

SWE Project - 1: Bonus

Members

- Adhiraj Anil Deshmukh (2021121012)
- Arjun Muraleedharan (2020101099)
- Pratham Gupta (2020101080)
- Keyur Ganesh Chaudhari (2020101100)
- Sambasai Reddy Andem (2020101014)

1E - Bonus

1. User Management

This is the subsystem that is involved in managing user accounts for Music, taking care of features such as authentication, user creation, and keeping track of albums and channels associated with specific users.

Structure: The following are the classes associated with this subsystem.

- `User` : Represents a user in the system.
- `AuthenticationToken` : Stores details of authentication tokens used to keep users logged into Music.
- `UserTrack` : Represents a user's relationship with a track. Users can listen to tracks on Music and like/unlike them.
- `UserAlbum` : Represents a user's relationship with an album. Users can rate a particular score for albums.

`UserTrack` and `UserAlbum` would come under user management since their specific purpose is to represent how the user would interact with the rest of the system. That is the reason `Playlist` has not been included even though each user can create and manage playlists, since `Playlist` represents an independent entity and is not built on some form of interaction.

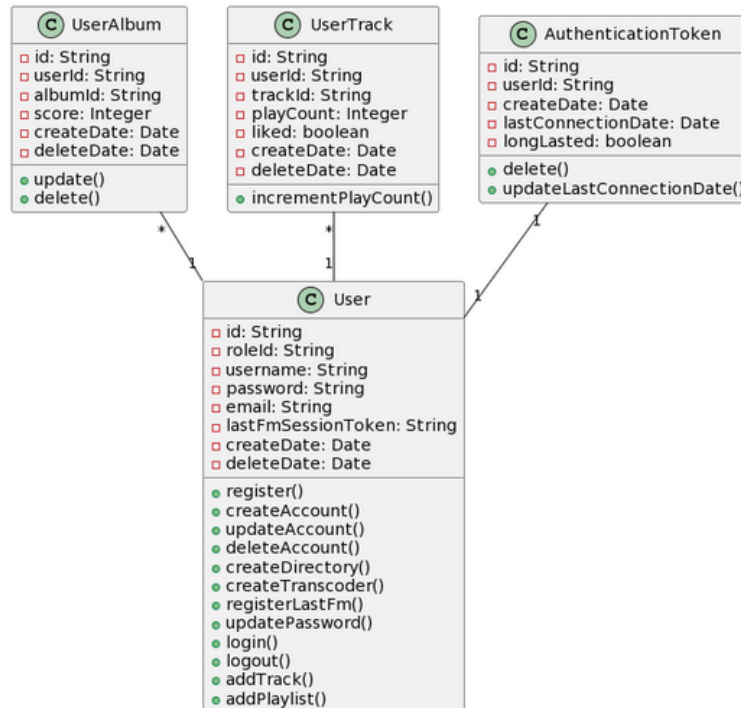
Functionality: This subsystem has the following features.

1. User creation, updation, deletion, and authentication
2. Maintaining records of user-album and user-track relationships

Behavior: The behavior of the system would be explained in terms of the above features.

1. First, the user (assuming they have an account) logs in to the system. The process of authentication is performed via the `login()` function. It would verify the username and password of the user and create an authentication token on the server, which is represented by the `AuthenticationToken` class. The authentication token would include a reference to the user, which helps in checking privileges and obtaining user details if relevant. For logging out, the authentication token would be deleted.
2. For user creation, updation, and deletion, the privileges of the user have to be that of an admin. For the sake of abstraction, we have opted to mention the relevant classes and functioning of the administrative system in a separate section. Firstly, the subsystem checks if the role of the user is "admin" via the `roleId` attribute of the user and that they have the required privileges to create/update/delete a user. Then, a user object is created with the supplied details, or in the case of updation and deletion, the user is retrieved and updated/deleted.
3. Whenever, albums and tracks are added, listened to, liked/unliked or scored by the user, records of `UserTrack` and `UserAlbum` would be created to store the relationship between the user and these tracks/albums.

Bearing the above points in mind, the following is our proposed UML diagram for the subsystem.



`AuthenticationToken` would have a one-one relationship with `User`, since a user can only have one authentication token stored at a particular time while logged in. However, since a user can like/unlike or listen to multiple tracks and create multiple albums, `UserAlbum` and `UserTrack` have many-one relationships with `User`. All of the relationships in this system are simple associations.

▼ 1.1. Action Space 'U'

1. Log into account
2. Check if user has the privilege to access user options
3. Check if user has the privilege to access directory options
4. Check if user has the privilege to access transcoder options
5. Log out of account
6. Create new user account
7. Update user account
8. Delete user account
9. Create directory
10. Create transcoder
11. Update directory
12. Update transcoder
13. Delete directory
14. Delete transcoder
15. Validate Directory

▼ 1.2. Output Space 'Y'

1. Login page
2. Home page
3. Actions drop-down
4. User access page
5. Directory access page
6. Transcoder access page
7. User visible on user access page
8. Changes visible on user access page
9. User not visible on user access page
10. Directory visible on directory access page
11. Changes visible on directory access page
12. Directory not visible on directory access page
13. Transcoder visible on transcoder access page
14. Changes visible on transcoder access page
15. Transcoder not visible on transcoder access page

▼ **1.3. States 'X'**

1. NotLoggedIn
2. LoggedIn
3. ViewingAllActions
4. ViewingUserActions
5. ViewingDirectoryActions
6. ViewingTranscoderActions
7. UserAdded
8. UserUpdated
9. UserRemoved
10. BeforeDirectoryValidation
11. InvalidDirectoryAdded
12. InvalidDirectoryUpdated
13. ValidDirectoryAdded
14. ValidDirectoryUpdated
15. DirectoryRemoved
16. TranscoderAdded
17. TranscoderUpdated
18. TranscoderRemoved

▼ **1.4. Initial State 'X0'**

NotLoggedIn

▼ 1.5. Display Map 'h'

State	Observable
NotLoggedIn	Login page
LoggedIn	Home page
ViewingAllActions	Actions drop-down
ViewingUserActions	User access page
ViewingDirectoryActions	Directory access page
ViewingTranscoderActions	Transcoder access page
UserAdded	User visible on user access page
UserUpdated	Changes visible on user access page
UserRemoved	User not visible on user access page
BeforeDirectoryValidation	-
DirectoryAdded	Directory visible on directory access page
DirectoryUpdated	Changes visible on directory access page
DirectoryRemoved	Directory not visible on directory access page
TranscoderAdded	Transcoder visible on transcoder access page
TranscoderUpdated	Changes visible on transcoder access page
TranscoderRemoved	Transcoder not visible on transcoder access page

▼ 1.6. Transition Relation

Current State	Action	Next State
NotLoggedIn	Log into account	LoggedIn
LoggedIn	View all actions	ViewingAllActions
ViewingAllActions	Check if user has the privilege to access user options	ViewingUserActions
ViewingAllActions	Check if user has the privilege to access directory options	ViewingDirectoryOptions
ViewingAllActions	Check if user has the privilege to access transcoder options	ViewingTranscoderOptions
ViewingUserActions	Create new user account	UserAdded
ViewingUserActions	Update user account	UserUpdated
ViewingUserActions	Delete user account	UserRemoved
ViewingTranscoderActions	Create transcoder	TranscoderAdded
ViewingTranscoderActions	Update transcoder	TranscoderUpdated
ViewingTranscoderActions	Delete transcoder	TranscoderRemoved
ViewingDirectoryOptions	Create directory	BeforeDirectoryValidation
ViewingDirectoryOptions	Update directory	BeforeDirectoryValidation
BeforeDirectoryValidation	Validate directory	DirectoryAdded/DirectoryRemoved (depending on the previous state)
ViewingDirectoryOptions	Delete directory	DirectoryRemoved
(Any state)	Log out of account	NotLoggedIn

▼ Assumptions

1. By checking if a user has the privilege to access a certain set of options, we assume that the process leads to a successful verification of the possession of the privilege. If the user does not have the privilege, this action would never be able to occur in the first place, and therefore, they would not be able to access the options, in turn, not leading to that state.
2. "Validate directory" refers simply to the process wherein the path of a directory is checked for correctness. This correctness does not affect the creation/update of a directory, so the state is guaranteed to go to one of `DirectoryAdded` or `DirectoryRemoved`.
3. In the last transition relation, the Logout action from any state would lead to the `NotLoggedIn` state.

2. Last FM Integration

This subsystem manages the Music system's integration with the Last.fm service. Music allows users to link their Last.fm profile, and sync their listening history with it in order to build a profile of the user's musical taste.

Structure: The following are the classes associated with this subsystem.

- `User` : Represents a user in the system. This class would store the Last.fm token of the user.
- `LastFmSession` : Represents a Last.fm session of a particular user. The session lasts for as long as the user has linked their credentials to Music, and would be destroyed when the user unlinks.
- `Track` : Represents a music track. Last.fm requires access to the listening history of the user and so would be related to the `Track` class.
- `UserTrack` : Represents a user's relationship with a track.

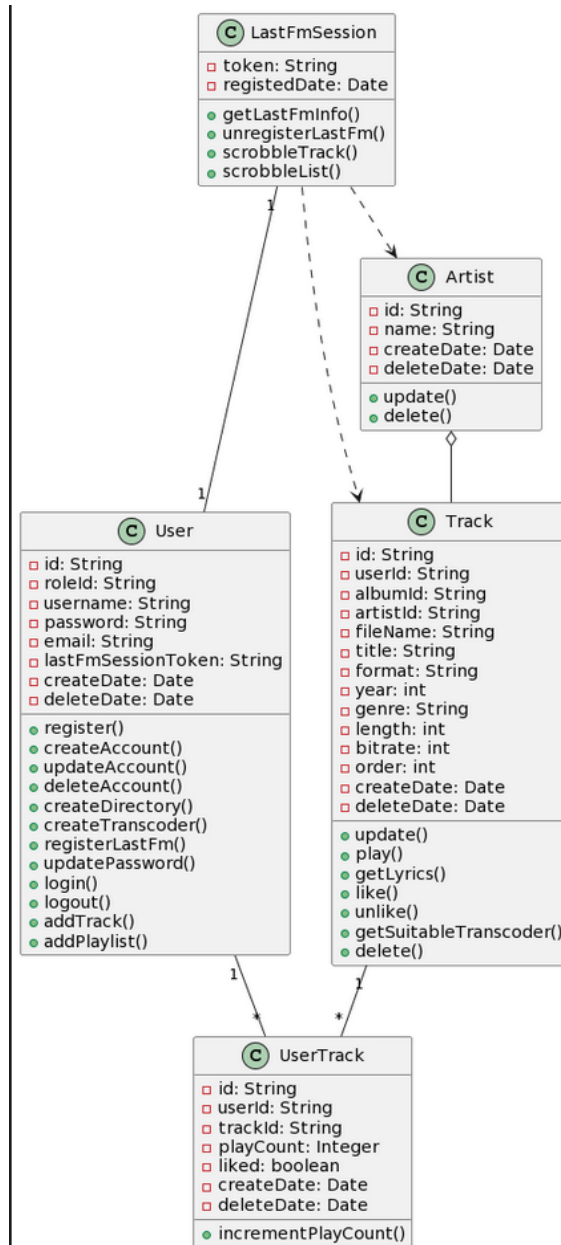
Functionality: The purpose of the subsystem is to do the following.

1. Enable the user to link their Last.fm account with their Music account.
2. Allow Last.fm access to users' listened-to tracks to analyze them.

Behavior: The behavior of the system would be explained in terms of the above features.

1. When the user enters their Last.fm credentials and submits them, a session is established with Last.fm via a token which would be stored in the `User` class. This is done via the `registerLastFm()` method. As long as this session lasts (until the user unlinks their Last.fm account from Music), the Last.fm service would be integrated with Music. The `unregisterLastFm()` method would delete this session.
2. In order to have the Last.fm service perform analysis on the tracks listened to by the user, the `scrobbleTrack()` and `scrobbleList()` (for a list of tracks) methods are present. These take in the track(s) and have Last.fm perform analysis on them. These methods require access to both the track and the artist in order to perform its analysis, hence the presence of these two classes in the subsystem.

Bearing the above points in mind, the following is our proposed UML diagram for the subsystem.



As mentioned in the User Management section, **User** would have a one-many association with **UserTrack** and so would **Track**, since one track could be interacted with by multiple users. Since **LastFmSession** uses the **Artist** and **Track** classes for scrobbling, it would have dependencies on those classes. Since one artist can have multiple tracks, but **Artist** and **Track** are independent entities within the system, there is an aggregation relationship between them.

Transition Model

▼ Action Space (U)

1. Enter Last.fm credentials (Correct)
2. Enter Last.fm credentials (Incorrect)
3. Logout from Last.fm account

4. Play Track
5. Play Playlist/Album
6. Stop Playing

▼ States (X)

1. [Last.fm](#) Not Linked
2. [Last.fm](#) Linked
3. Playing Track
4. Playing Playlist/Album

▼ Initial State (X0)

- [Last.fm](#) not Linked

▼ Output Space (Y)

1. [Last.fm](#) Login Section

Last.fm scrobbling

Last.fm username and password won't be stored.

**Last.fm
username**

**Last.fm
password**

⚡ Connect

2. [Last.fm](#) Login Successful

Last.fm scrobbling

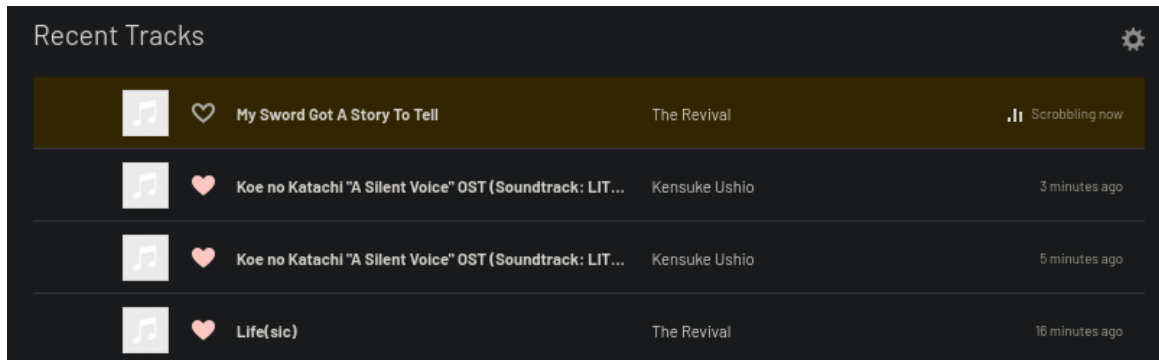
Your account is linked to Last.fm

ffspratham

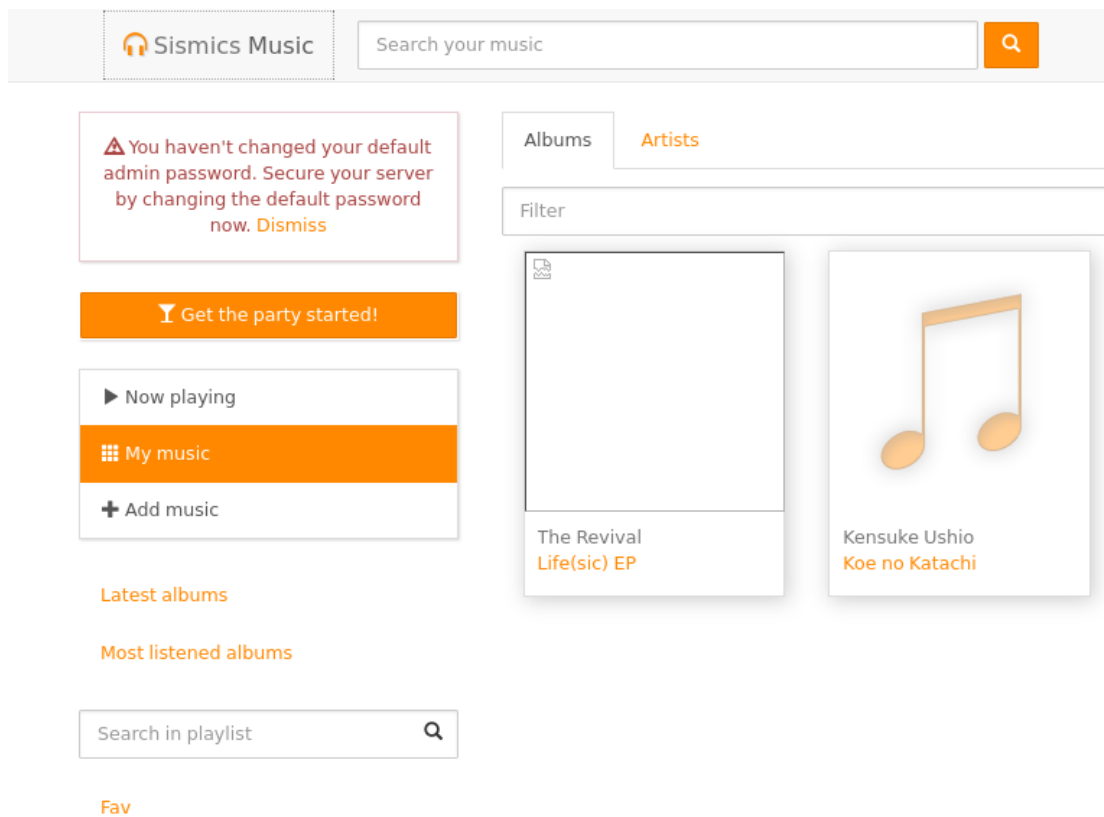
Scrobbled 1 songs since Feb 22, 2023.



3. Scrobbling Track / Playlist / Album



4. Home Screen



▼ Display Map (h)

X	Y
Last.fm Not Linked	Last.fm Login Section
Last.fm Linked	Last.fm Login Successful / Home Screen
Playing Track	Scrobbling Track
Playing Playlist/Album	Scrobbling Playlist/Album

▼ Transition Relation (f)

States \ Actions	Enter <u>Last.fm</u> Credentials (Incorrect)	Enter <u>Last.fm</u> Credentials (Correct)	Logout from <u>Last.fm</u> Account	Play Track	Play Playlist/Album	Stop Playing
<u>Last.fm</u> Not Linked	<u>Last.fm</u> Not Linked	<u>Last.fm</u> Linked	-	-	-	-
<u>Last.fm</u> Linked	-	-	<u>Last.fm</u> Not Linked	Playing Track	Playing Playlist/Album	-
Track Playing	-	-	<u>Last.fm</u> Not Linked	Playing Track	Playing Playlist/Album	<u>Last.fm</u> Linked
Playlist/Album Playing	-	-	<u>Last.fm</u> Not Linked	Playing Track	Playing Playlist/Album	<u>Last.fm</u> Linked

3. Administrator Features

This is the subsystem that handles the administrative features made available to certain users via their roles and privileges in the system. The system follows a role-based access control system, allowing for more security.

Structure: The following are the classes associated with this subsystem.

- **User** : Represents a user in the system.
- **Role** : Represents a particular role in the system. While the roles as we understand it currently are “user” and “admin”, having a dedicated class for roles opens up the possibility of adding differing degrees of role-based access control (such as allowing some advanced features for a certain kind of user, but not others).
- **Privilege** : Represents a particular privilege (like having admin permissions or having privileges to change passwords). Allows for greater flexibility in the mentioned role-based access control system as you can choose which roles get which privileges.
- **RolePrivilege** : Links roles and privileges.
- **Directory** : Represents a local directory where music is stored.
- **Transcoder** : Represents a transcoder entity.

The **Directory** and **Transcoder** classes do not directly play a role in the administrative system but are present merely because their features are controlled by the admins.

Functionality: The administrative subsystem permits certain users, called admins, to exercise certain privileges not available to other users.

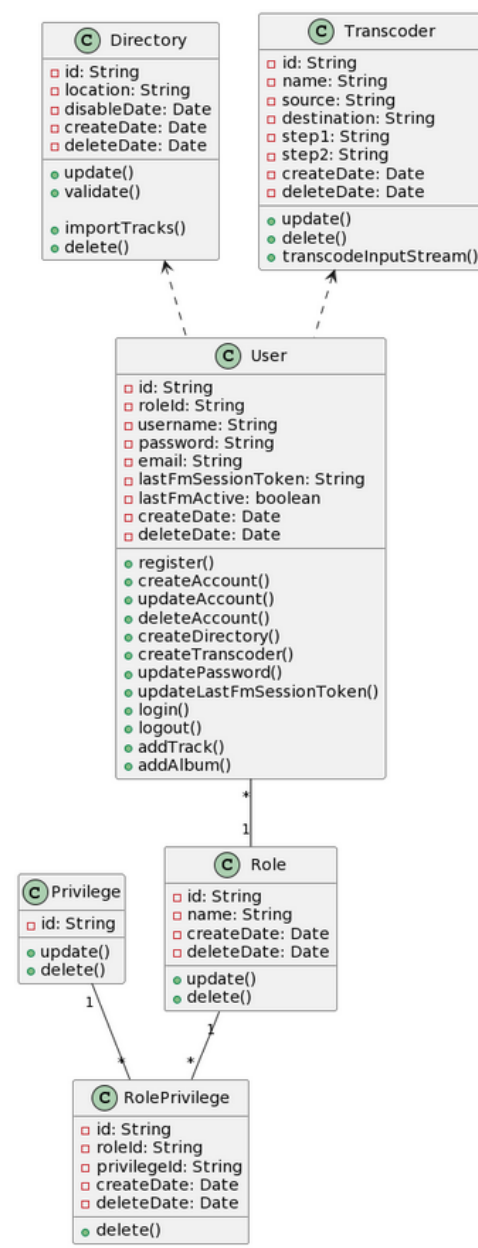
1. They can create, update and delete user accounts.
2. They can add new directories, delete them or change the local directory to which music is stored.
3. They can add, delete or update transcoders.

Behavior: The behavior of the system would be explained in terms of the above features.

1. The details regarding user creation, updation, and deletion have been mentioned in the User Management section. However, it does not mention how privilege checking is done. In order to check if a user is an admin, one check if the user's role (via **roleId**) has the “ADMIN” privilege (via the **Privilege** class). The corresponding privileges for each role are obtained through the **RolePrivilege** class. The features here and the relevant methods in the **User** class can only be accessed if the admin privileges are granted to the user.
2. The creation, updation, and deletion of directories and transcoders occur in a very similar manner to that of users. First, the privileges of the user are checked, and if the user is an admin, the supplied details are validated (in the case of creation and updation). For directories, validation checks whether the supplied path is actually a valid and writable local path. The requested action is performed if the user has the required privileges. Revalidation of the directory can

be done again (in the frontend via the “Rescan” option), hence the presence of a separate `validate()` method for the directory.

Bearing the above points in mind, the following is our proposed UML diagram for the subsystem.



A user has a single role, but a single role could have multiple users under it, eg. There can be several admins in the system. Since users can be grouped based on their roles, `User` and `Role` have an aggregation relationship (they are still independent entities). A role can be shared by many `RolePrivilege` objects, and so can a privilege, since a role can have multiple privileges, and a single privilege can be shared by multiple roles. The `User` class' (or specifically an admin account's) relationships with the `Directory` and `Transcoder` classes are slightly unconventional dependencies; they are of type `<<instantiate>>` since the `User` class cannot function without these classes in order to create objects of those classes. There is no conceptual or class link between them.

Transition Model

▼ Action Space (U)

1. Add directory
2. Delete Directory
3. Rescan directories
4. Add transcoder
5. Delete transcoder
6. Add user
7. Edit user
8. Delete user
9. Display logs
10. Sanity check

▼ Output Space (Y)

Directories, Transcoders, and users shown in their respective tabs with add, edit, and delete options. Sanity checks and logs in their respective tabs.

▼ States (X)

Directories (n), Transcoder (n), Users (n) for $n \in \mathbb{N}$



▼ Initial State (X_0)

Directories (0), Transcoder (0), Users (0)

▼ Display map (h)

1. State - $D(n)$

Scanned directories

Location	Active	Valid	Writable	
/home/pratham/code/SWE/Project_1	✓	✓	✓	
/home/pratham/code/SWE/Project_1/songs	✓	✓	✓	

Observable - Added Directories shown under the Directories tab with Active, Valid, Writable permissions

2. State - $U(n)$

Users management Add

Username	Create date
admin	2023-02-16
whatev	2023-02-22

Observable - Added users shown under the Users tab with edit permissions

3. State - $T(n)$

Transcoders

No transcoder

Observable - Added transcoders shown under the transcoder tab

▼ Transition Relation (f)

X	U	X'
$D(n), T(n), U(n)$	Add Directory	$D(n+1), T(n), U(n)$
$D(n), T(n), U(n)$	Delete Directory	$D(n-1), T(n), U(n)$
$D(n), T(n), U(n)$	Add Transcoder	$D(n), T(n+1), U(n)$
$D(n), T(n), U(n)$	Delete Transcoder	$D(n), T(n-1), U(n)$
$D(n), T(n), U(n)$	Add user	$D(n), T(n), U(n+1)$
$D(n), T(n), U(n)$	Delete user	$D(n), T(n), U(n-1)$
U_i	Edit user	U'_i
D_i	Rescan directories	D'_i

4. Library Management

This subsystem focuses on the main part of Music: playing, importing, and managing of music libraries and albums. This is what makes Music a music application.

Structure: The following are the classes associated with this subsystem.

- **User** : Represents a user in the system.
- **Track** : Represents a music track.
- **UserTrack** : Represents a user's relationship with a track.
- **Artist** : Represents a music artist.
- **Album** : Represents an album of tracks by an artist. It should be noted here that there is no specific method that creates an album. This is because an album is automatically created by the code when track information is added via

the frontend and read by the backend. Therefore, it can be said that album creation is actually done via the `addTrack()` function of the User class.

- `UserAlbum` : Represents a user's relationship with an album. Users can rate a particular score for albums.
- `Playlist` : Represents a user-created music playlist.
- `PlaylistTrack` : This represents the relationship between a playlist and a track.
- `Directory` : Represents a local directory where music is stored.
- `Transcoder` : Represents a transcoder entity.
- `ImportResource` : Contains functionality for importing music from local directories or external resources.
- `PlayerResource` : Contains functionality for managing music that one is currently playing.
- `Queue` : Represents a music queue for a particular user.

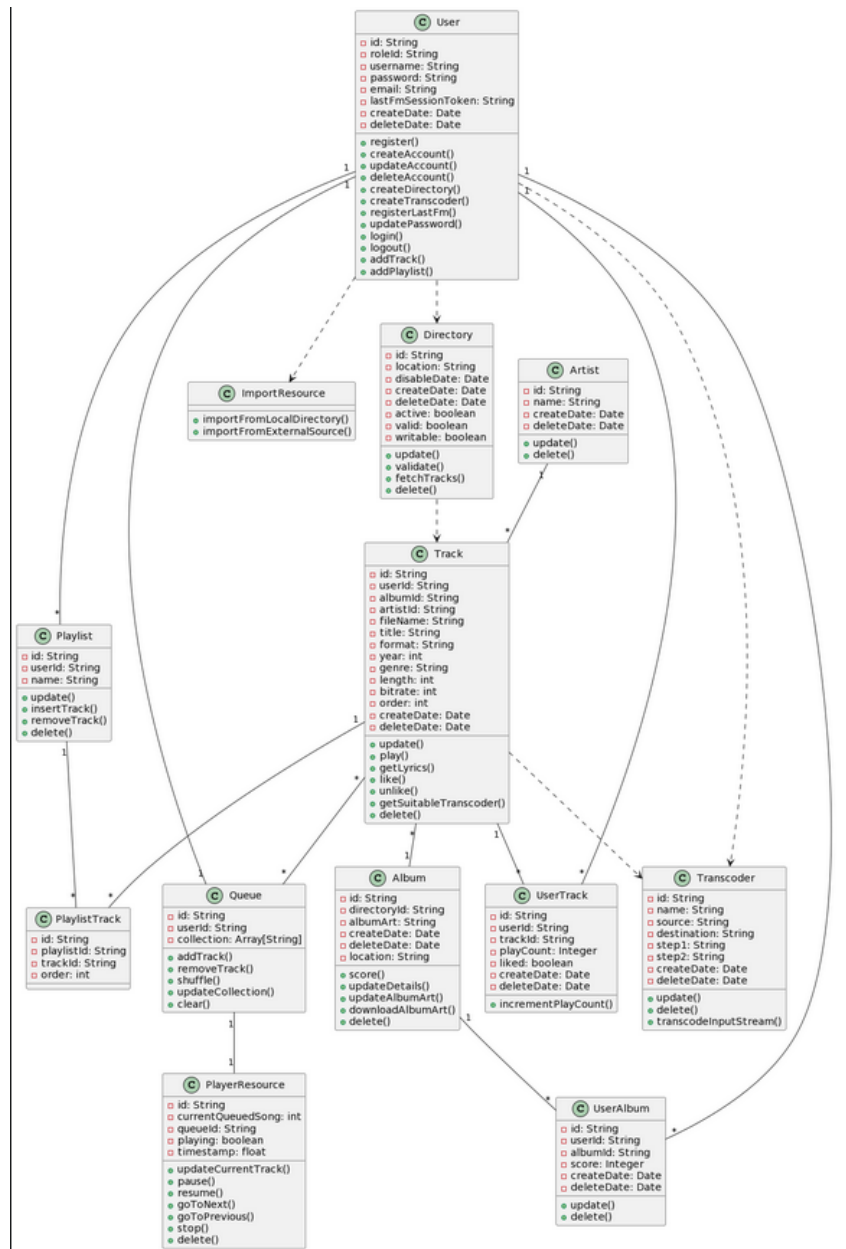
Functionality: The purpose of the subsystem is to do the following.

1. To play, pause and stop a music stream
2. To add, update and delete track, album, artist, and playlist information
3. To import music tracks from local directories or external sources
4. To store imported music in admin-added directories

Behavior: The behavior of the system would be explained in terms of the above features.

1. Initially, when a track is played, the music file stored in the local directory would be retrieved (with the help of `fetchTracks()` in the `Directory` class) and converted to a media stream by a suitable transcoder (found using `getSuitableTranscoder()`). The track would then be added to a created `Queue` object. A player controlled by `PlayerResource` would then be able to control this media stream via dedicated functions, and resume, stop or pause the stream, or skip ahead or back. As the user chooses to add more tracks or play a playlist/album instead, the queue would get updated accordingly, and can be shuffled or modified via `Queue`.
2. The user would be able to add/update/delete tracks, albums, artists and playlists via dedicated methods in the `User` class. Users can listen to or like/unlike tracks (the status of which is recorded by `UserTrack`) and rate a score for albums (recorded by `UserAlbum`). Moreover, metadata of tracks, playlists and albums can be updated (including album art).
3. When the user adds a track, it makes use of `ImportResource`, which contains dedicated methods to import music from a local directory or an external source (where the YouTube URL would be supplied). This would be done via `addTrack()`, which would then create a record of the track accordingly. The imported track would be added to a dedicated directory via the `addTrack()` method of the `Directory` class.

Based on the above, the following is our proposed UML diagram of this subsystem.



Transition Model

▼ Action Space (U)

1. Import Track/Playlist
2. Provide source Link/Zip file
3. Update Track/Playlist Details
4. Remove Track/Playlist
5. Update Playlist
6. Update Library
7. Play Track/Playlist

8. Like Track/Playlist
9. Unlike Track/Playlist
10. Increment Count
11. Stop Playing Track/Playlist

▼ States (X)

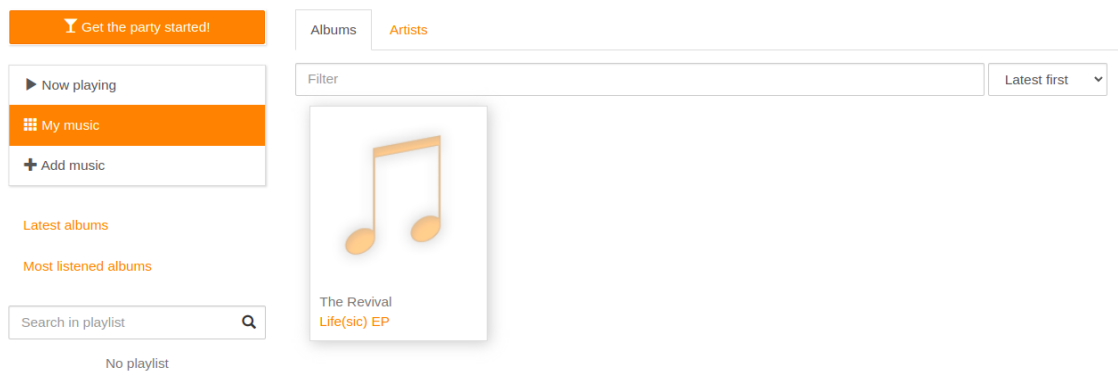
1. Library Home
2. Import music
3. Updating Details
4. Updating Playlist
5. Playing Track/Playlist
6. Updating Library

▼ Initial State (X0)

- Library Home

▼ Output Space (Y)

1. Homepage



2. Upload music page

Upload music from local computer

Single audio file or ZIP file containing audio files are supported.

Select a file...

3. Updating details

Add imported music to my collection

✓ Add all

#	Album artist	Album	Artist	Title
upload1327454307880019020.mp3				
2	The Revival	Life(sic) EP	The Revival	Juicy Stars

4. Playlists page

midnight

[▶ Play all](#) [↻ Shuffle](#) [+ Add all](#) [🗑 Delete](#)

Title	Artist	Album	🕒
Life(sic)	The Revival	Life(sic) EP	2:02 ♥

5. Music

Sismics Music

Search your music

admin ⚙ 🔒 Logout

⚠ You haven't changed your default admin password. Secure your server by changing the default password now. [Dismiss](#)

🔥 Get the party started!

▶ Now playing

📁 My music

➕ Add music

Latest albums

Most listened albums

Search in playlist

Fav

The Revival Life(sic) EP

played 12 times

[▶ Play all](#) [↻ Shuffle](#) [+ Add all](#) [✎ Edit tags](#) [➕ Add to playlist](#)

#	Title	🕒
1	My Sword Got A Story To Tell	3:08
2	Juicy Stars	4:04
3	Samurai & Bullshit	2:59
4	Life(sic)	2:02

Change album art

0:22/2:03sec

⏮ ⏪ ⏩ ⏭

6. Library settings

[▶ Play all](#) [↻ Shuffle](#) [+ Add all](#) [✎ Edit tags](#) [➕ Add to playlist](#)

▼ Display Map (h)

X	Y
library home	Homepage
import music	Upload music page
updating	Add imported music
Playing Playlist/Album	Scrobbling Playlist/Album
playing track/playlist	Music
updating Library	Library Settings

▼ Transition Relation (f)

X	U	X'
Library Home	Import Track/Playlist	Updating Details
Updating Details	Provide Source / Zip file, Update Track/playlist Details	Updating Library
Library Home	Update Library	Updating Library
Updating Library	Add Track/Playlist	Updating Details
Updating Library	Remove Track/Playlist	Library Home
Updating Library	Update Playlist	Updating Details
Library Home	Play Track/Playlist	Playing Track/Playlist
Playing Track/Playlist	Update Count	Playing Track/Playlist
Playing Track/Playlist	Stop Playing Track/Playlist	Library Home
Library Home	Like Track/Playlist	Library Home
Library Home	Unlike Track/Playlist	Library Home

Strengths and Weaknesses of the System

Strengths:

1. Because of the role-based privilege system in the user management feature of Music, it becomes very easy to grant users privileges and create relationships between roles and privileges, as opposed to having a specific set of roles and a specific set of privileges for them. Not only does this help with security, but the way this has been implemented makes it flexible enough to be able to selectively grant privileges to users in any manner.
2. The way the system has been implemented, particularly the class-based structure shown via the UML diagrams, appears to have a general pattern of “Class A—Intermediary Class—Class B”. The presence of the intermediary class makes it much easier to represent and manage relationships between A and B, rather than including an attribute in either of the classes to represent a relationship with the other (which means that deletion of objects would result in invalid references at times). In order to manage all aspects of a relationship between A and B, one has to simply modify the intermediary class, and this would be difficult to do with attribute-based relationship representation.
3. The usage of data access points provides an abstract interface to the use of the database through DBIs that makes the code both modular and encapsulated at the backend.

Weaknesses:

1. The user management system only allows the creation/updation/deletion of new user accounts through the admin account. This can be a security risk if the admin account becomes compromised.
2. No alerts are raised at the UI end on the success of actions like adding entities.