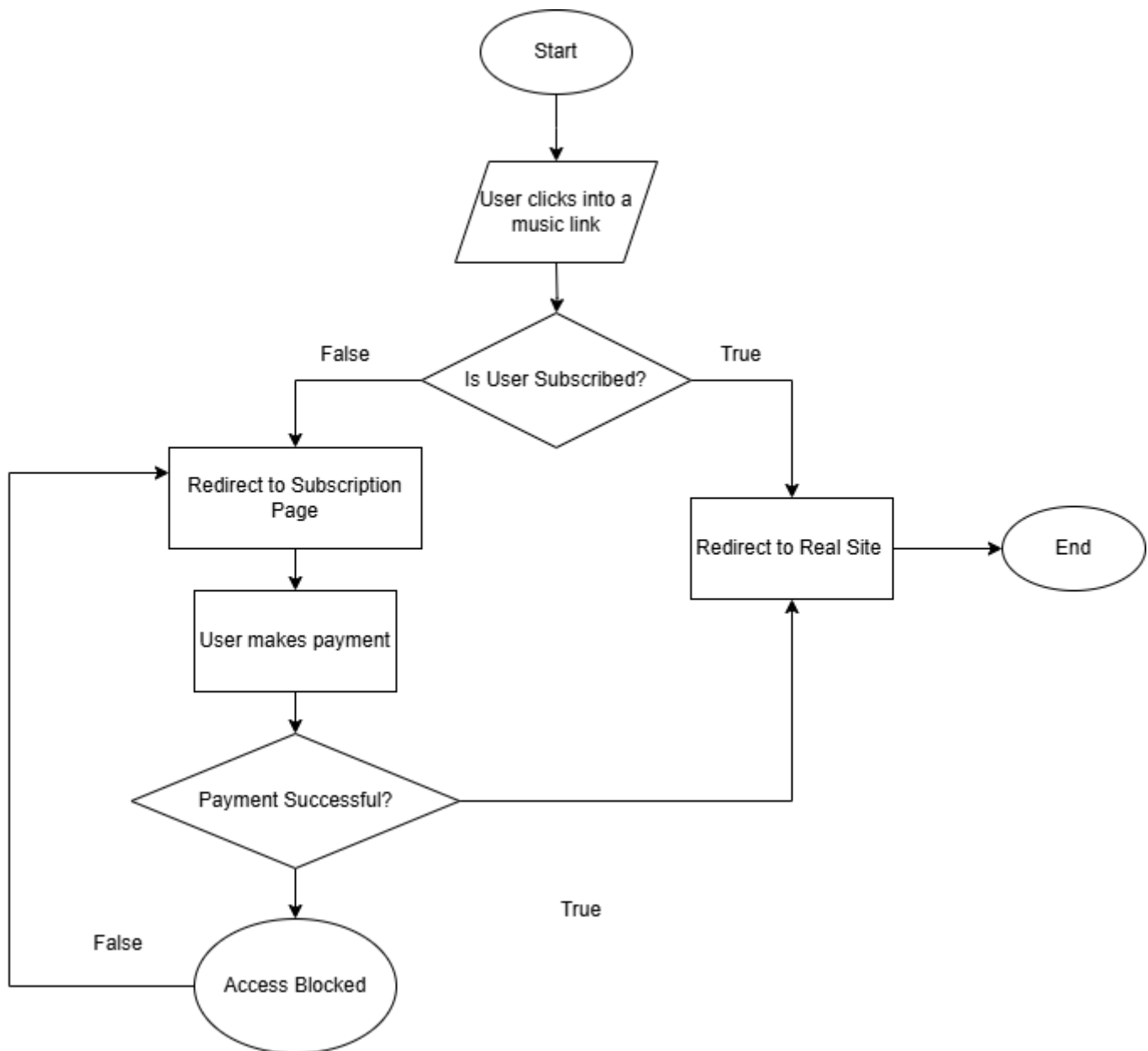
## 3.2 User

### 3.2.1 Create Account



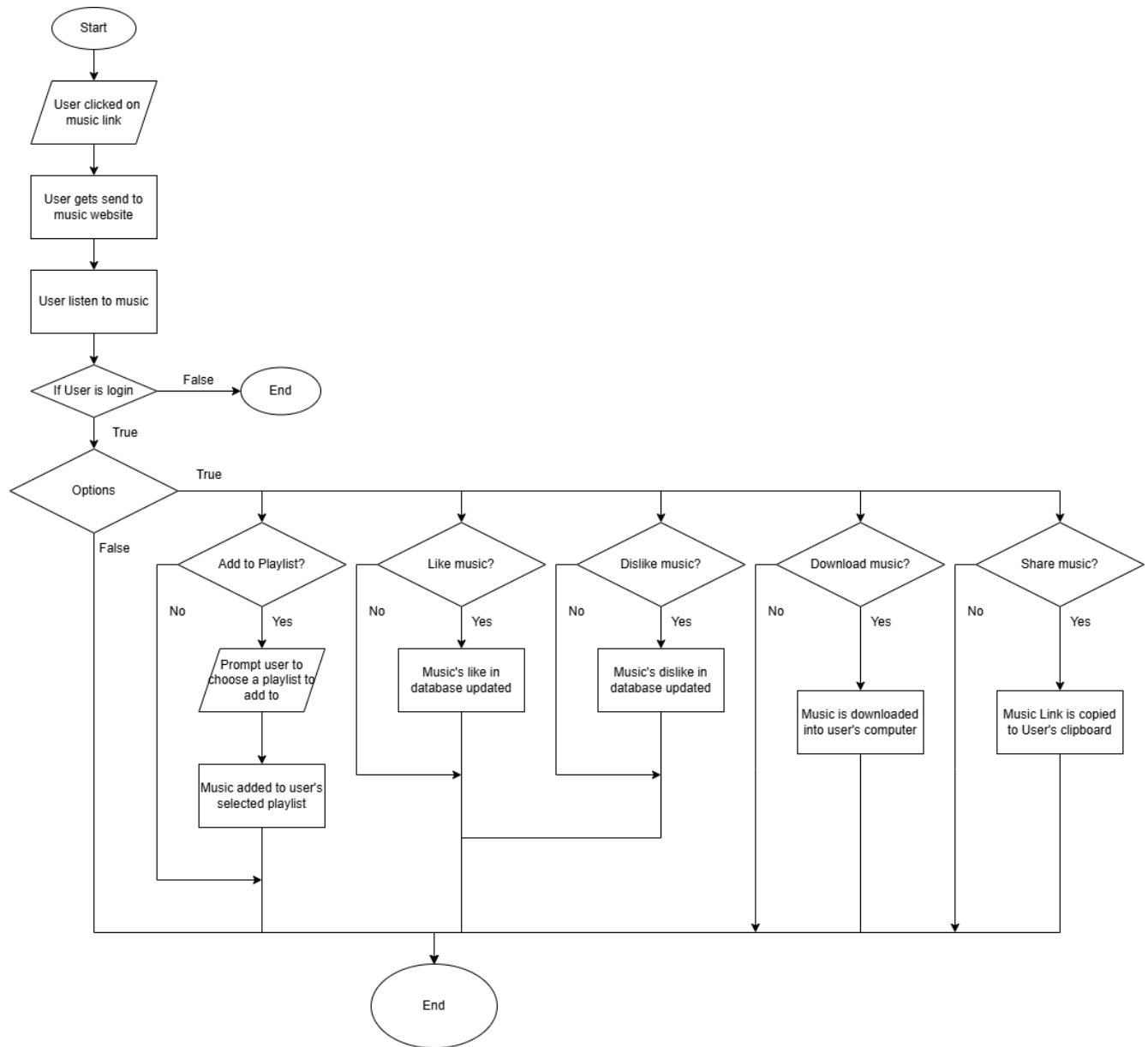
The user is required to enter their name and password. The system will validate if name is available and if both pass the requirements. If yes, both the name and the password would be entered into the system database.

### 3.2.2 Buy Membership



When the user clicks into a music page that requires a membership. The system will prompt the user to subscribe to the music creator to be able to view the music. The user is then required to make a payment to be able to access all the music creator's song. Failure to do so will be blocked of access.

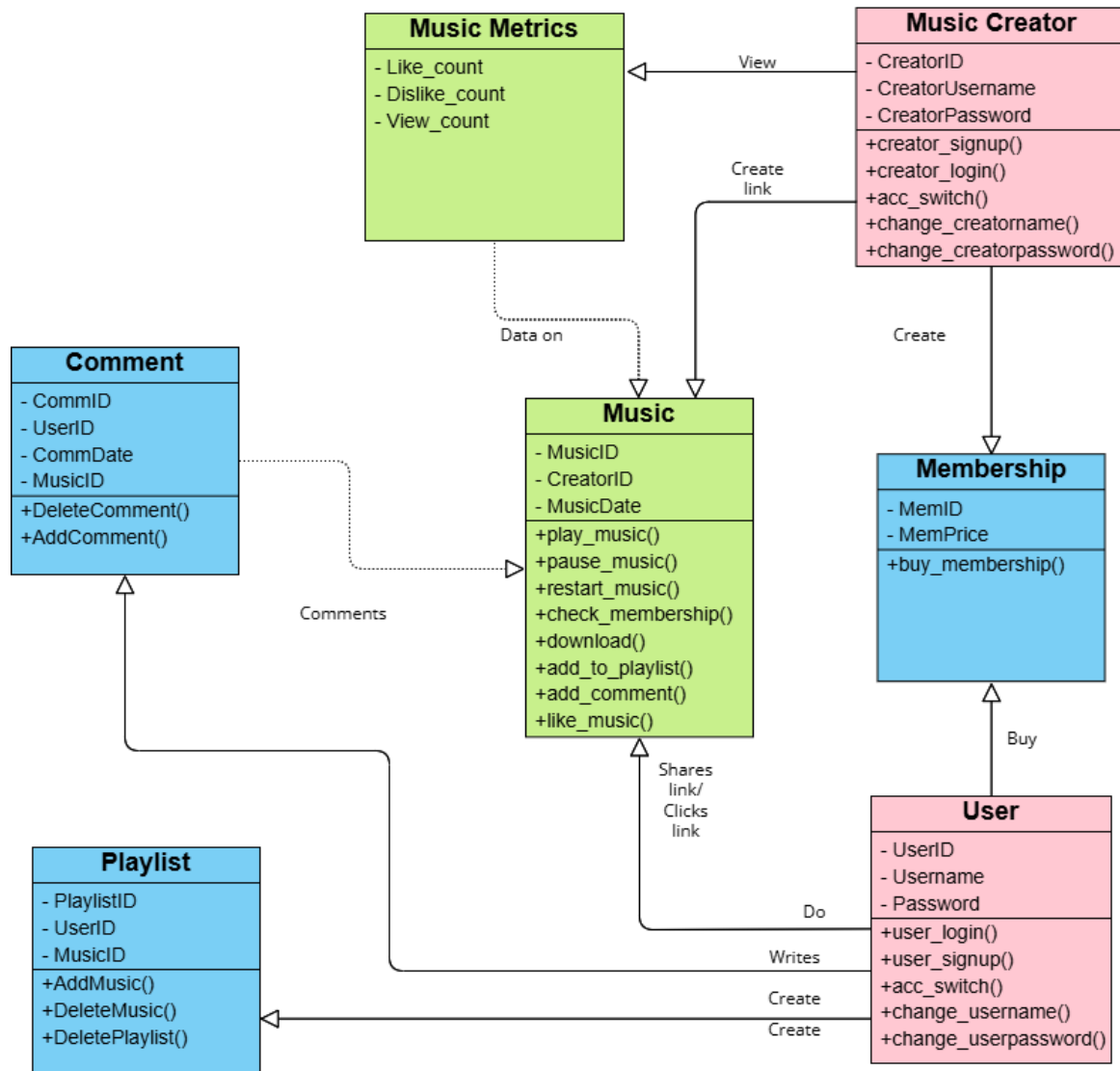
### 3.2.3 Interact with music



When the user clicks into a music link. The user is presented with a page with multiple options, the user is able to play, pause and replay the music, in addition, the user is able to add the music to playlist, like, dislike the music , download and share the music. Each of this options will update the music's database or provide the corresponding music/music link to the user.

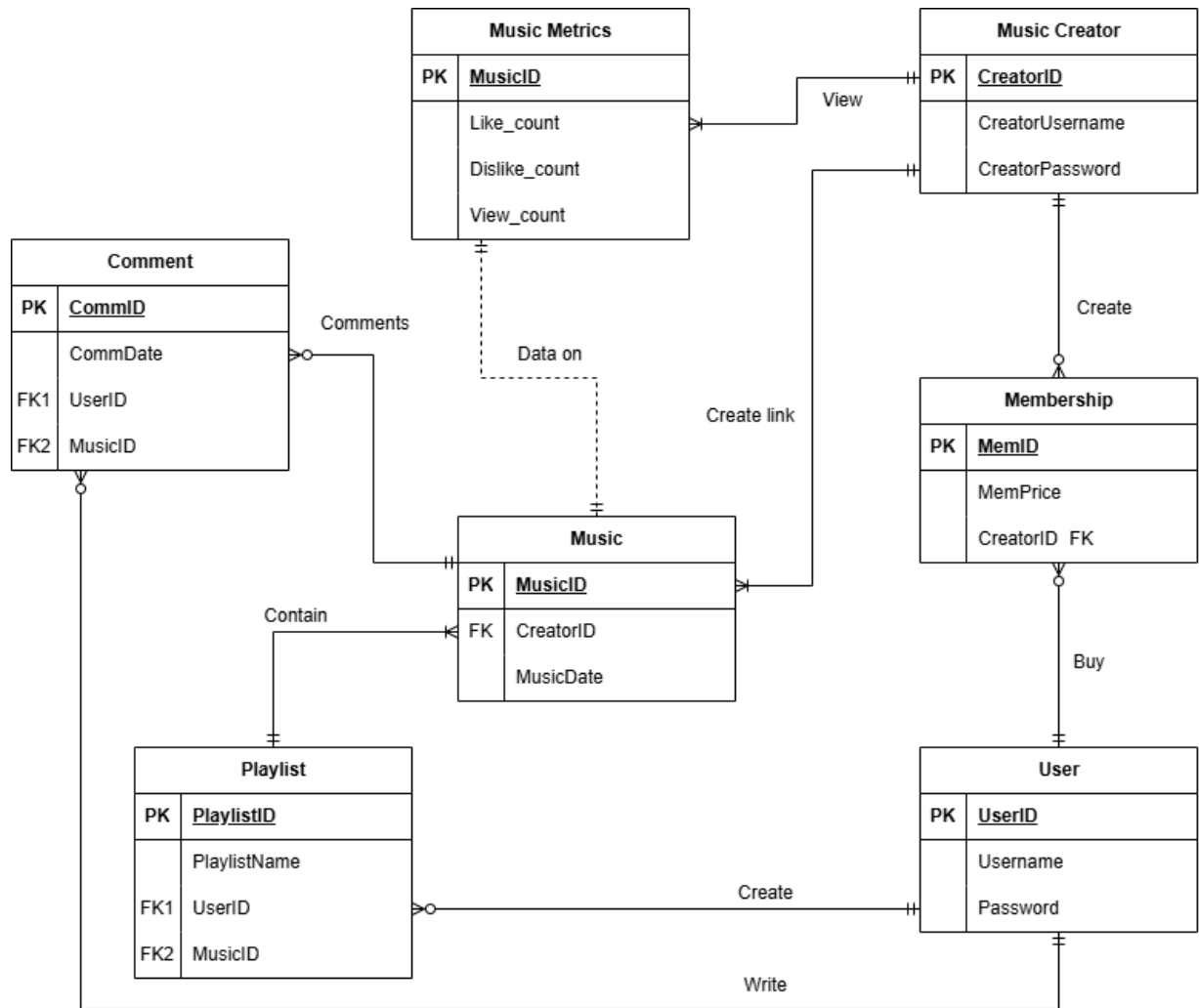
## 4 Requirements Modeling

### 4.1 Class Diagrams / ERD



Class diagram of MusicHub





ERD Diagram of MusicHub's Database

## 4.2 Classes / Entities

Classes/Entities	Description
Music Creator	<ul style="list-style-type: none"><li>• <b>Attributes:</b><ul style="list-style-type: none"><li>◦ <b>CreatorID</b>: Unique identifier for the music creator.</li><li>◦ <b>CreatorUsername</b>: The username of the music creator.</li><li>◦ <b>CreatorPassword</b>: The password of the music creator.</li></ul></li><li>• <b>Methods:</b><ul style="list-style-type: none"><li>◦ <b>creator_signup()</b>: Allows the creator to sign up for the platform.</li><li>◦ <b>creator_login()</b>: Allows the creator to log in to their account.</li><li>◦ <b>acc_switch()</b>: Enables switching between accounts.</li><li>◦ <b>change_creatormname()</b>: Allows the music creator to change their username.</li><li>◦ <b>change_creatorpassword()</b>: Allows the music creator to change their password.</li></ul></li></ul>
User	<ul style="list-style-type: none"><li>• <b>Attributes:</b><ul style="list-style-type: none"><li>◦ <b>UserID</b>: Unique identifier for the user.</li><li>◦ <b>Username</b>: The username of the user.</li><li>◦ <b>Password</b>: The password of the user.</li></ul></li><li>• <b>Methods:</b><ul style="list-style-type: none"><li>◦ <b>user_login()</b>: Allows the user to log in.</li><li>◦ <b>user_signup()</b>: Allows the user to sign up.</li><li>◦ <b>acc_switch()</b>: Enables switching between accounts.</li><li>◦ <b>change_username()</b>: Allows the user to change their username.</li><li>◦ <b>change_userpassword()</b>: Allows the user to change their password.</li></ul></li></ul>

<b>Music</b>	<ul style="list-style-type: none"> <li>• <b>Attributes:</b> <ul style="list-style-type: none"> <li>◦ <b>MusicID:</b> Unique identifier for the music track.</li> <li>◦ <b>CreatorID:</b> ID of the music creator who uploaded the music.</li> <li>◦ <b>MusicDate:</b> The date the music was uploaded.</li> </ul> </li> <li>• <b>Methods:</b> <ul style="list-style-type: none"> <li>◦ <b>play_music():</b> Plays the music track.</li> <li>◦ <b>pause_music():</b> Pauses the music track.</li> <li>◦ <b>restart_music():</b> Restarts the music playback.</li> <li>◦ <b>check_membership():</b> Checks if the user has a membership to access premium content.</li> <li>◦ <b>download():</b> Allows users to download the music. (requires login)</li> <li>◦ <b>add_to_playlist():</b> Adds the track to a playlist. (requires login)</li> <li>◦ <b>add_comment():</b> Enables users to add comments to the music. (requires login)</li> <li>◦ <b>like_music():</b> Allows users to like/dislike the track. (requires login)</li> </ul> </li> </ul>
<b>Music Metrics</b>	<ul style="list-style-type: none"> <li>• <b>Attributes:</b> <ul style="list-style-type: none"> <li>◦ <b>Like_count:</b> Total number of likes for a track.</li> <li>◦ <b>Dislike_count:</b> Total number of dislikes for a track.</li> <li>◦ <b>View_count:</b> Total number of views for a track.</li> </ul> </li> </ul>
<b>Comment</b>	<ul style="list-style-type: none"> <li>• <b>Attributes:</b> <ul style="list-style-type: none"> <li>◦ <b>CommID:</b> Unique identifier for a comment.</li> <li>◦ <b>UserID:</b> ID of the user who made the comment.</li> <li>◦ <b>CommDate:</b> The date the comment was posted.</li> <li>◦ <b>MusicID:</b> The ID of the music being commented on.</li> </ul> </li> <li>• <b>Methods:</b> <ul style="list-style-type: none"> <li>◦ <b>DeleteComment():</b> Deletes a comment.</li> <li>◦ <b>AddComment():</b> Adds a new comment.</li> </ul> </li> </ul>

<b>Playlist</b>	<ul style="list-style-type: none"><li>• <b>Attributes:</b><ul style="list-style-type: none"><li>○ <b>PlaylistID:</b> Unique identifier for a playlist.</li><li>○ <b>UserID:</b> ID of the user who created the playlist.</li><li>○ <b>MusicID:</b> The ID of the music track included in the playlist.</li></ul></li><li>• <b>Methods:</b><ul style="list-style-type: none"><li>○ <b>AddMusic():</b> Adds a music to the playlist.</li><li>○ <b>DeleteMusic():</b> Removes a music from the playlist.</li><li>○ <b>DeletePlaylist():</b> Deletes the entire playlist.</li></ul></li></ul>
<b>Membership</b>	<ul style="list-style-type: none"><li>• <b>Attributes:</b><ul style="list-style-type: none"><li>○ <b>MemID:</b> Unique identifier for the membership.</li><li>○ <b>MemPrice:</b> The price of the membership.</li></ul></li><li>• <b>Methods:</b><ul style="list-style-type: none"><li>○ <b>buy_membership():</b> Allows the user to purchase a membership.</li></ul></li></ul>

## 5 Behavioral & Flow Modeling

### 5.1 Sequence Diagrams

#### 5.1.1 User

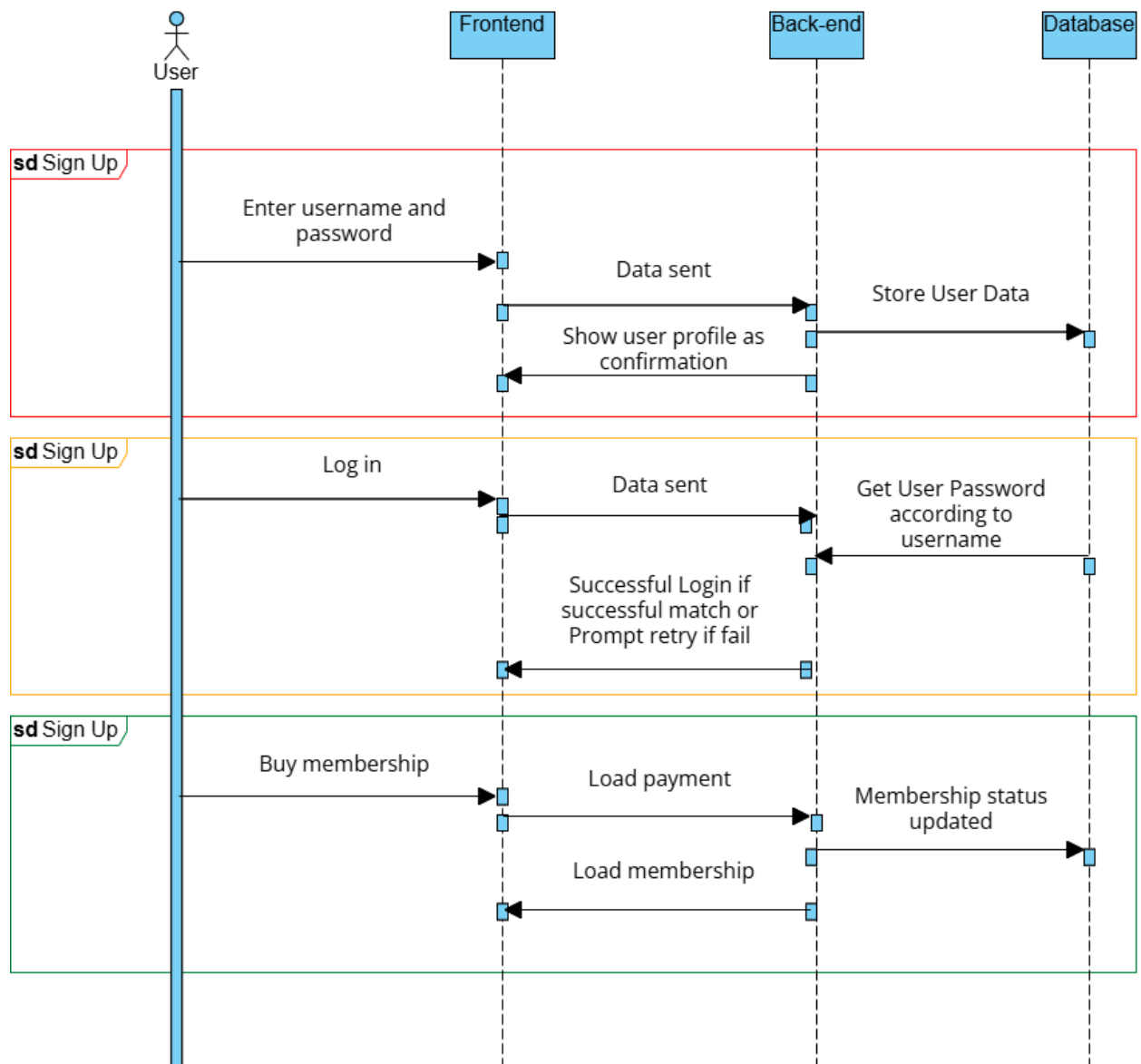
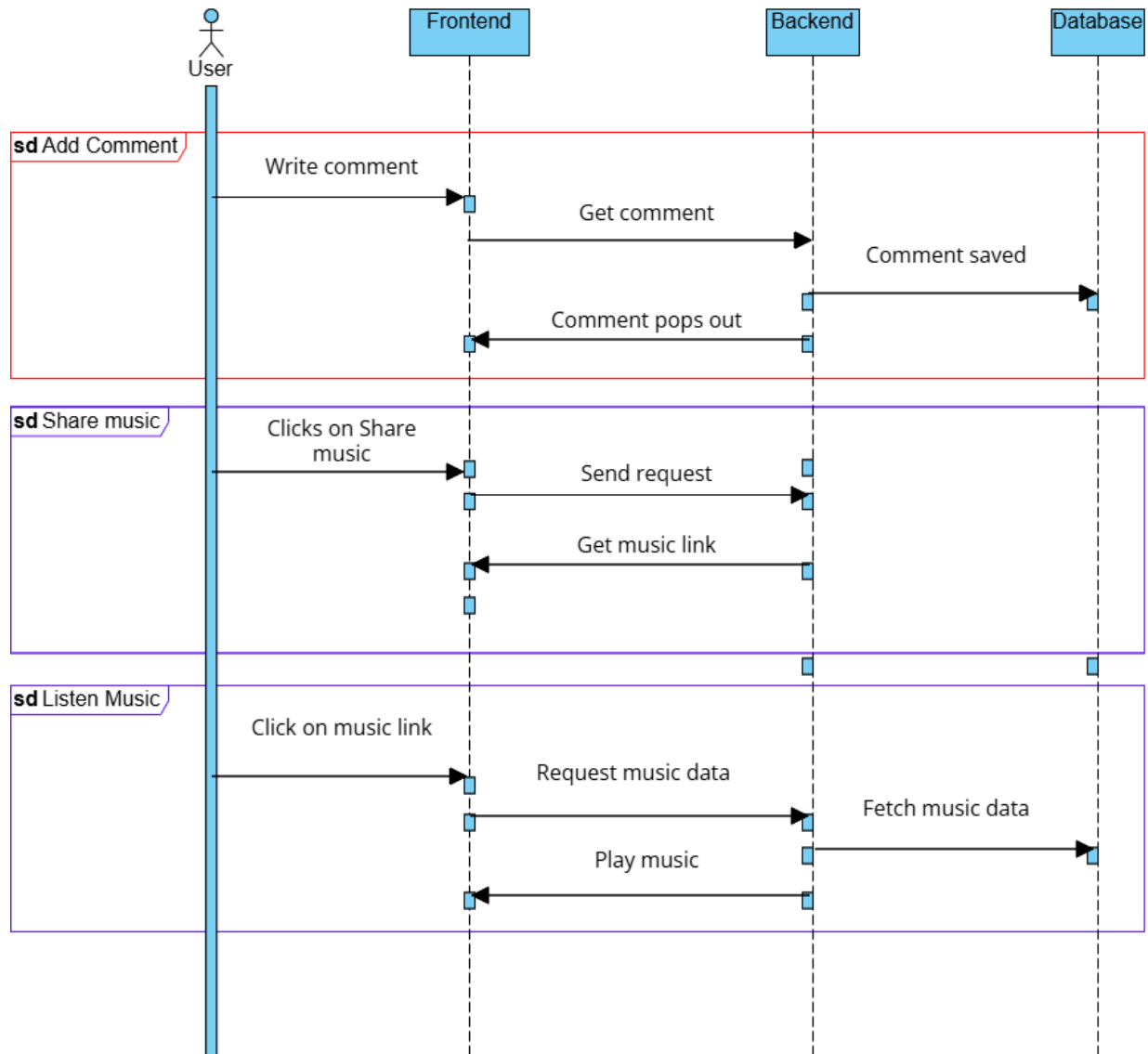


Diagram 5.1.1.1 User Sequence Diagram Part 1



*Diagram 5.1.1.2 User Sequence Diagram Part 2*

## 5.1.2 Music Creator

On top of the following functions of user, Music Creators also have access to the following functions.

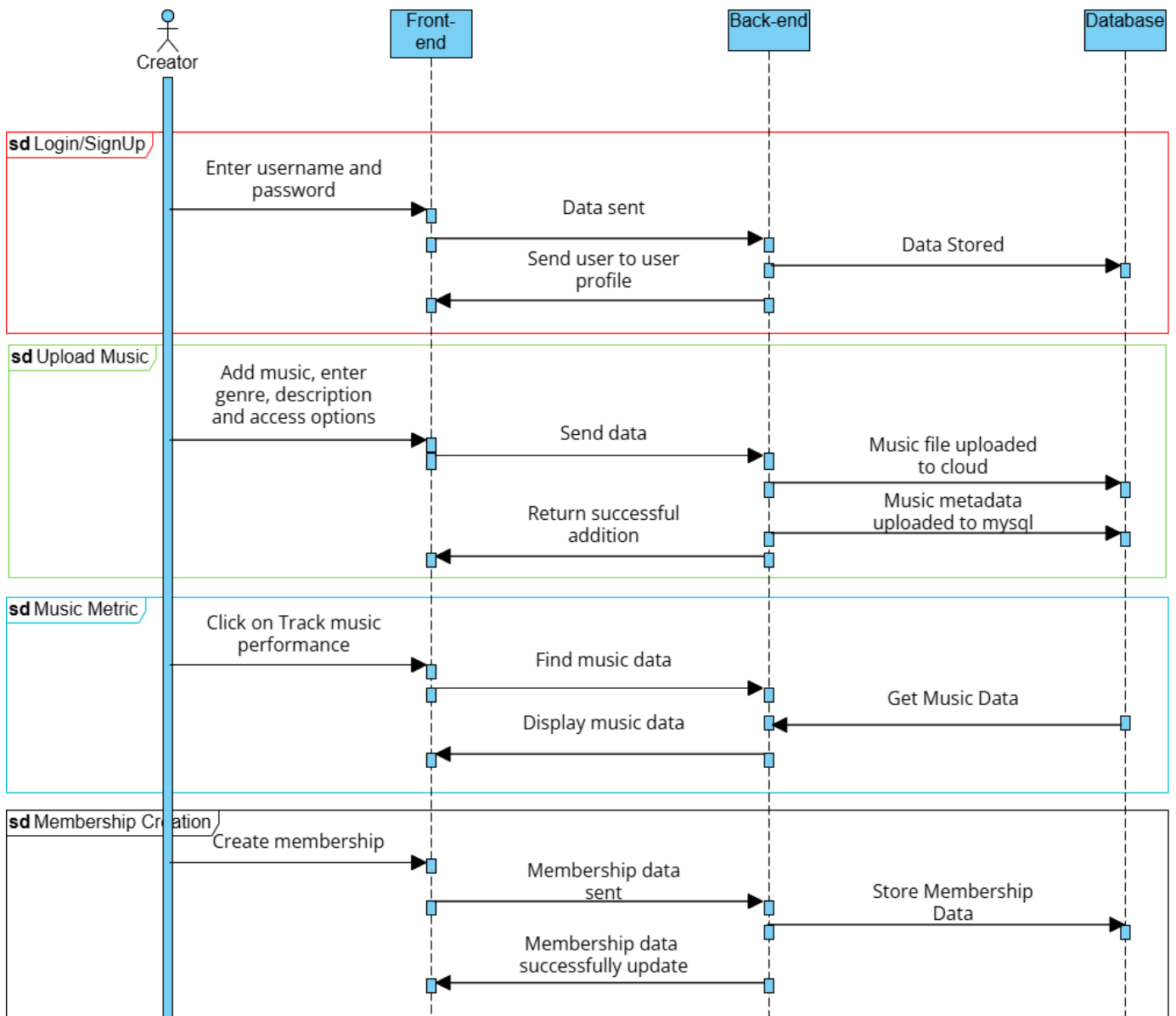
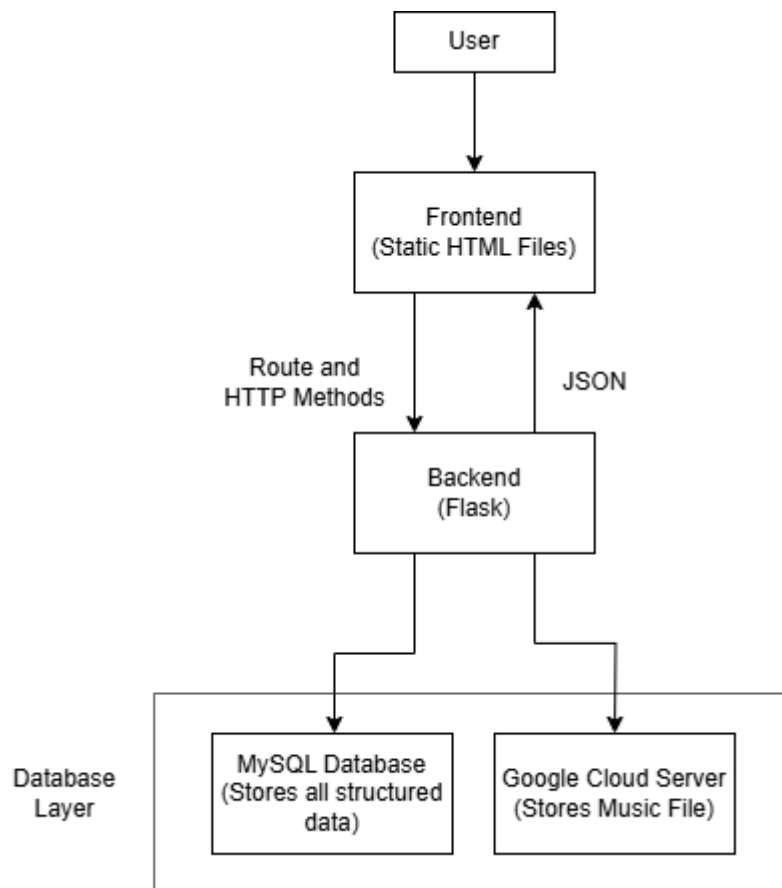


Diagram 5.1.2.1 Music Creator Sequence Diagram

## 6 Architecture Design

### 6.1 Software Architecture

The MusicHub is designed to allow Creators to upload music in a seamless and user-friendly environment to allow users to share it to the community and to allow users to be able to listen to music in a hassle free environment. It provides an intuitive interface for users to browse music and enable music creators to create playlists, and share it on the platform. The system is built with a modular architecture, split into several subsystems that cater to different aspects of the app's functionality.



*A brief look into MusicHub's Architecture*



### **6.1.1 FrontEnd**

The frontend subsystem is responsible for delivering an interactive user interface. It handles the presentation layer and ensures that users can easily upload, play, and share music without need of any guides or help.

#### **Functionality:**

- Provides an interactive user-friendly interface for creators and listeners.
- Handles user interactions such as music uploads, playback, playlist management, and sharing.

#### **Implementations:**

- Github pages
- HTML Static Pages
- CSS
- Some flask functionalities to modify data

#### **Team Responsibilities:**

- Implement intuitive easy to understand UI/UX design.
- Develop suitable button and interface for music upload, playlist creation, and music playback.

### **6.1.2 BackEnd**

The backend subsystem manages business logic, processing user requests, and ensuring seamless interaction with the database. It serves as the intermediary between the frontend and the database.

#### **Functionality:**

- Manages core business logic.
- Processes user requests and mediates between the frontend and the database.
- Manages membership purchase and validation
- Ensures secure authentication and authorization.
- Handles media streaming and playlist operations.

#### **Implementations**

- APIs: RESTful and Fetch API
- Develop and maintain APIs.
- Implement business logic for user actions and content management.

#### **Team Responsibilities:**

- Develop and maintain APIs.
- Implement business logic for user actions and content management.

### **6.1.3 Database**

The database subsystem stores and manages the music files, user data, playlists, and analytics. In our project, we divide it into two parts: structured data is stored in MySQL while music files are stored in Google Cloud.

- **Functionality:**
  - Stores and manages music files, user data, playlists, and analytics.
  - Provides efficient data retrieval and storage mechanisms.
- **Technologies:**
  - Music File Storage: cloud storage services
  - User, Analytics, Music Metadata, Genre Type Data: MySQL with SQLAlchemy
- **Team Responsibilities:**
  - Design and optimize database schemas.
  - Ensure data integrity and implement backup mechanisms.
  - Handle large-scale data storage and retrieval efficiently.

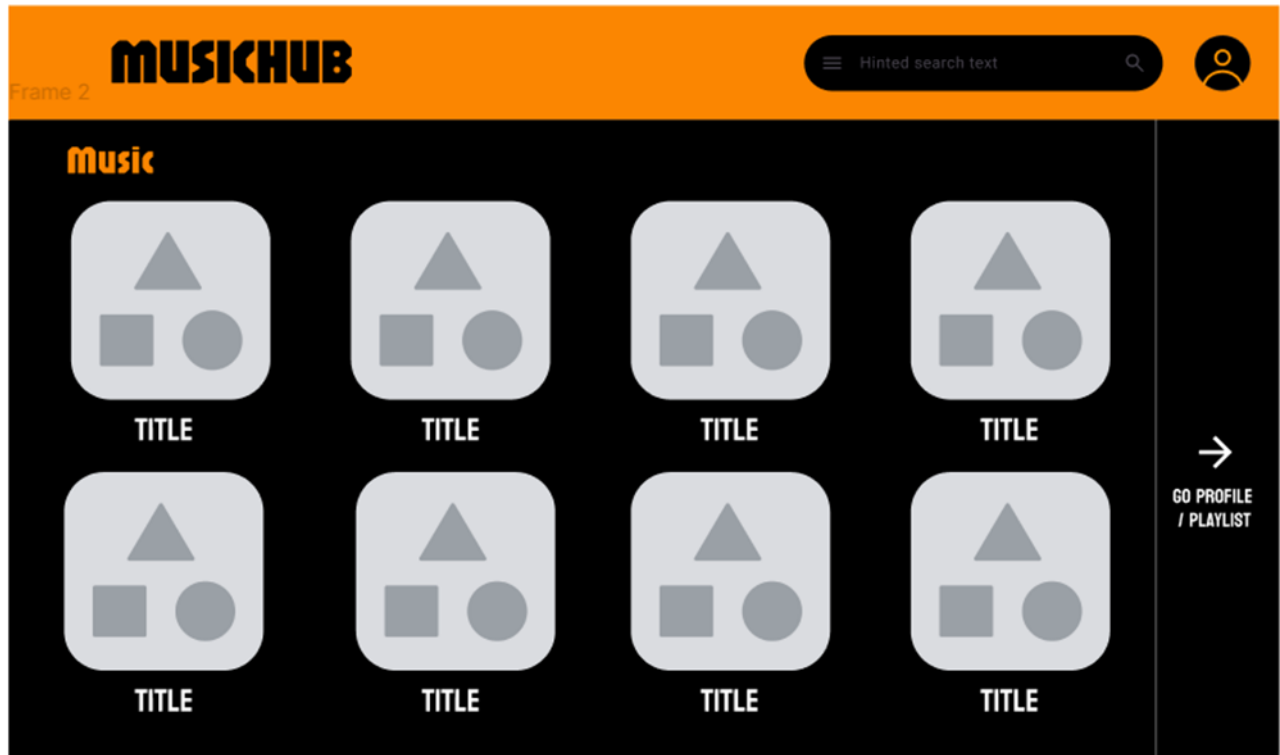
### **6.1.4 Subsystem Distribution**

The system development is divided among four teams of students, each responsible for a specific subsystem:

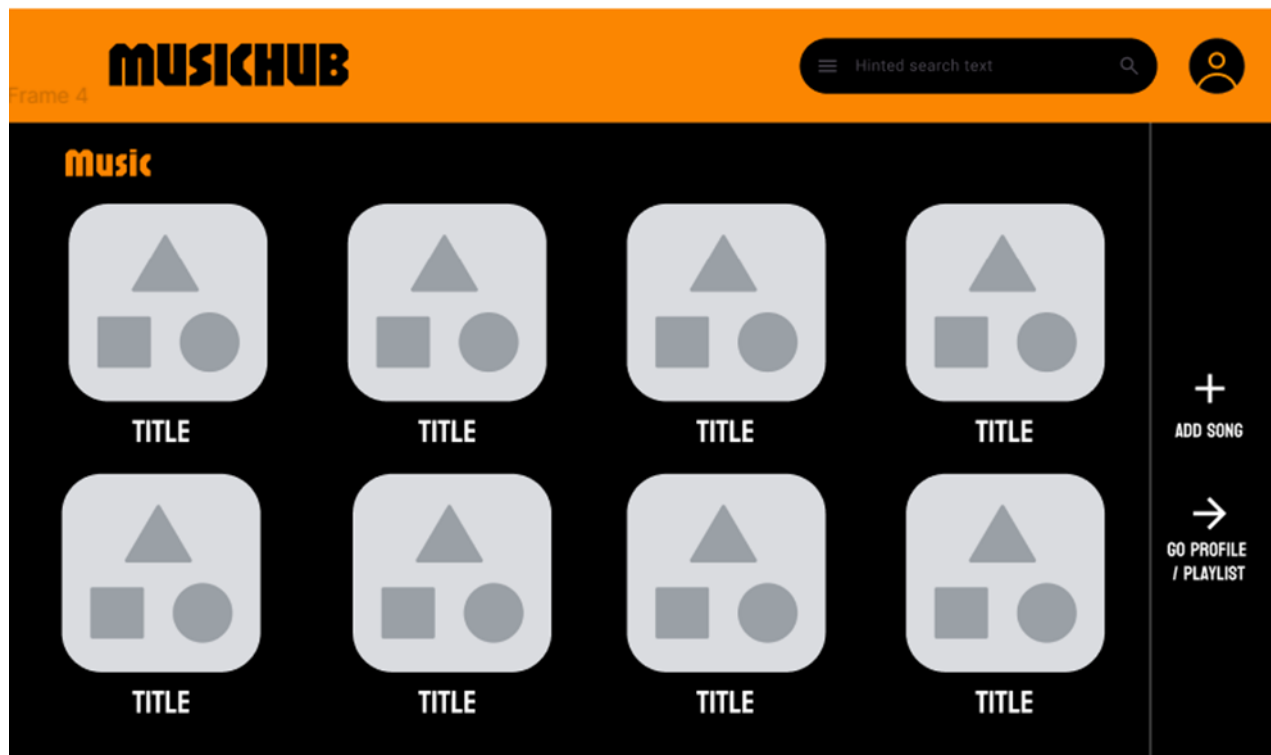
- **Frontend Team:** Responsible for UI/UX design, implementing user interactions, and ensuring responsiveness.
  - Team Members: Ryan , Chee Rui
- **Backend Team:** handles API requests, processes page logic, and communicates with the data storage layer
  - Team Members: Chee Rui , Mei Yong Peng
- **Database Team:** Designs and optimizes data storage, manages user and music data, and ensures database integrity.
  - Team Members: Chan Ka Ken , Ryan

## 7 Interface Design

### 7.1 Main Screens

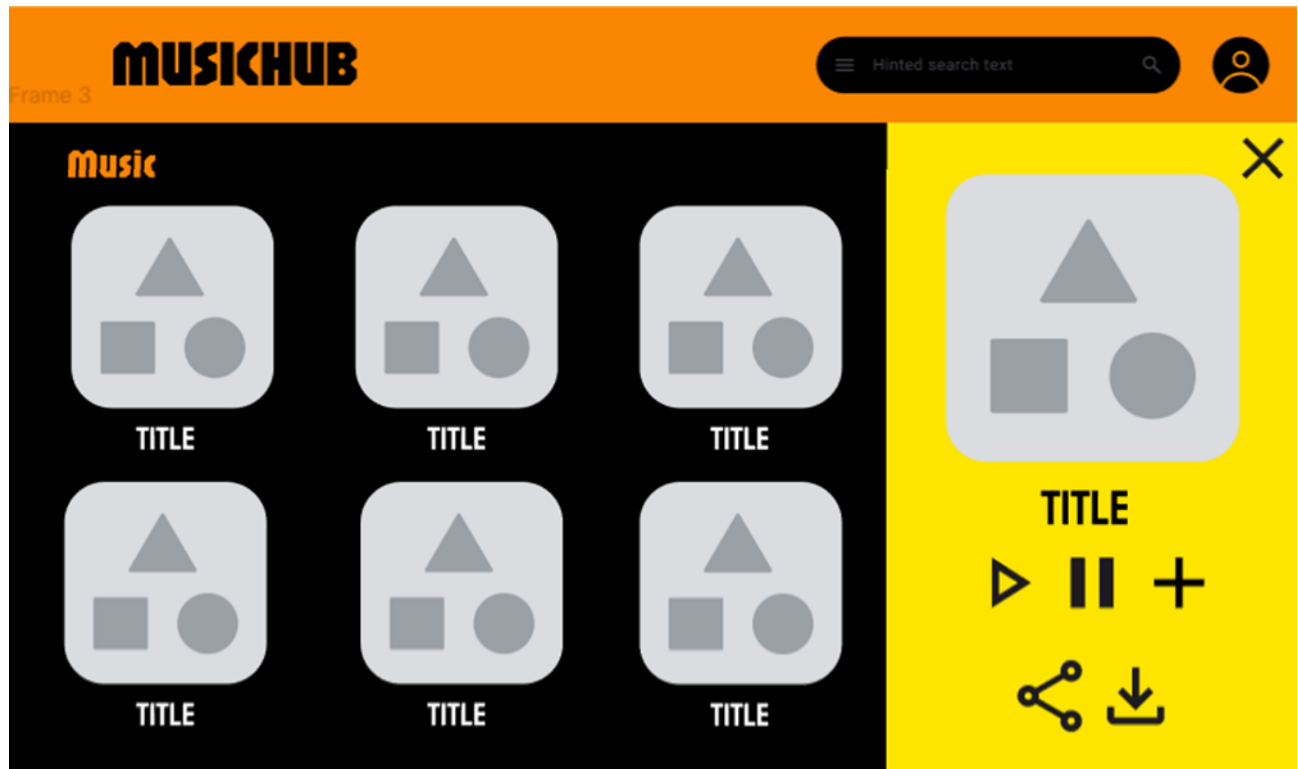


This is the main screen for User. There are 8 songs that will be recommended to the user each time they open. The right bar which is the toolbar, consist only 1 tool and that is the directory to Profile and Playlist Page.

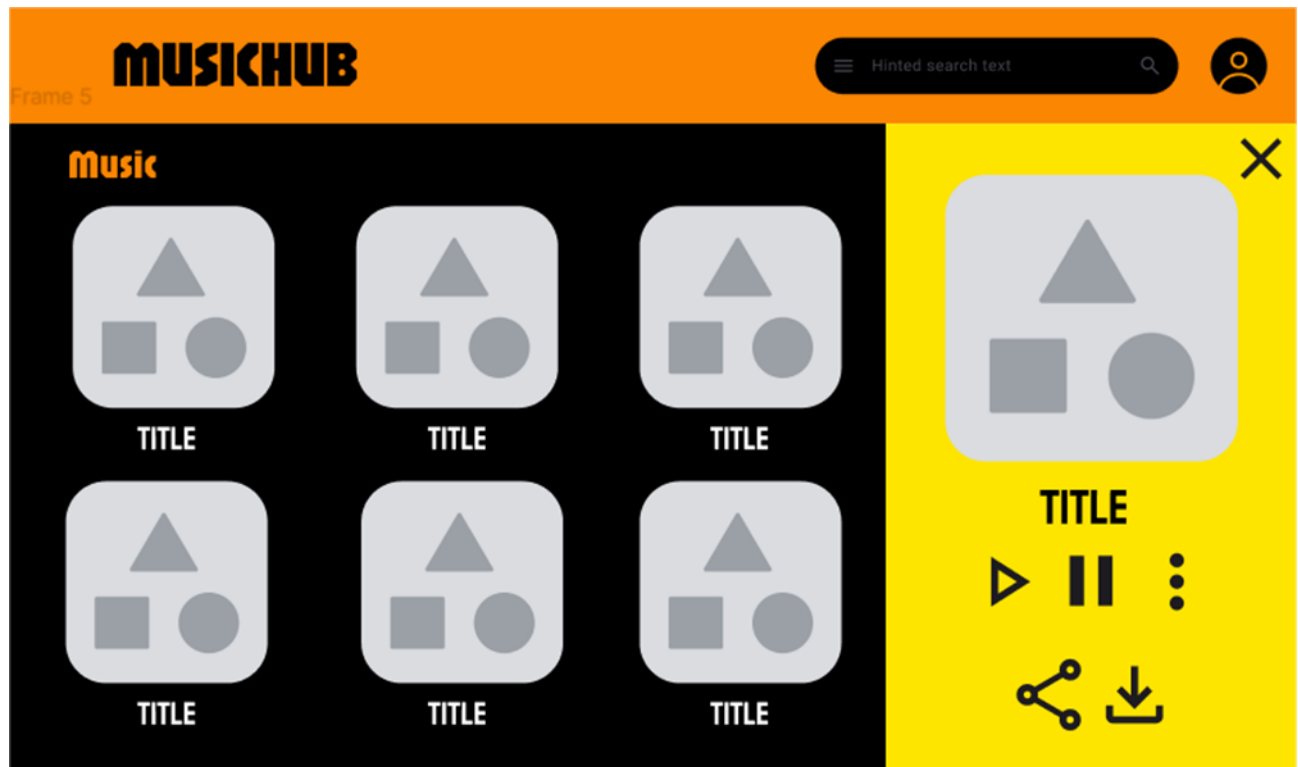


This is the main screen for Music Creator. There are 8 songs that will be recommended to the user each time they open. The right bar which is the toolbar, consists of the directory to Profile and Playlist and the Add Song feature.

## 7.2 Playing Music Screen



This is the music screen for User. There will be 6 songs that still will be recommended to the user each time they open. The right section is music the User has chosen to listen. Within the section, User can play and pause the music, add music to the playlist, share music and download music.

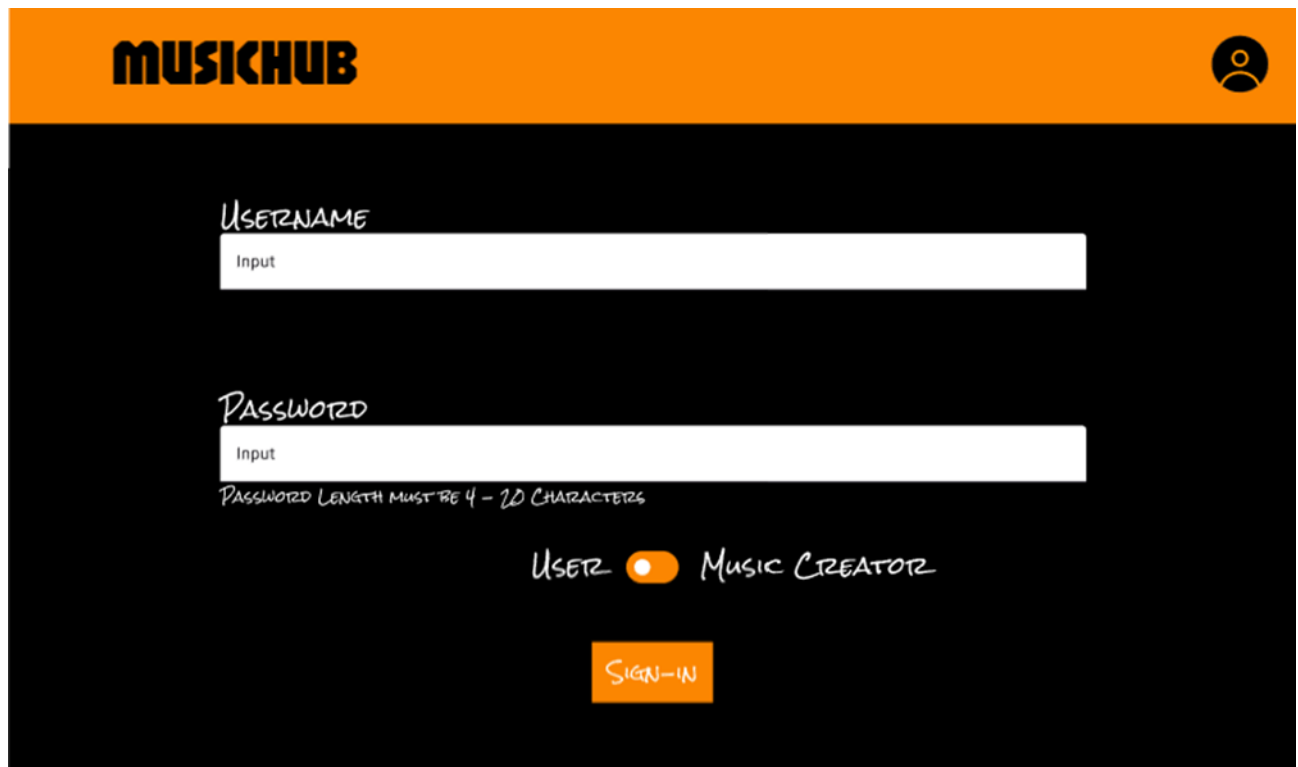


This is the music screen for Music Creator. The feature is mostly the same as the User's version. The only difference is instead of adding music to the playlist, it is changed to the directory to music analytics.

### 7.3 Login / Sign-in Screen

The mockup shows a login interface for 'MUSICHUB'. It features a dark background with an orange header bar containing the 'MUSICHUB' logo and a user profile icon. The login form includes two input fields: 'USERNAME' and 'PASSWORD', both with 'Input' placeholder text. A 'Forgot Password?' link is positioned to the right of the password field. Below the fields is a toggle switch labeled 'USER' and 'MUSIC CREATOR', with the 'MUSIC CREATOR' option currently selected. At the bottom, there are two orange buttons: 'LOGIN' and 'SIGN-IN'. To the right of the 'SIGN-IN' button, there is a note: '<<< No Account? Click This'.

This is the Login page. User and Music Creator can choose what kind of account they possesses and they have to key-in their username and password. If a person doesn't have an account, they can press the Sign-in button to create their account.

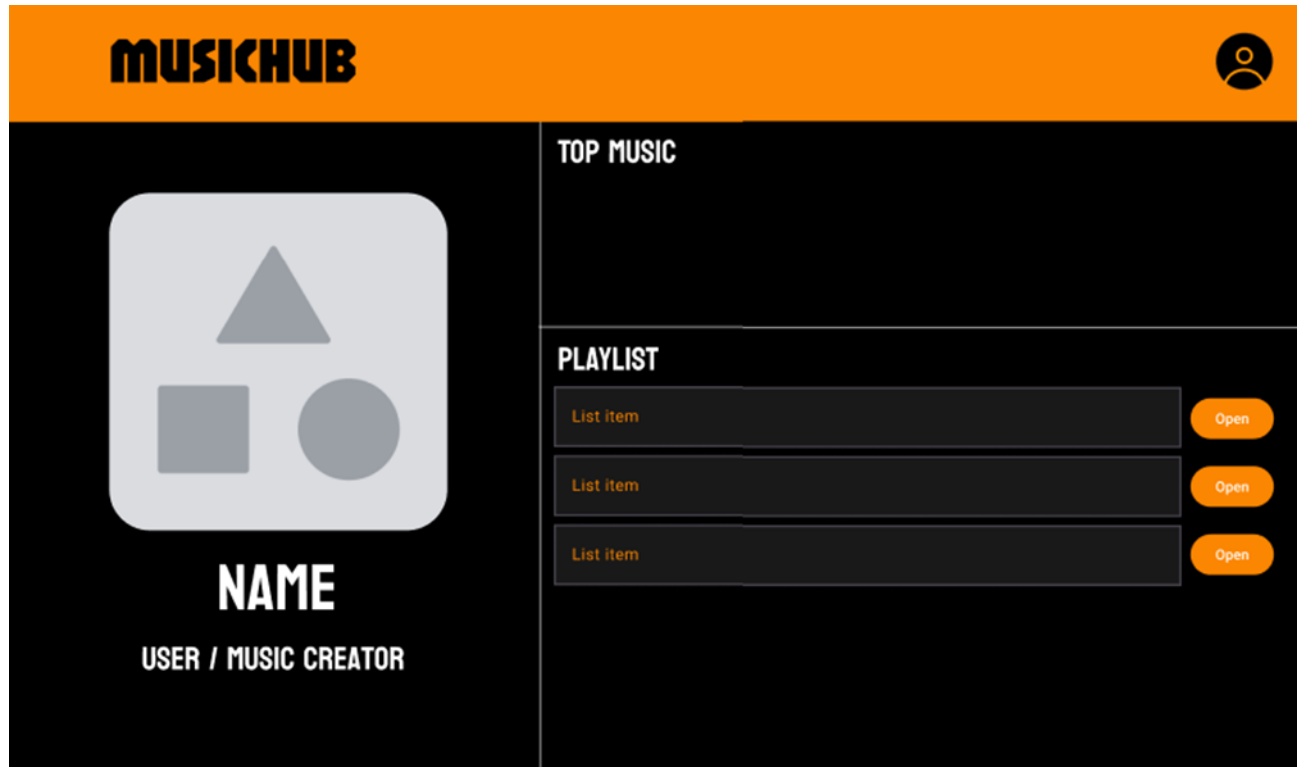


The image shows a sign-in page for a music sharing system. The page has a dark background with an orange header bar. The header bar contains the 'MUSICHUB' logo on the left and a user profile icon on the right. The main content area is dark and contains two white input fields. The first field is labeled 'USERNAME' and has a placeholder text 'Input'. The second field is labeled 'PASSWORD' and has a placeholder text 'Input'. Below the password field, there is a note: 'PASSWORD LENGTH MUST BE 4 - 20 CHARACTERS'. Below the input fields, there is a toggle switch labeled 'USER' and 'MUSIC CREATOR'. The 'USER' option is selected, indicated by a white dot. Below the toggle switch, there is an orange button labeled 'SIGN-IN'.

This is the Sign-in page. Anyone without an account can create an account here by key-in their desired username and password. The password should be at least 4 - 20 characters. Then they can choose what kind of account they want. After that, their account will be created after all requirement met and sign-in button is pressed.

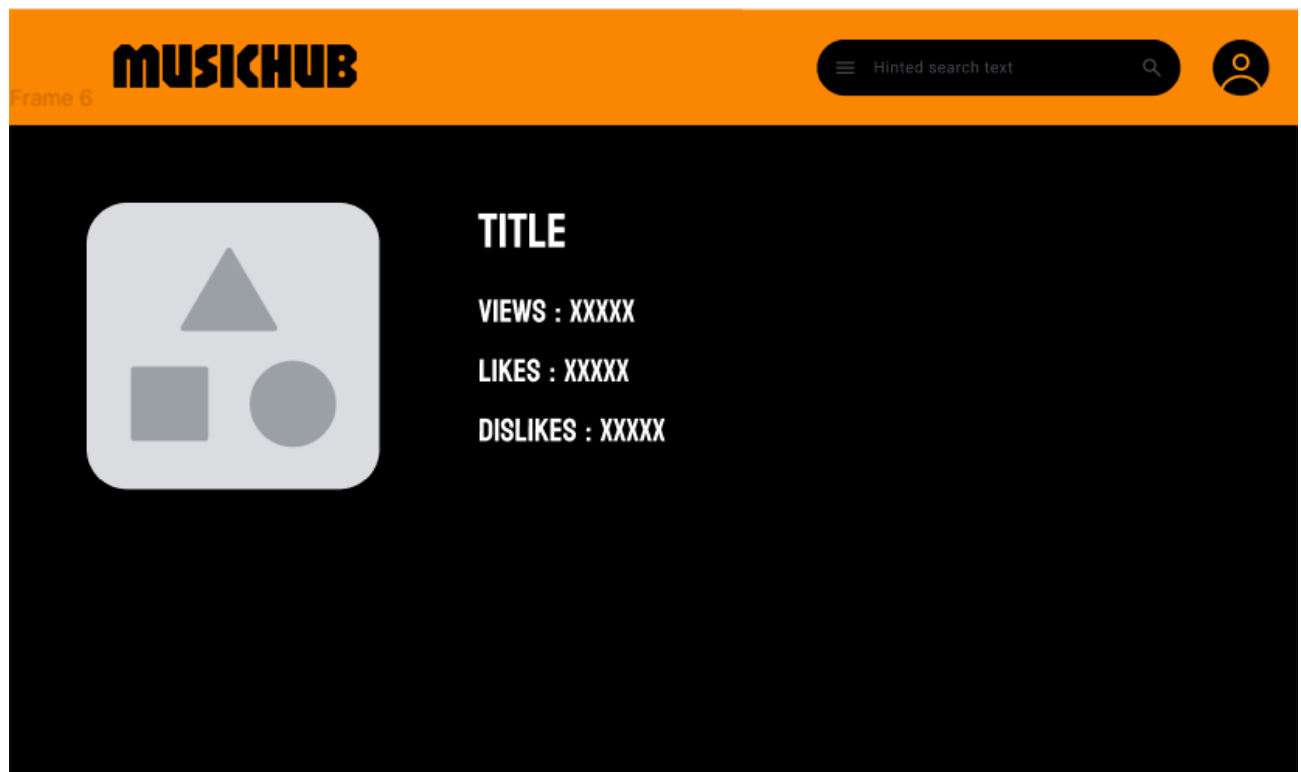


## 7.4 Profile / Playlist Screen



This is the Profile and Playlist page. It will display the User or Music Creator status and their name. If the person is a Music Creator, the top music will display their best played music. The playlist can be created by pressing the playlist word and playlist can be edited by pressing into the playlist button.

## 7.5 Music Analytic Screen



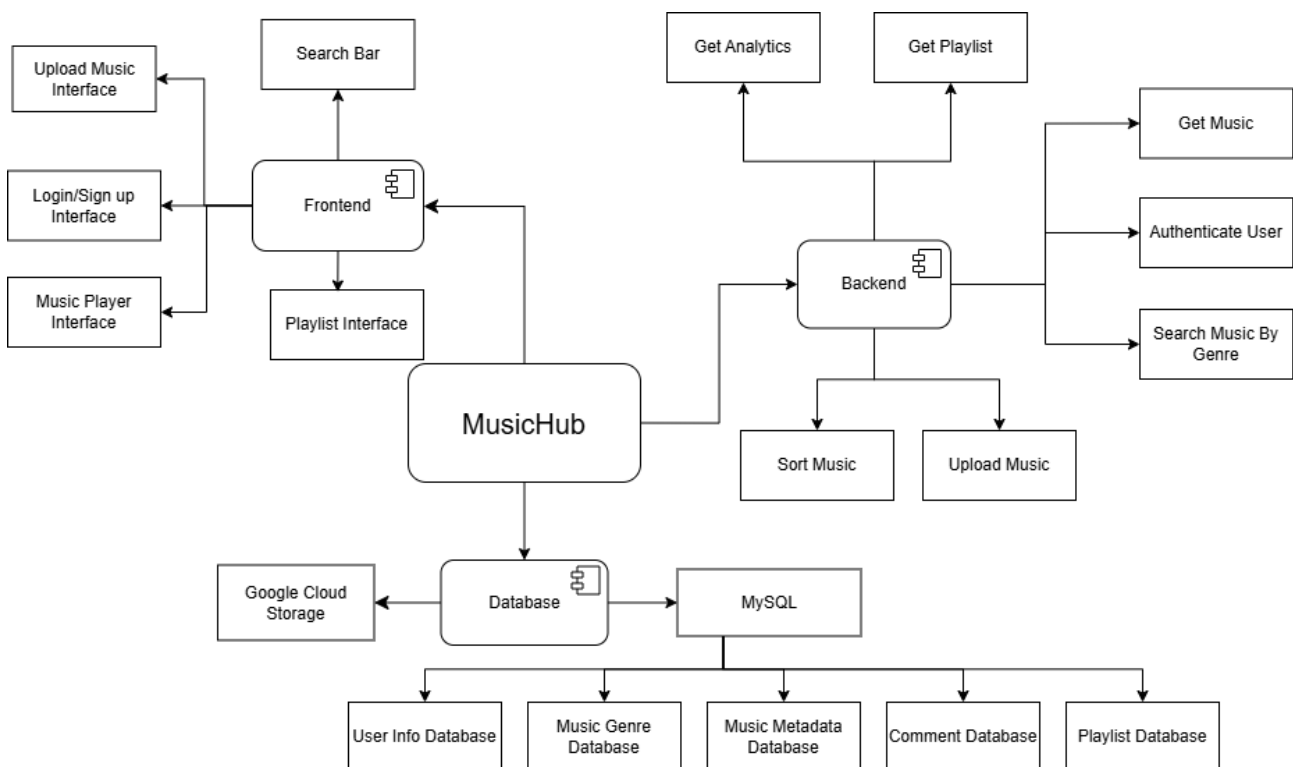
This is the Music Analytic page. It only display the music picture, title, view count, like count and dislike count.

## 8 Component Design

### 8.1 Main Layers

MusicHub's design is centered around three main layer which is then further broken down into components:

1. User Interface Layer (**Frontend**): Handles user interactions through website visual modules such as login/signup, music upload, a statistics dashboard, playlist management, and an audio player.
2. Application Layer (**Backend**): Manages what happens after user interactions, how data will be transferred between frontend and database and simple calculations.
3. Data Storage Layer (**Database**): Ensures secure and efficient storage of user accounts, music metadata, performance metrics, playlists and sharing links.

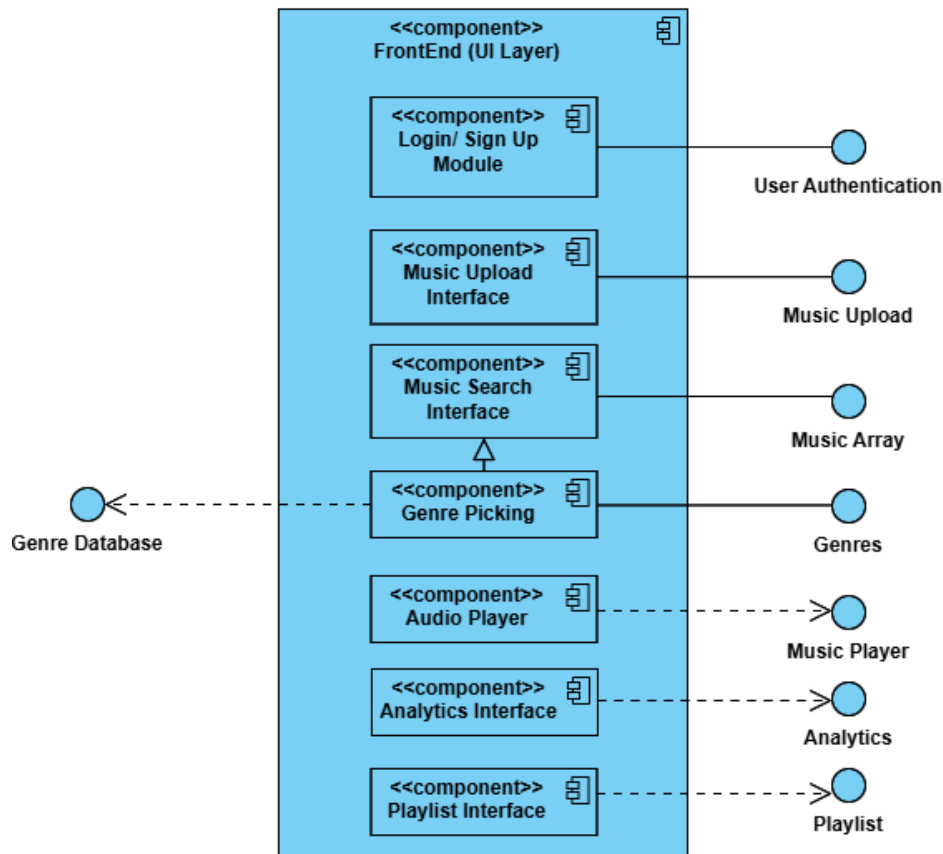


#### 8.1.1 Breakdown of Systems of MusicHub

### 8.1.1 Frontend (UI layer)

Our Frontend is composed of using Flask which implements it using HTML, CSS and Javascript with some flask functionalities which helps us to modify the HTML easily through the backend.

This layer focuses on delivering an intuitive and user-friendly experience to users visually. It ensures seamless interaction with the application's backend and database.

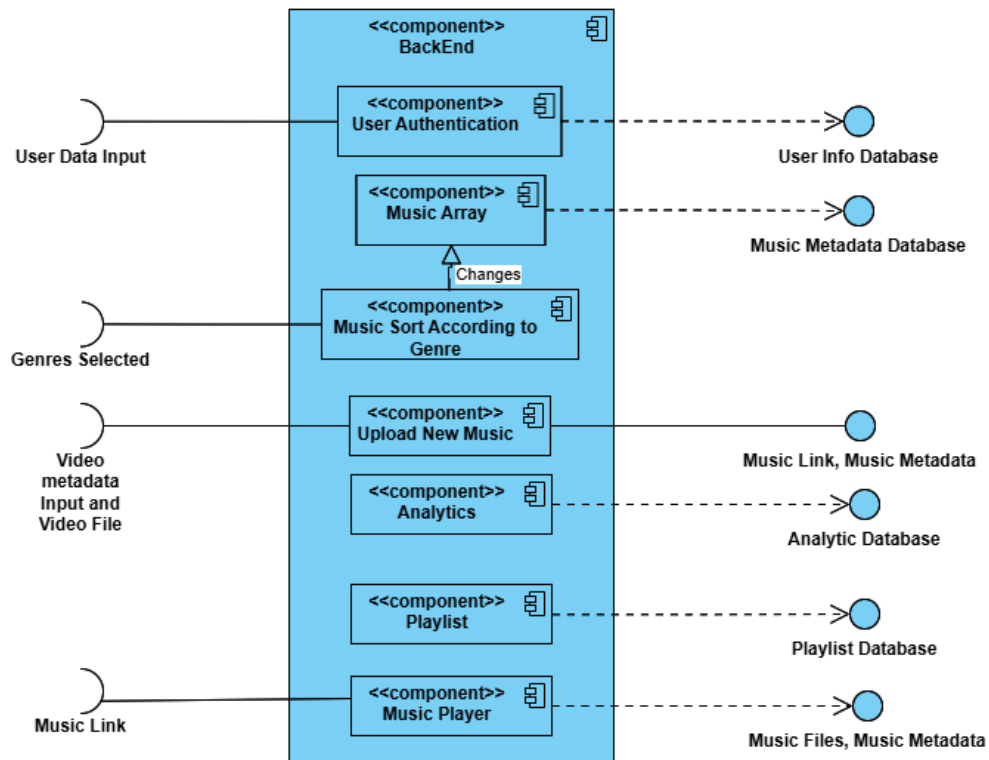


#### Components:

- **Login/Signup Module:** Handles user authentication and onboarding. Integrates with the backend for user data storage and retrieval.
- **Music Upload Interface:** Provides an interface for music creators to upload tracks, set music names, access controls, and generate sharing links.
- **Statistics Dashboard:** Displays music performance metrics, such as plays, likes, and dislikes, in a clean and minimalistic way.
- **Playlist Manager:** Allows users to create and manage playlists for their favorite tracks.
- **Audio Player:** A responsive and lightweight music player to stream music.

## 8.1.2 Backend (Application Layer)

The backend layer is the core of MusicHub's server-side logic. It handles API requests, processes page logic, and communicates with the data storage layer to ensure seamless functionality.



### Components:

- **Flask Framework:** Serves as the primary backend framework, providing RESTful API endpoints for communication between the front-end and back-end systems.
- **Request Handler:** Processes incoming user requests, validates inputs, and routes them to the appropriate service using the Fetch API.
- **Service Handlers:**
  - **Authentication Service:** Manages user login and signup.
  - **Upload Service:** Handles file uploads, validates metadata, and stores files securely in the cloud.
  - **Metrics Service:** Get and simplify data from the Metrics Database for real-time statistics in a simple manner..
- **Error Handler:** Manages exceptions and ensures graceful degradation of services during failures.

### **12.1.3 Data Storage Layer**

The Data Storage Layer is responsible for managing and securing all data within the MusicHub ecosystem. Our design combines structured database management with scalable cloud storage to optimize performance and handle diverse data types efficiently.

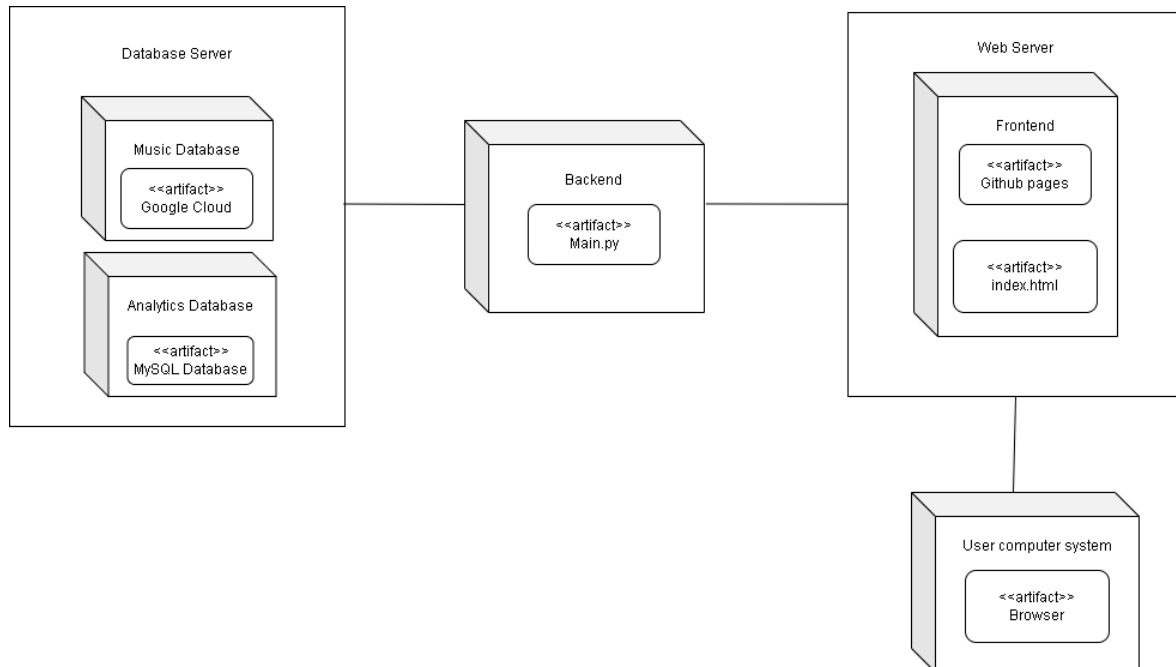
#### **Components:**

- **MySQL with SQLAlchemy:**
  - Utilized for storing structured data such as:
    - User account information
    - Music metadata
    - Performance metrics (e.g., play counts, likes, dislikes).
    - Music Comments
    - Playlist information (e.g., playlist titles and track associations).
    - Sharing link records (e.g., link expiration and access control settings).
- **Local File Storage:**
  - Reasons on why we chose it to store music files::
    - Accessibility - According to our needs, we determine that it is enough for a small user base.
    - Reliability - Its functionalities and compatibility with Flask and PythonAnywhere ensures that it would work reliably within our website.

This hybrid approach to data storage ensures that MusicHub remains efficient, scalable, resource friendly and secure while providing a seamless experience for users.

## 9 Deployment Diagram

### 9.1 Deployment Diagram



#### 1. Backend:

- This component is responsible for the core logic of the web application. It is hosted as a Python file and communicates with both the **Music Database** and the **Analytics Database**.

#### 2. Database Server:

- Contains two primary databases:
  - **Music Database:** Hosted on Local File Storage, used to store and manage music-related data such as tracks and metadata.
  - **Structured Database:** A MySQL database, used to store user data, password and other application-related metrics.

#### 3. Web Server (Frontend):

- The frontend is hosted on GitHub Pages.
- It includes html pages for the web design, css for extra providing an interface for the user to interact with the application.

#### 4. User Computer System:

- Represents the client-side system where users access the application through a browser.

## **9.2 Data Flow:**

- The **backend** communicates with both the **Music Database** and the **MySQL Database** to fetch and store required data.
- The **frontend** interacts with the backend to serve data to the user's browser.
- Users access the application through a browser running on their system, which retrieves the frontend from GitHub Pages.



## 10 Implementation

MusicHub is a **music-sharing platform** that allows users to upload, stream, and manage music. It uses a **Flask-based** backend with a **Jinja-templated frontend** and an **SQLAlchemy database** with local file storage as support for music files and image files

### Key Features:

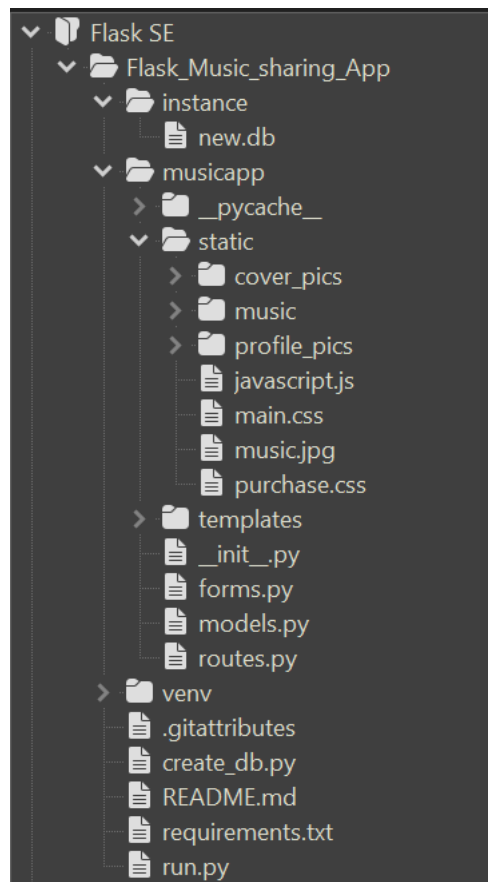
- User authentication (listeners & creators)
- Uploading and streaming music
- Filtering music by genre
- User comments and likes on music tracks
- Playlists for organizing music
- Music metrics viewable for viewers

### 10.1 Development Environment

#### Operating System & Tools:

- **Operating System:** Windows 10
- **Backend:** Flask (Python)
- **Frontend:** HTML, CSS (Bootstrap), JavaScript
- **Database:** SQLite / Local file storage
- **Version Control:** Git & GitHub
- **Development Tools:**
  - **IDE:** Visual Studio Code
  - **Virtual Environment:** venv
  - **Dependency Management:** requirements.txt

## Project Structure:



**Instance** contains the SQLAlchemy database

**Static** contain Javascript, CSS and the folders to store relevant music and image files.

**Templates** contain the HTML Files.

**Forms.py** contains Flask code used to created forms to be submitted.

**Models.py** contains database structures.

**Routes.py** contains all routing of codes and is the connection between all.

**Requirements.txt** contains all the dependencies needed by python for easier installation.

## 10.2 Software Integration

### How the System Works:

- **Frontend:** Users interact with the platform via an HTML/CSS/JavaScript interface.
- **Backend:** Flask processes user requests, retrieves/stores data, and serves music files.
- **Database:** SQLAlchemy ORM manages user accounts, uploaded songs, playlists, and interactions while local file storage manages profile picture, cover picture and music files.

File	Description
app.py	Main entry point of the Flask application. Initializes the app, database, and configurations.
models.py	Defines the database schema using SQLAlchemy ORM. Contains tables like User, Music, Playlist, and MusicMetrics.
routes.py	Contains Flask routes to handle user requests such as music uploads, streaming, and profile updates.
forms.py	Defines WTForms for user input validation (e.g., login, account updates, and music upload forms).
config.py	Stores application settings, including database configuration and security keys.
static/	Contains all static files such as CSS, JavaScript, and images.
static/music/	Stores uploaded MP3 files.
static/cover_pic/	Stores album cover images.
templates/	Contains HTML templates rendered by Flask using Jinja2.
instance/	Stores the local SQLite database (new.db) for testing purposes.
requirements.txt	Lists all Python dependencies required for the project.
run.py	Starts the Flask server for local development.

### Example of a page running:

```
# User Login
@bp.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('main.home'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.tryna_user.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user, remember=form.remember.data)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('main.home'))
        else:
            flash('Login Unsuccessful. Please check username and password', 'danger')
    return render_template('login.html', title='Login', form=form)
```

#### *Login Route in route.py*

The moment login() is called by the program, the template login.html is rendered by render\_template, while providing the needed title and forms. LoginForm from forms.py is used by calling it explicitly. The data then would be validated when the user submits to check if it is correct, if the data is required to be stored, it will be stored into the database here by using db.session.add() and db.session.commit. After everything is done, here it redirects the user to the home page.

```
class LoginForm(FlaskForm):
    tryna_user = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me')
    submit = SubmitField('Login')
```

#### *LoginForm from forms.py*

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Log In</legend>
                <div class="form-group">
                    {{ form.tryna_user.label(class="form-control-label") }}
                    {% if form.tryna_user.errors %}
                        {{ form.tryna_user(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.tryna_user.errors %}
                                <p> BOOP </p>
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.tryna_user(class="form-control form-control-lg") }}
                    {% endif %}
                </div>
                <div class="form-group">
                    {{ form.password.label(class="form-control-label") }}
                    {% if form.password.errors %}
                        {{ form.password(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.password.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.password(class="form-control form-control-lg") }}
                    {% endif %}
                </div>
                <div class="form-check">
                    {{ form.remember(class="form-check-input") }}
                    {{ form.remember.label(class="form-check-label") }}
                </div>
            </fieldset>
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-info") }}
            </div>
        </form>
    </div>
    <div class="border-top pt-3">
        <small class="text-muted">
            Need An Account? <a class="ml-2" href="{{ url_for('main.register') }}">Sign Up Now</a>
        </small>
    </div>
{% endblock content %}
```

*Login.html from /template*

The rest of the code follows the same format and idea. To prevent a bloated documentation, the rest of the code can be found in [https://github.com/ARandomRui/Flask\\_Music\\_sharing\\_App](https://github.com/ARandomRui/Flask_Music_sharing_App) with the same file structure as said previously.

## 10.3 Database

The database for the music-sharing application is implemented using SQLAlchemy ORM, which provides a structured way to manage data persistence. The database schema consists of multiple interrelated tables that handle user authentication, music storage, playlist management, social interactions, and premium memberships.

Below is an overview of the key database components:

Table Name	Primary Key	Foreign Keys	Description
User	id	None	Stores user details, including username, password, role, and profile image.
Music	id	creator_id → User, genre_id → Genre	Stores music details such as title, creator, upload date, and cover image.
MusicMetrics	music_id	music_id → Music	Tracks likes, dislikes, and views for each song.
Comment	id	user_id → User, music_id → Music	Stores user comments on music tracks.
Playlist	id	user_id → User	Stores playlists created by users.
PlaylistMusic	id	playlist_id → Playlist, music_id → Music	Many-to-Many relationship between playlists and music tracks.
Membership	id	creator_id → User	Stores membership plans offered by creators.
PurchasedMembership	id	user_id → User, membership_id → Membership	Stores which users have purchased memberships.
Genre	id	None	Stores different music genres available on the platform.

### Database Query Examples

#### 1. Retrieving All Songs Uploaded by a Specific Creator

```
creator_songs = Music.query.filter_by(creator_id=current_user.id).all()
```

#### 2. Fetching the Most Liked Songs

```
most_liked =  
MusicMetrics.query.order_by(MusicMetrics.like_count.desc()).limit(10).all()
```

#### 3. Retrieving Comments on a Specific Song

```
comments = Comment.query.filter_by(music_id=song_id).all()
```

## 11 Testing

### 11.1 Testing Strategy

#### 1. Top-Down Integration Testing

- **Purpose:** Ensures that high-level features work before integrating lower-level components.
- **Approach:** Testing starts with major modules, and stubs (temporary code) replace unfinished components.
- Example:
  - Test user authentication (login/signup).
  - Integrate music playback functionality.
  - Test playlist creation and management.
  - Add social sharing and external integrations.

#### 2. Inter-Class Testing

- **Purpose:** Ensures that different classes in the system interact correctly.
- **Approach:** Tests interactions between key components of the app.
- Example:
  - Validate that the User class correctly interacts with the Playlist class when adding a song.
  - Test whether the Music class retrieves metadata (e.g., artist, duration) from the database correctly.
  -

#### 3. Behavior Testing (Black-Box Testing)

- **Purpose:** Validates system behavior without checking internal code.
- **Approach:** Focuses on expected outputs based on different user inputs.
- Example:
  - When a user uploads an unsupported file format, the system should display an error.
  - If a song is deleted, it should no longer appear in the user's playlist.

#### 4. Partition Testing (Equivalence Partitioning & Boundary Testing)

- **Purpose:** Ensures input fields and file handling work correctly.
- **Approach:** Divides inputs into valid and invalid groups to test system responses.
- Example:
  - Equivalence Partitioning: Test music uploads with different file formats (MP3, WAV, FLAC, unsupported formats).
  - Boundary Testing: If the upload limit is 50MB, test with 49MB, 50MB, and 51MB files.

#### 5. Random Testing (Exploratory Testing)

- **Purpose:** Identifies unexpected system failures by performing random actions.
- **Approach:** Testers interact unpredictably with the app to simulate real-world usage.

- Example:
  - Rapidly switch between songs to check for playback issues.
  - Upload multiple large files at once to test system stability.
  - Randomly press buttons and navigate menus to find UI bugs

## 11.2 Test Data

### 1. User Authentication Test Data

- Purpose: To verify login, signup, and authentication mechanisms.

### 2. Music Upload & Storage Test Data

- Purpose: To test the system's ability to handle different file types and sizes.

### 3. Music Playback Test Data

- Purpose: To verify smooth streaming and playback.

### 4. Social Sharing & Interaction Test Data

- Purpose: To test sharing features and interactions.

## 11.3 Acceptance Testing

Criteria	Fulfilled	Remarks
User should be able to sign up and log in successfully.	Yes	Works as expected.
System should show an error for incorrect credentials.	Yes	Displays "Incorrect password" message
Songs should play without buffering on a stable internet.	Yes	Smooth playback on high-speed connection.
Users should be able to create and manage playlists.	Yes	Playlists can be created, edited, and deleted.
Users should be able to share songs with friends.	Yes	Link could be share successfully to friends
Users should be able to save music into playlist.	Yes	Users are able to see their saved music in "My Playlist"
User should be able to comment on music.	Yes	Music Creator is able to view the respective messages by other users
Music Creators should be able to view music metrics	Yes	Music Creator is able to view the music metrics.

Date tested : 11 / 2 / 2025

% Complete : 100%

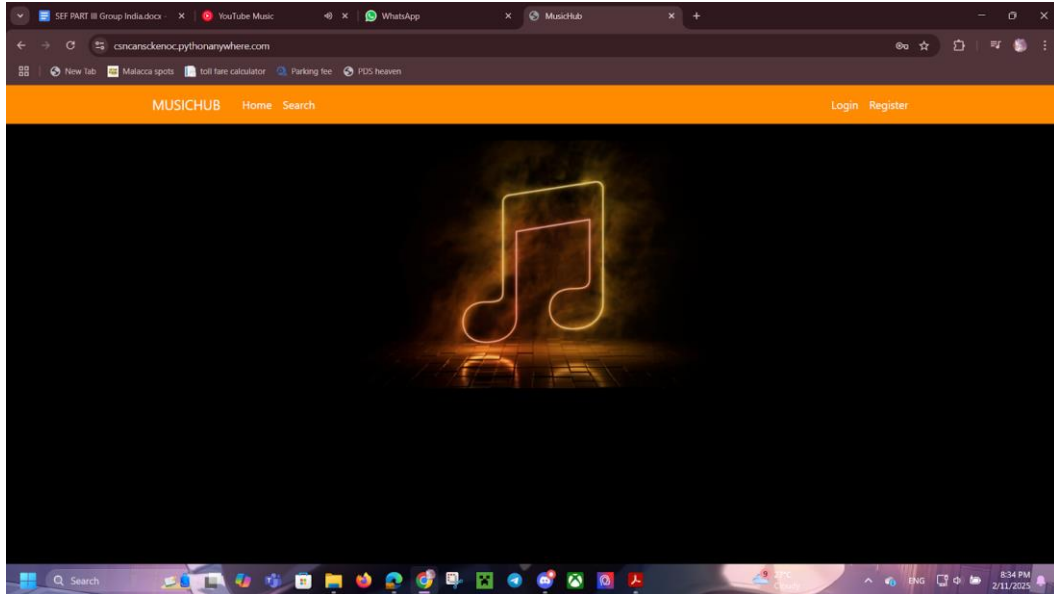
Tested by : Chan Ka Ken

Verified by : Chee Rui



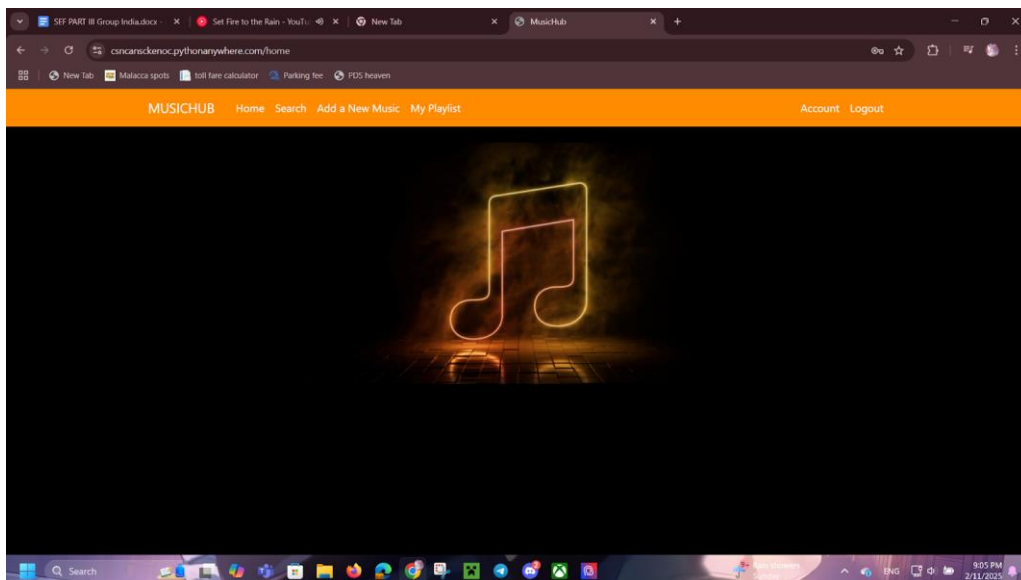
## 12 Sample Screens

### 12.1 Main Screen



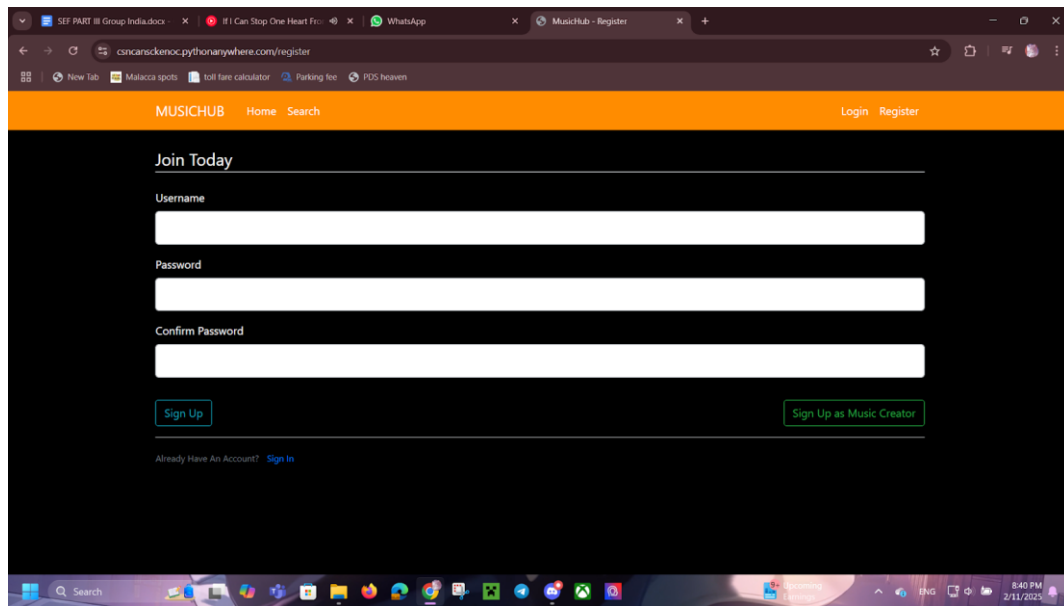
This is the main screen before login or register. On the top orange toolbar, it has the Home, Search, Login and Register buttons.

#### 12.1.1 Home Menu



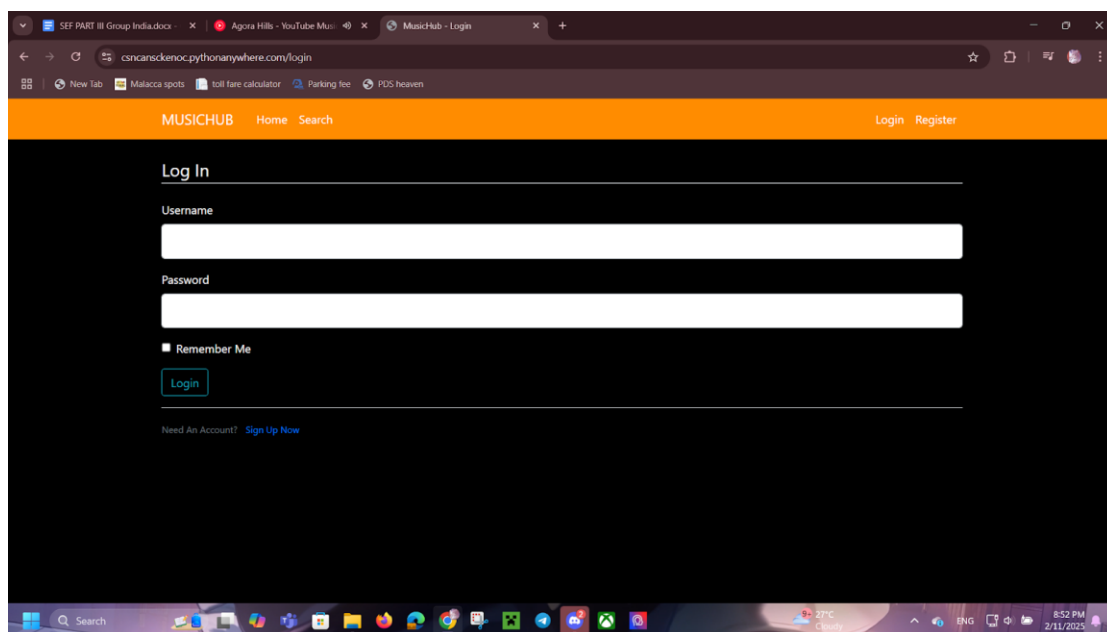
This is the Home menu. As you have login your account, you can access the features provided in the top orange toolbar. For music creators, there is an extra feature of adding music.

## 12.1.2 Register

A screenshot of a web browser displaying the 'MusicHub - Register' page. The browser's address bar shows the URL 'cscanskenoc.pythonanywhere.com/register'. The page has a dark theme with an orange header bar. The header contains the text 'MUSICHUB' followed by 'Home' and 'Search' links, and 'Login' and 'Register' links on the right. The main content area is titled 'Join Today' and features three input fields for 'Username', 'Password', and 'Confirm Password'. Below these fields are two buttons: 'Sign Up' and 'Sign Up as Music Creator'. At the bottom of the form, there is a link that says 'Already Have An Account? Sign In'.

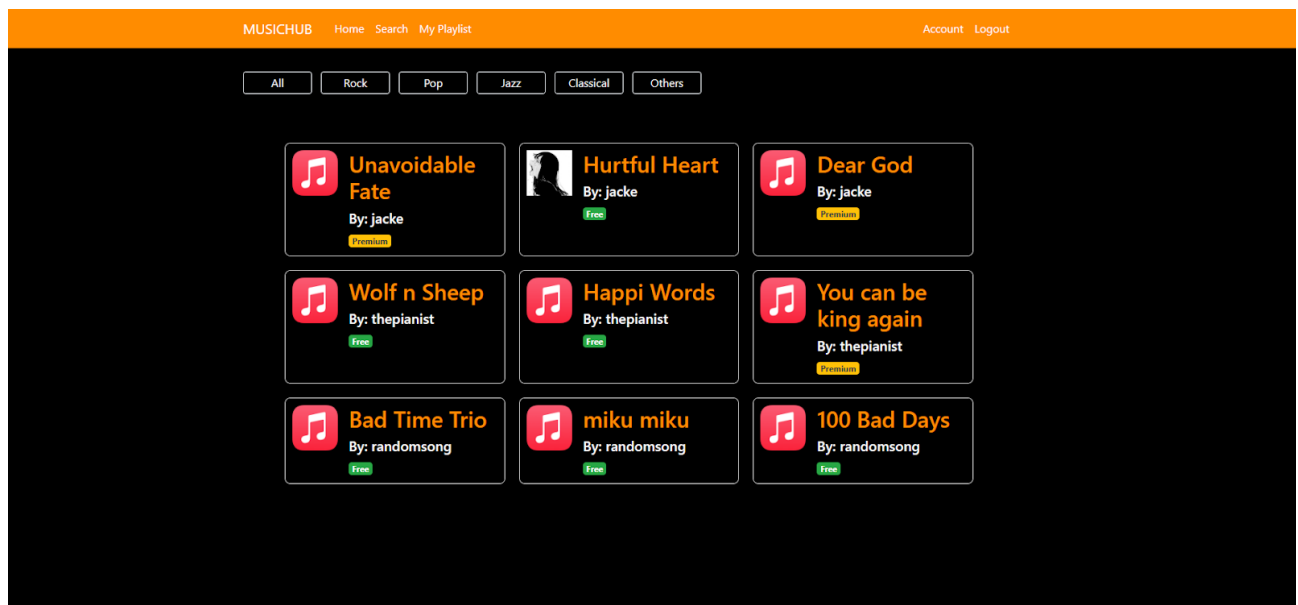
This is the Register panel. Users can enter their username and password. Users can also choose to be a normal account or a music creator account.

## 12.1.3 Login

A screenshot of a web browser displaying the 'MusicHub - Login' page. The browser's address bar shows the URL 'cscanskenoc.pythonanywhere.com/login'. The page has a dark theme with an orange header bar. The header contains the text 'MUSICHUB' followed by 'Home' and 'Search' links, and 'Login' and 'Register' links on the right. The main content area is titled 'Log In' and features two input fields for 'Username' and 'Password'. Below these fields is a checkbox labeled 'Remember Me' and a 'Login' button. At the bottom of the form, there is a link that says 'Need An Account? Sign Up Now'.

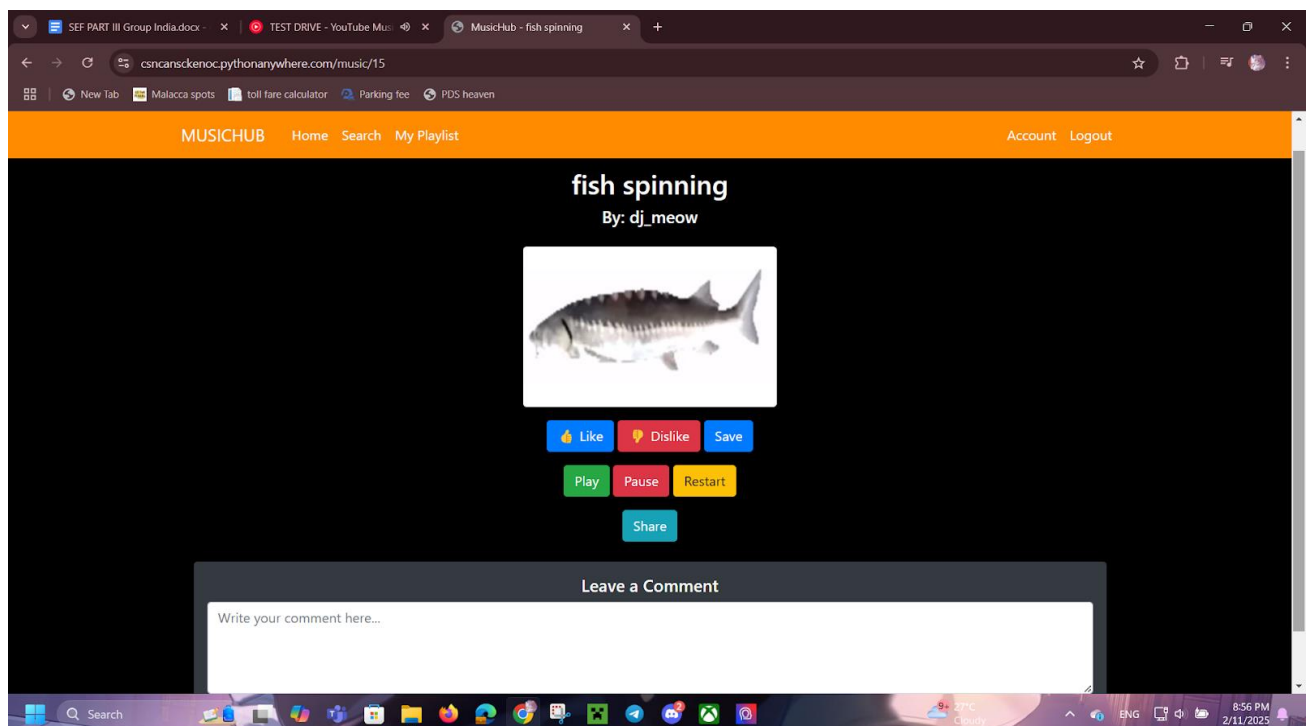
This is the Login panel. Users can login to their existing account and use every feature provided.

## 12.1.4 Search



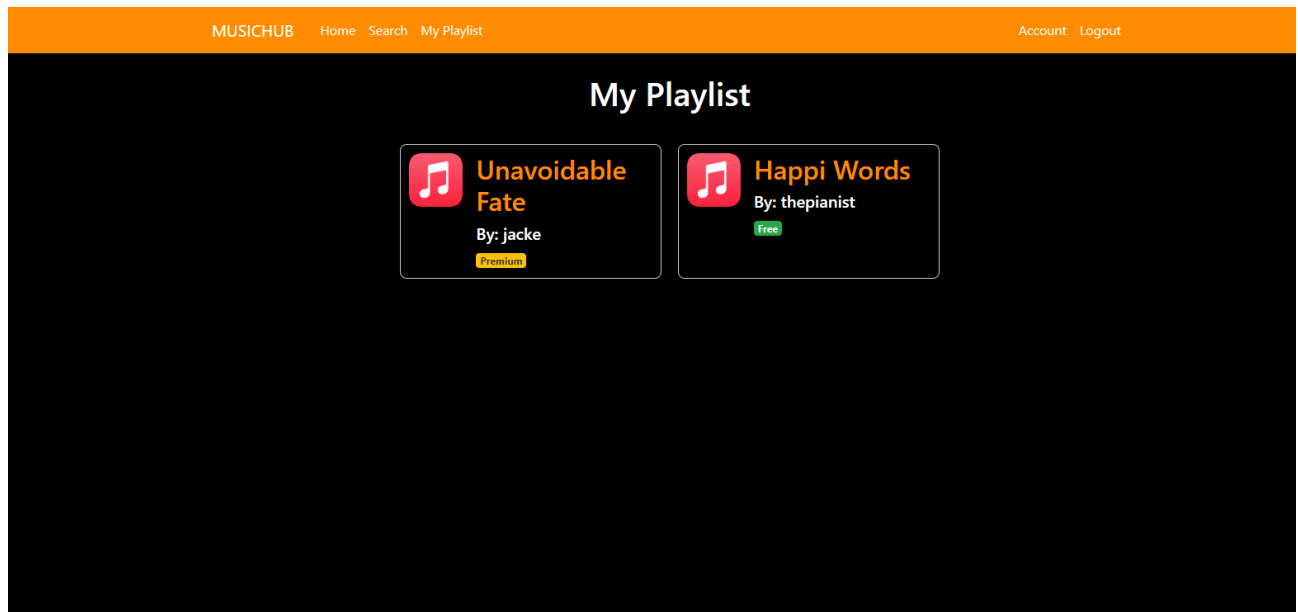
This is the Search panel. Users can search for music using the genre below the orange toolbar and click on the music.

## 12.1.5 Music



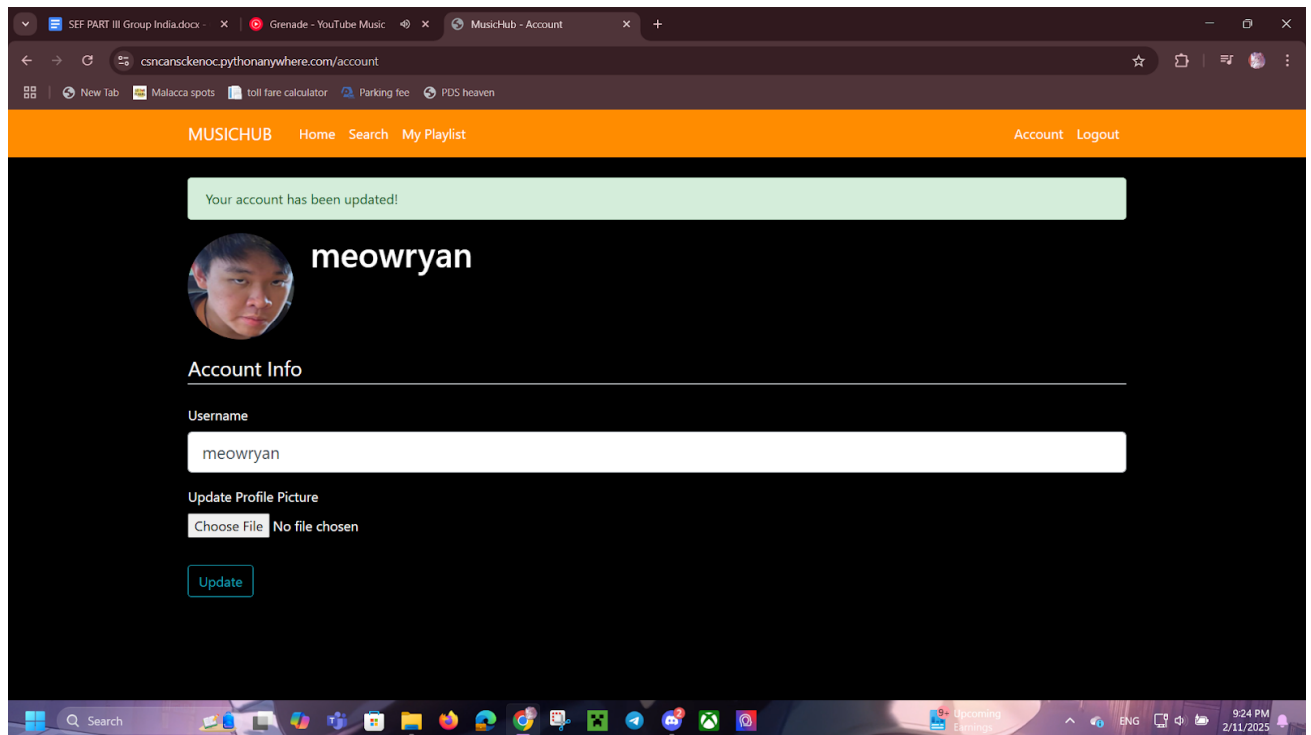
This is the Music panel. Users can like, dislike, save, play, pause, restart, share and comment on the music.

## 12.1.6 Playlist



This is the Playlist panel. The music saved will appear here.

## 12.1.7 Account



This is the Account panel. Users can edit their profiles here.

## 13 Conclusion

### 1. Completion of Software

The development of the **Music-Sharing Platform** was successfully completed, ensuring seamless music uploading, sharing, and playback. The following core components were implemented:

- **User Authentication & Management:**
  - Flask-Login for secure user authentication
  - User registration, login, password reset, and profile management
- **Music Management System:**
  - Users can upload, manage, and share music
  - Metadata storage for songs (title, artist, tags, duration)
  - Playlist creation and management
- **Frontend & UI Implementation:**
  - HTML, CSS, and JavaScript for a responsive and intuitive interface
  - Jinja2 templating for dynamic content rendering
- **Data Storage Layer:**
  - **SQLite with SQLAlchemy** for structured user and music data
  - **Local storage** for music files and user profile images

These implementations ensure that the platform provides a simple yet efficient music-sharing experience, free from unnecessary complexities like search indexing or moderators.

### 2. Software Quality Assurance

To maintain a **high standard of data reliability and performance**, our team conducted multiple levels of testing:

- **Unit Testing:** Verified the correctness of database operations, such as inserting, updating, and deleting records.
- **Integration Testing:** Ensured smooth data flow between the backend API, the frontend, and cloud storage.
- **Performance Testing:** Evaluated database query speeds and optimized indexes for large datasets.
- **Security Testing:** Applied encryption techniques to protect sensitive user data and prevent unauthorized access.

Through these quality assurance efforts, we ensured that the **overall software** is robust, scalable, and secure.

### 3. Group Collaboration

The development was a team effort, with clear communication and division of responsibilities to ensure smooth progress. Collaboration efforts included:

- **Database Schema Design:** Team discussions helped finalize table structures to ensure efficiency and scalability.
- **Report Management:** Team members coordinated to document progress, challenges, and solutions in the final project report, ensuring clarity and accuracy in technical explanations.
- **Code Management:** Version control through GitHub was used to track changes and prevent conflicts.
- **Testing & Debugging:** Peer reviews and collective troubleshooting sessions helped identify and resolve issues effectively.

By maintaining clear communication and structured workflows, our team ensured a **seamless integration** of the system.

### 4. Problems Encountered & Solutions

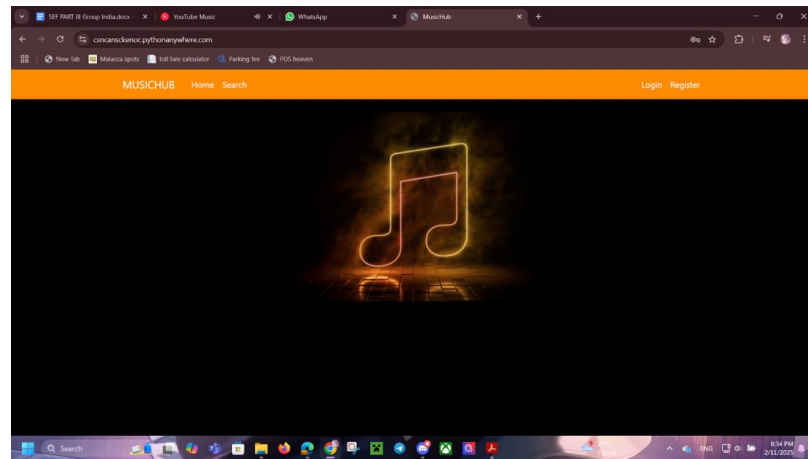
1. **Data Synchronization Challenges:**
  - Issue: When a creator deleted a track, the associated music file and cover image remained in the server directory.
  - Solution: Implemented **related file deletion** in the `delete_music()` function and deleted database instances dependent on said music. Also ensured the cover image was only deleted if it wasn't the default.
2. **Performance Bottlenecks with Large Playlists:**
  - Issue: Query execution times were high when retrieving large playlists.
  - Solution: We optimized **database indexing** and refined queries to enhance performance.
3. **Schema Modification Conflicts:**
  - Issue: As multiple team members modified database models, conflicts occurred during merges.
  - Solution: We enforced **consistent migration scripts** and scheduled **regular schema review meetings** to minimize conflicts.
4. **Security Risks in User Authentication Data:**
  - Issue: Storing authentication data required extra protection against potential breaches.
  - Solution: Implemented **password hashing** and **token-based authentication** to enhance data security.

## 14 User Guide

### 5. For every users:

<https://csncansckenoc.pythonanywhere.com/>

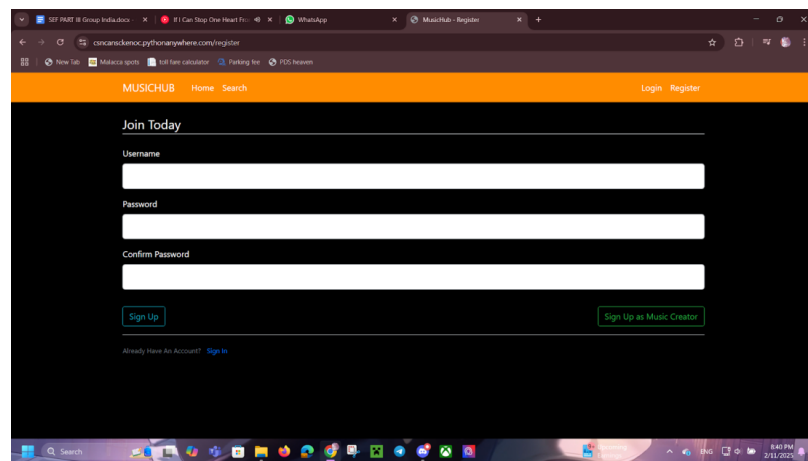
Copy the link and paste it in your browser.



This is “Home” of our program.

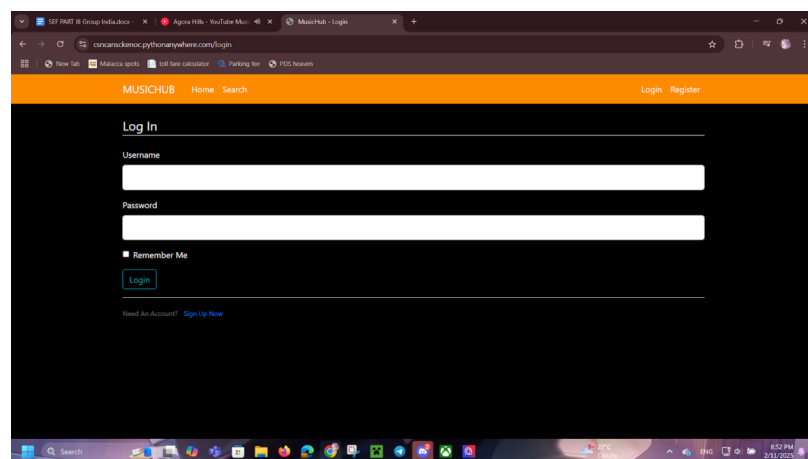
Buttons you may click on the top bar:

1. Search: For music
2. Login: For existing user
3. Register: For new user



This is the “Register” panel.

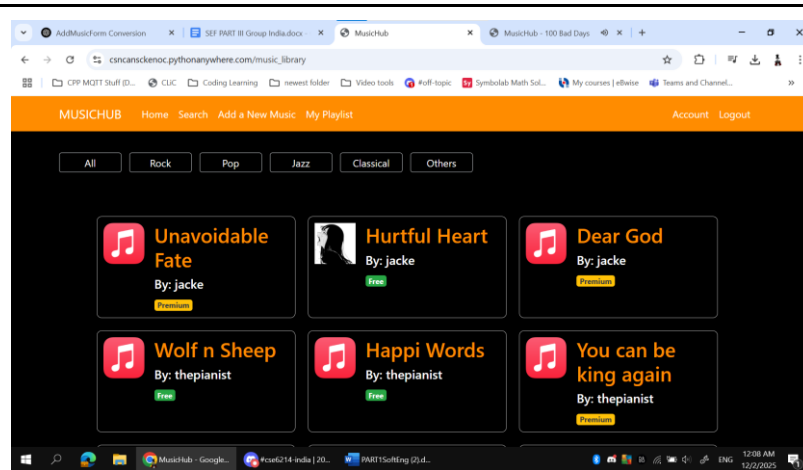
Fill in the “Username” and “Password” to sign up for an account. If you desire to be a music creator, click the green button. If not, click the blue button.



This is the “Login” panel.

Fill in the “Username” and “Password” to log in to your existing account.

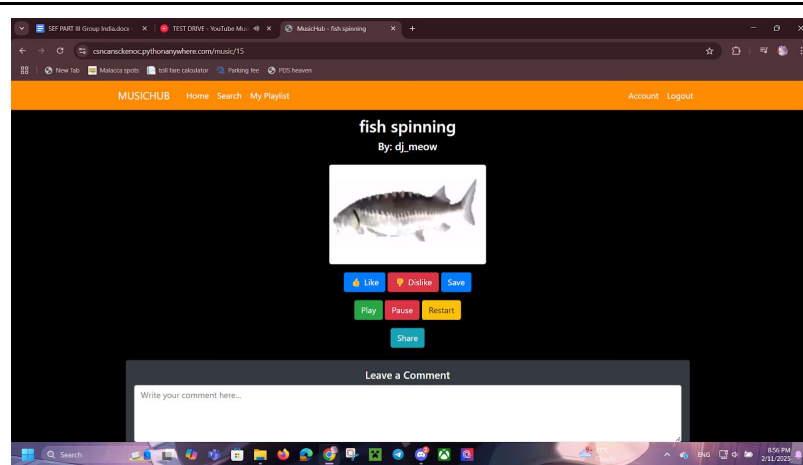
## Software Requirements Specification for Music Sharing System



This is the “Search” panel.

You may search the music you want to hear with the genres below the orange tool bar.

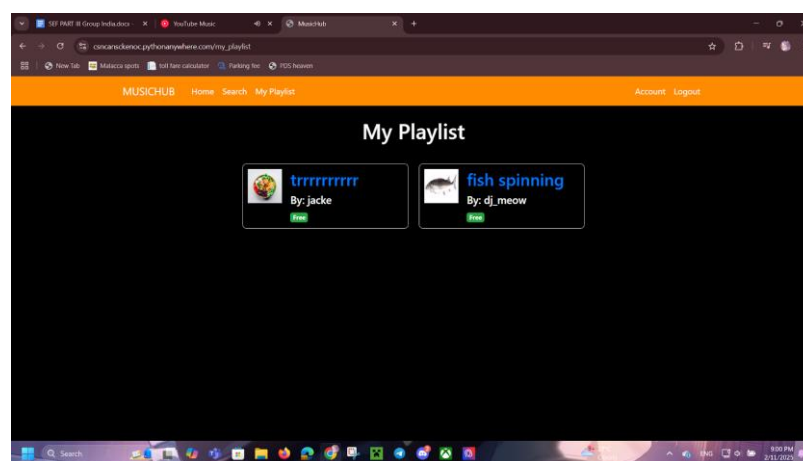
Click on the music you want to listen.



This is the “Music” panel.

All the buttons are the functions stated:

1. Like
2. Dislike
3. Save (to Playlist)
4. Play
5. Pause
6. Restart
7. Share (Link to clipboard)
8. Comment

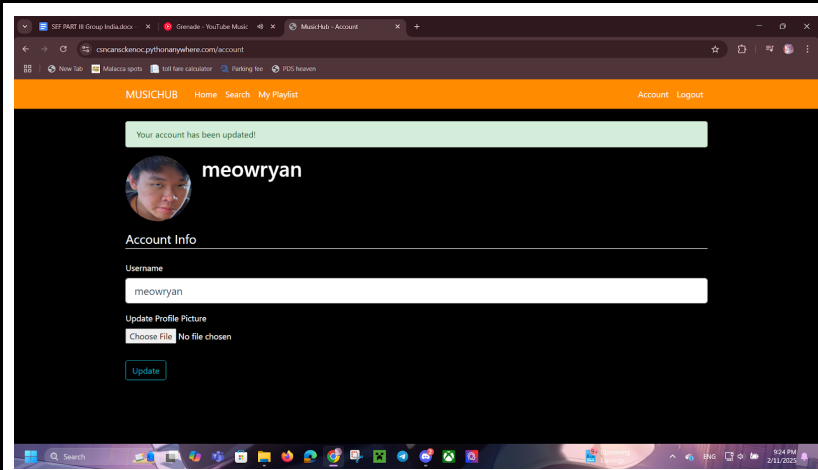


This is the “My Playlist” panel.

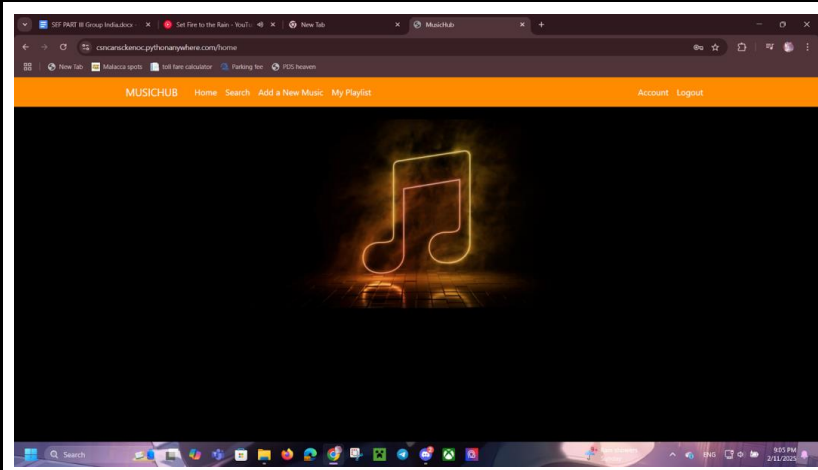
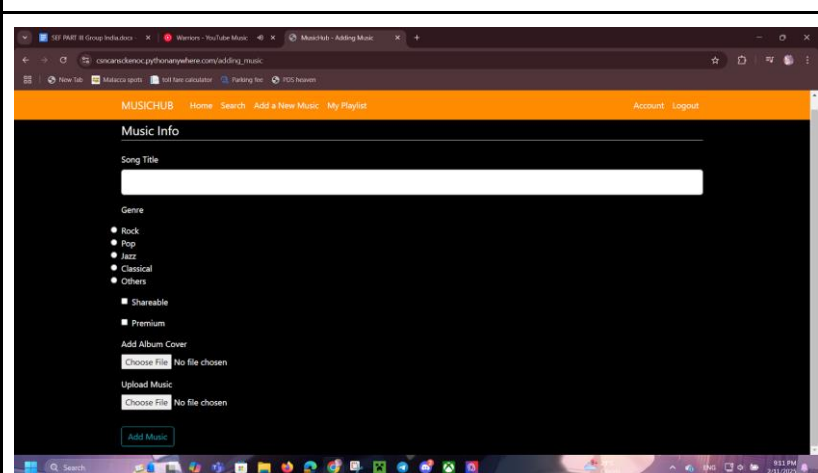
After you clicked the “Save” button in the “Music”, the song will be saved here.

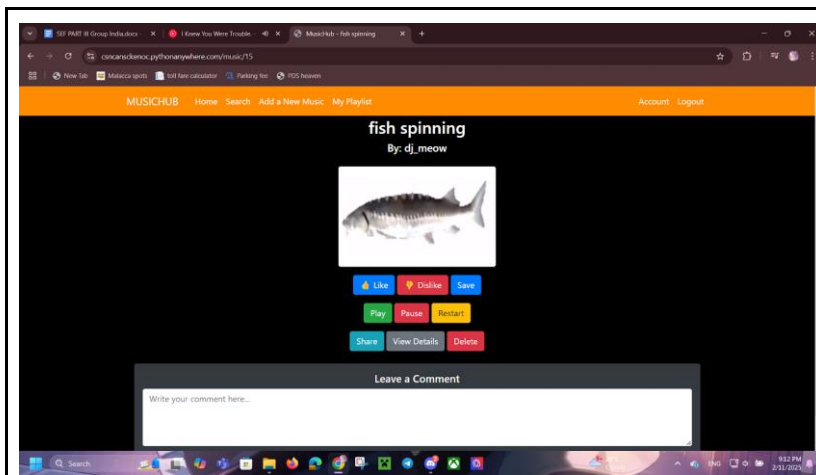
Click on the song you want to hear.



	<p>This is the “Account” panel.</p> <p>You can edit your username and your profile picture.</p>
---	---

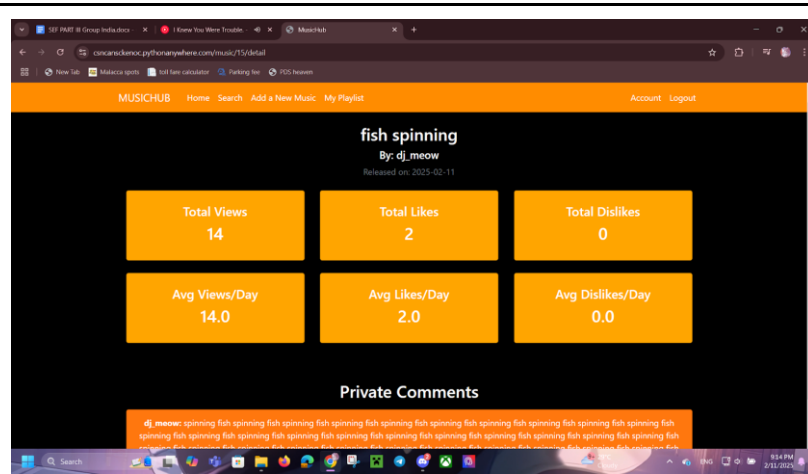
## 6. Only for music creator:

	<p>For the “Home” menu, the top bar will appear one more feature, which is the “Add a New Music” feature.</p> <p>You can add new music into it.</p>
	<p>This is the “Add a New Music” panel.</p> <p>Insert the song details needed. Then you may add your music.</p>



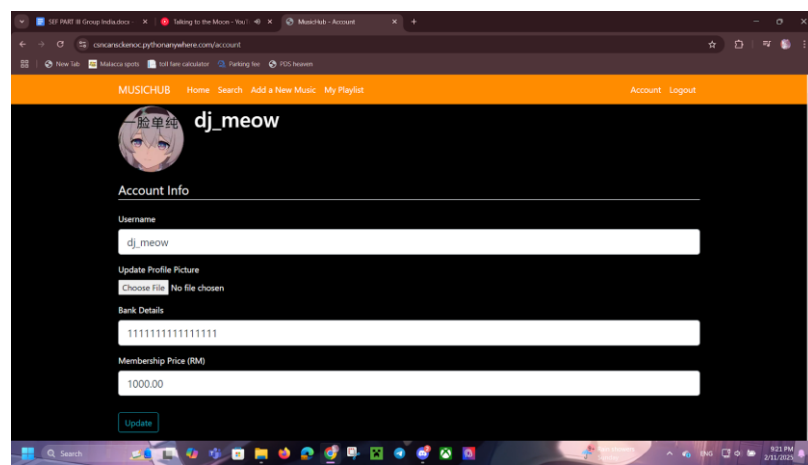
This is the “Music” panel.

Different from normal users’ “Music” panel, you can see your song details and even delete your song.



This is the “View Detail” panel.

You can view your particular video



This is the “Account” panel.

You can edit your username, profile picture, bank details and your membership price.