Diagrams Spotiflop

Chitam Harone
Leyx-valade Hugo

Table of contents :

ER Diagram :

The Entity-Relationship (ER) Diagram models the relationships between various entities in the system. In this diagram:

Users can have one or more playlists.

Each **Playlist** contains multiple songs.

Songs are created by an **Author** and belong to a specific **Genre**.
This diagram provides a conceptual understanding of how data entities (e.g., users, playlists, songs) are linked.

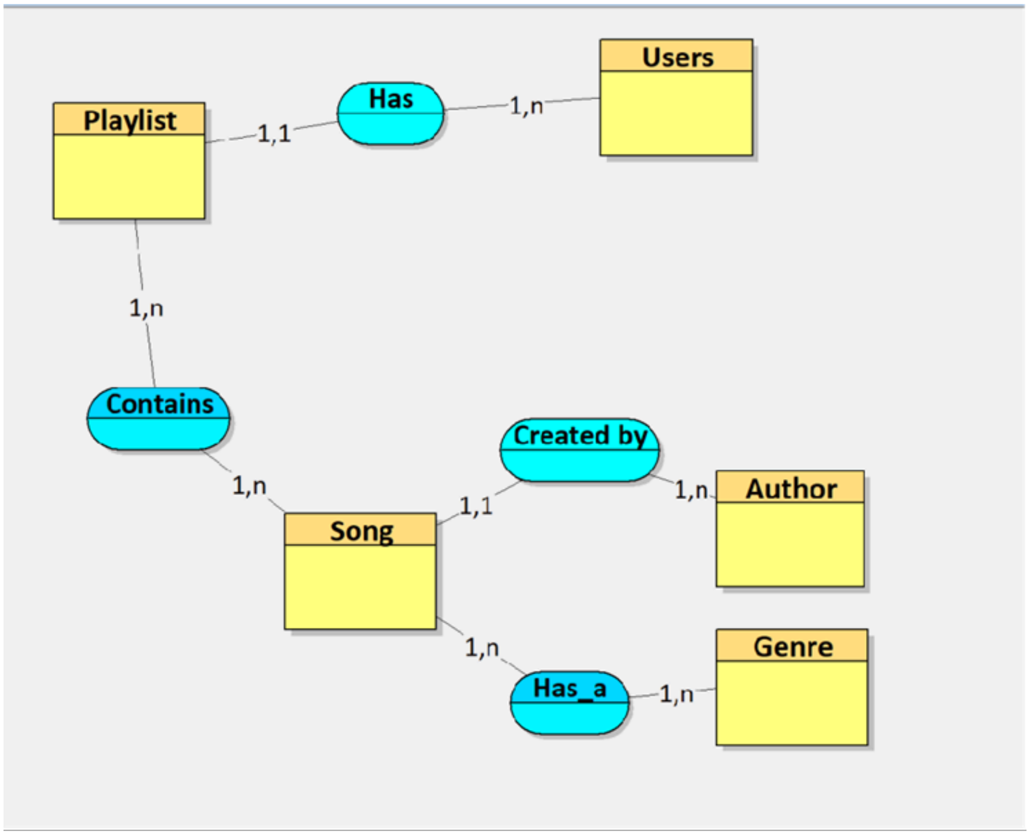


Table Structure Diagram :

This ER (Entity-Relationship) diagram illustrates the structure of the playlist management system, focusing on the relationships between entities and the attributes of each entity:

1. **User**: Represents the users of the system.

Attributes:

- id_user: Primary key (unique identifier for each user).
- username, first_name, last_name, email, password, role, and date_of_birth.

Relationships:

- A user can create multiple playlists (1 User → n Playlists).

2. **Playlist**:

Represents a collection of songs managed by a user.

Attributes:

- id_playlist: Primary key.
- title, date_of_post, number_of_save, description, state.

Relationships:

- A playlist belongs to one user (1 Playlist → 1 User).
- A playlist can contain multiple songs via the playlist_has_songs junction table (1 Playlist → n Songs).

3. **Song**:

Represents individual songs.

Attributes:

- id_song: Primary key.
- title, duration, number_of_streams, date_of_post, lyrics.

Relationships:

- A song is created by one author (1 Song → 1 Author).
- A song belongs to one genre (1 Song → 1 Genre).
- A song can belong to multiple playlists (1 Song → n Playlists via playlist_has_songs).

4. **Author**:

Represents the creator of a song.

Attributes:

- id_author: Primary key.
- alias, first_name, last_name, biography, verified.

Relationships:

- An author can create multiple songs (1 Author → n Songs).

5. **Genre**:

Represents the category or type of a song.

Attributes:

- id_genre: Primary key.
- name.

Relationships:

- A genre can be associated with multiple songs (1 Genre → n Songs).

6. **PlaylistHasSong**:

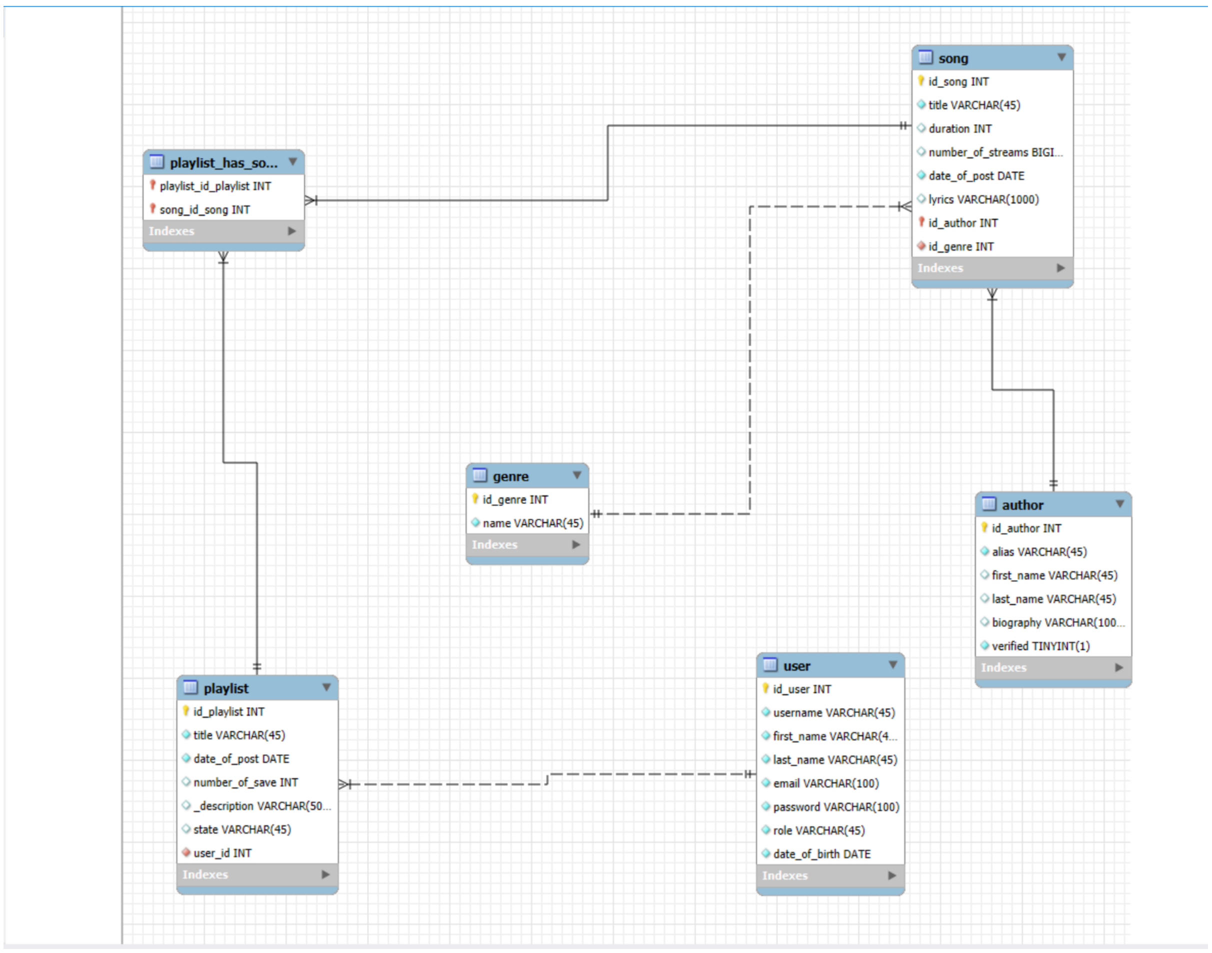
Represents the many-to-many relationship between playlists and songs.

Attributes:

- playlist_id: Foreign key referencing id_playlist.
- song_id: Foreign key referencing id_song.

Relationships:

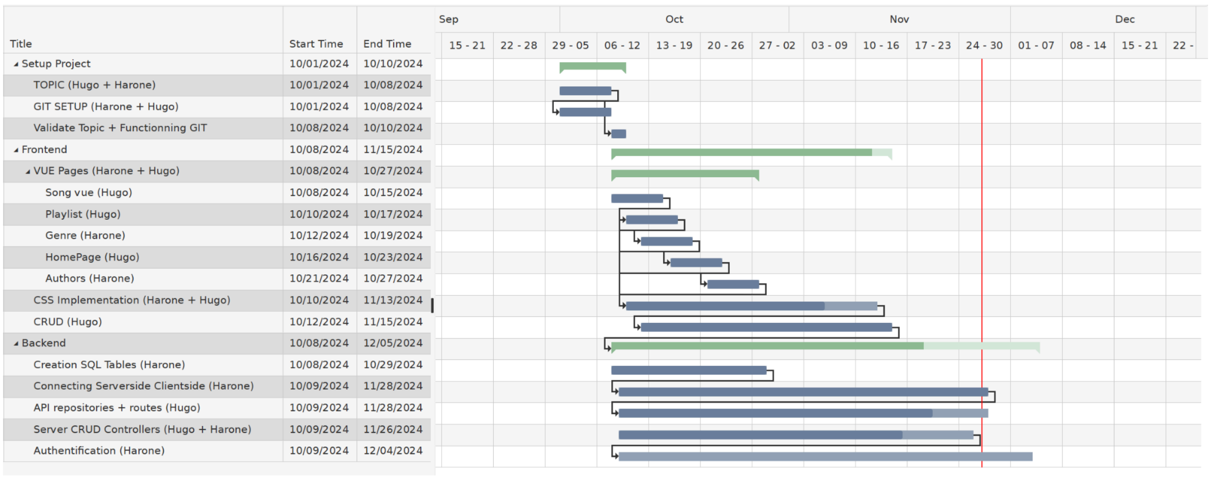
- Links multiple playlists to multiple songs.



Gantt Diagram :

The Gantt Diagram provides a timeline view of the project's tasks and milestones. It includes:Tasks: Each task is represented as a horizontal bar with a start and end date (e.g., GIT Setup, Frontend development). Dependencies: Some tasks are linked to indicate that one task must finish before another begins. Team Members: Tasks are assigned to specific team members (e.g., Hugo, Harone).

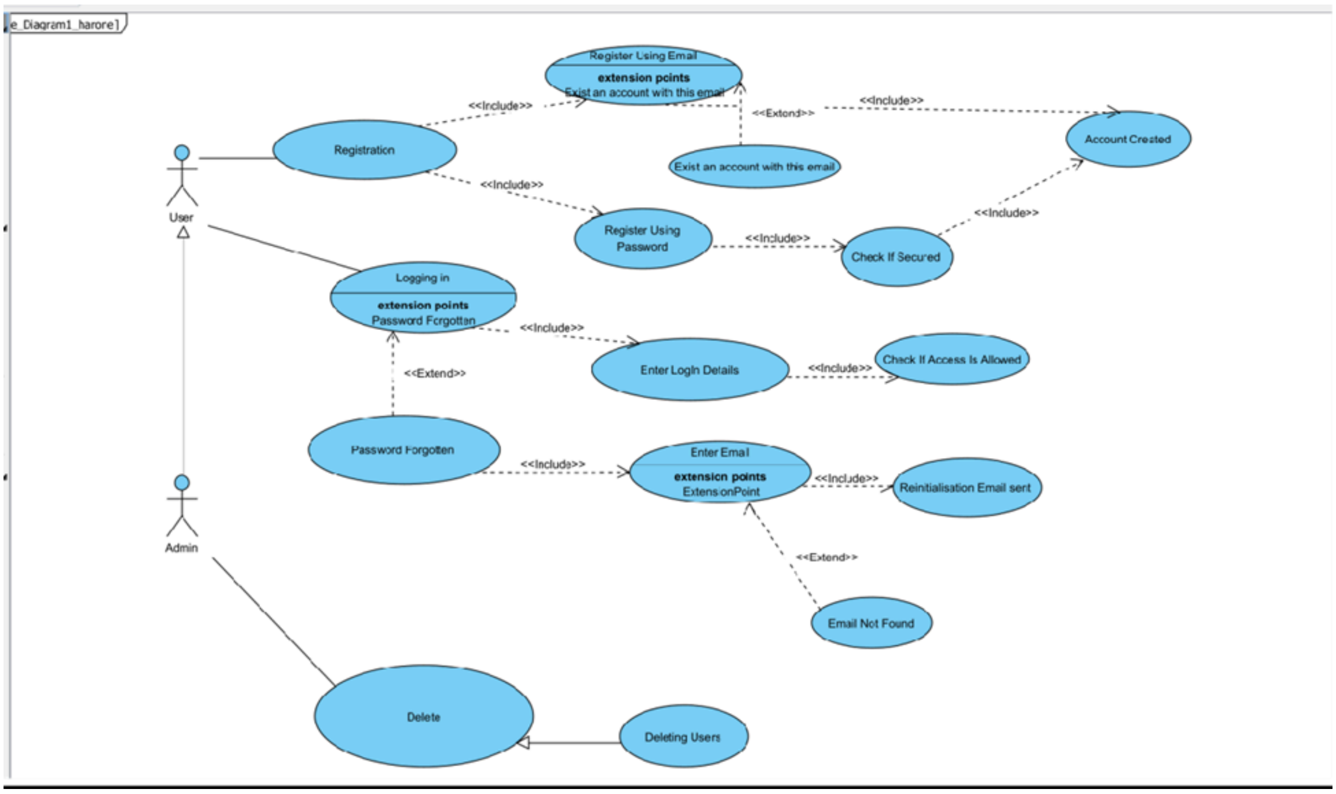
This chart is useful for tracking progress, managing workloads, and ensuring the project stays on schedule. It visually communicates deadlines and overlaps between tasks, helping coordinate the team's efforts effectively.



Title	Start Time	End Time
◢ Setup Project	10/01/2024	10/10/2024
TOPIC (Hugo + Harone)	10/01/2024	10/08/2024
GIT SETUP (Harone + Hugo)	10/01/2024	10/08/2024
Validate Topic + Functionning GIT	10/08/2024	10/10/2024
◢ Frontend	10/08/2024	11/15/2024
◢ VUE Pages (Harone + Hugo)	10/08/2024	10/27/2024
Song vue (Hugo)	10/08/2024	10/15/2024
Playlist (Hugo)	10/10/2024	10/17/2024
Genre (Harone)	10/12/2024	10/19/2024
HomePage (Hugo)	10/16/2024	10/23/2024
Authors (Harone)	10/21/2024	10/27/2024
CSS Implementation (Harone + Hugo)	10/10/2024	11/13/2024
CRUD (Hugo)	10/12/2024	11/15/2024
◢ Backend	10/08/2024	12/05/2024
Creation SQL Tables (Harone)	10/08/2024	10/29/2024
Connecting Serverside Clientside (Harone)	10/09/2024	11/28/2024
API repositories + routes (Hugo)	10/09/2024	11/28/2024
Server CRUD Controllers (Hugo + Harone)	10/09/2024	11/26/2024
Authentification (Harone)	10/09/2024	12/04/2024

Use Case Diagram (Authentification) :

This diagram shows the different interactions users and administrators can have with the authentication system. Users can register, log in, and recover forgotten passwords. Admins can delete users or manage system configurations.
The diagram uses relationships like <<include>>: it shows something that will happen everytime and <<extend>> to show optional and mandatory interactions.

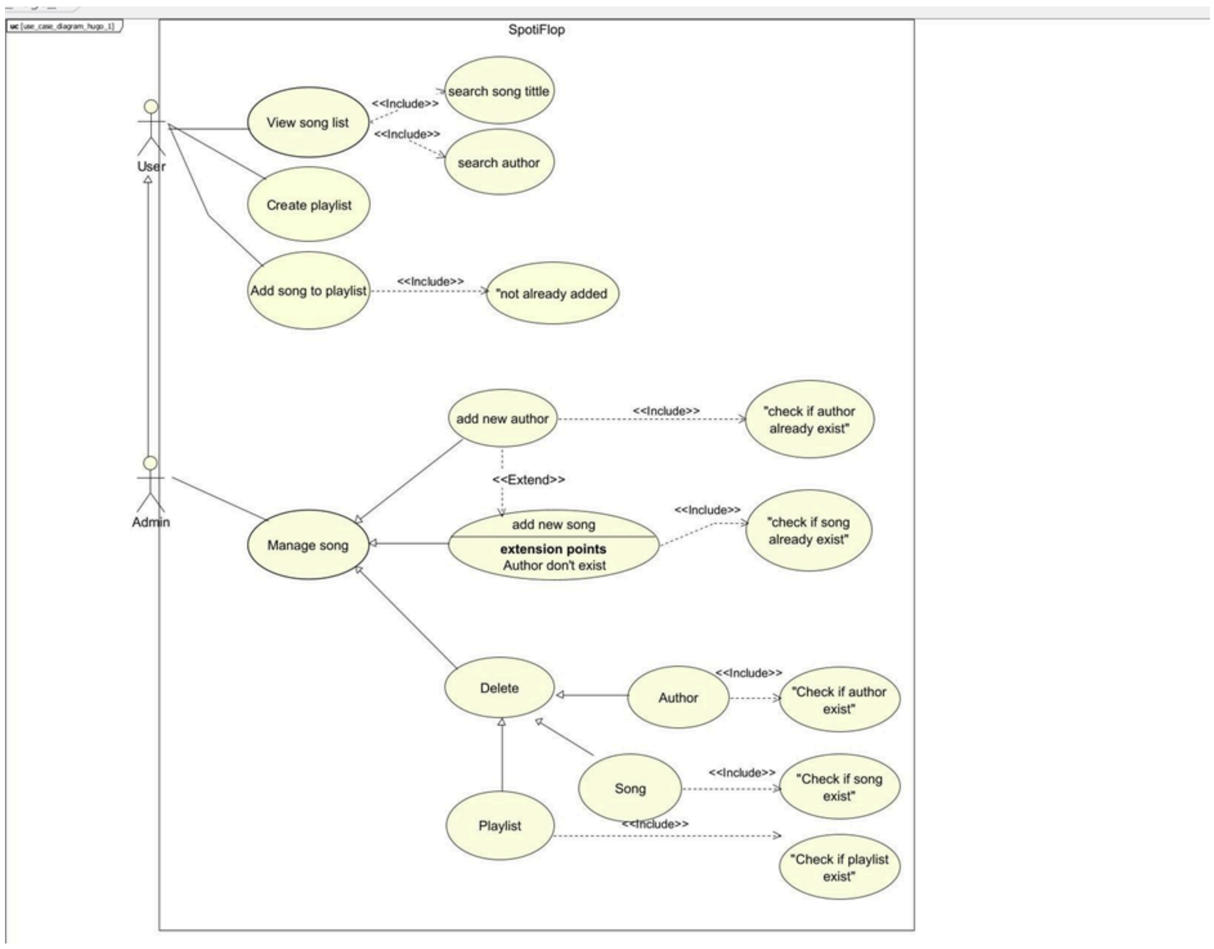


Use Case Diagram (Interaction User/Admin with Songs) :

The diagram represents how a user and an admin can manage a song. Both have distinct managing of IT:

-users can view a list of song, search one by the title and search an author, create a playlist and add a song to a playlist

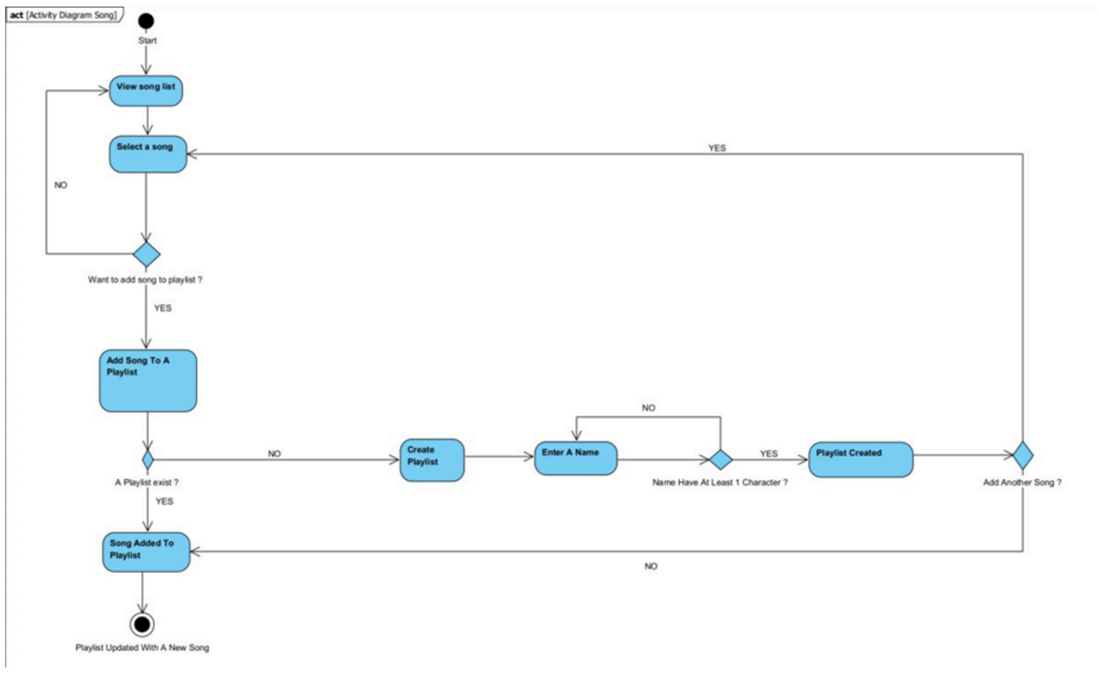
-admin manages by adding one, adding an author, delete an author a song or a playlist.



Activity Diagram (Adding Song To Playlist) :

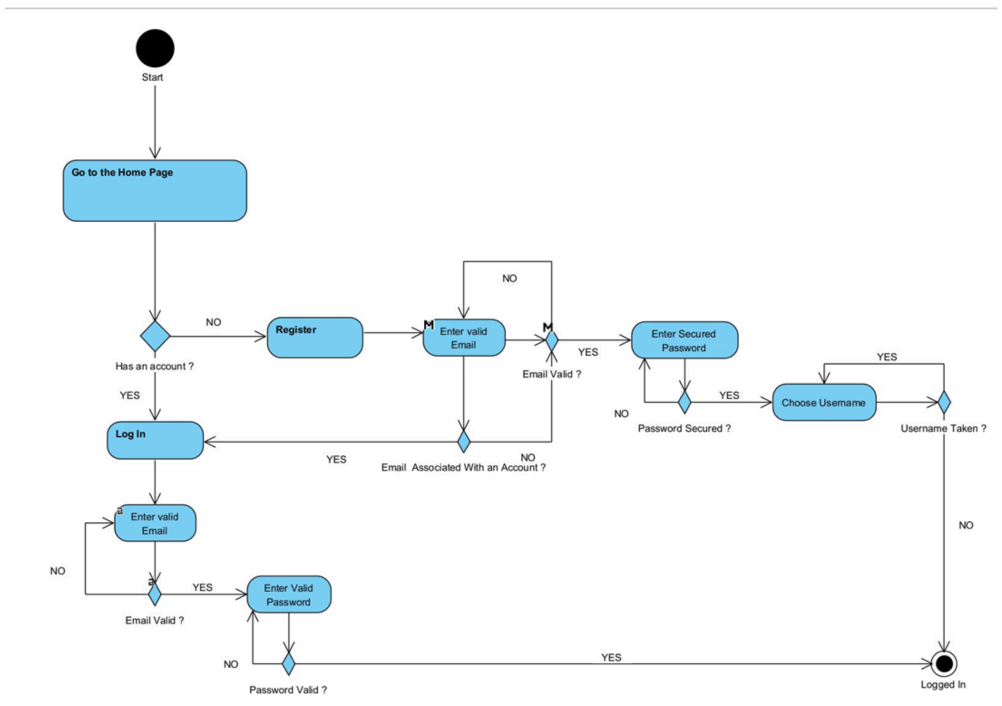
This activity diagram illustrates the process of adding a song to a playlist. The flow begins when a user selects a song:

The system checks if the user wants to add it to a playlist. If no playlist exists, the user can create one. The song is then added to the playlist.
This diagram highlights user decisions and the system's responses step-by-step.



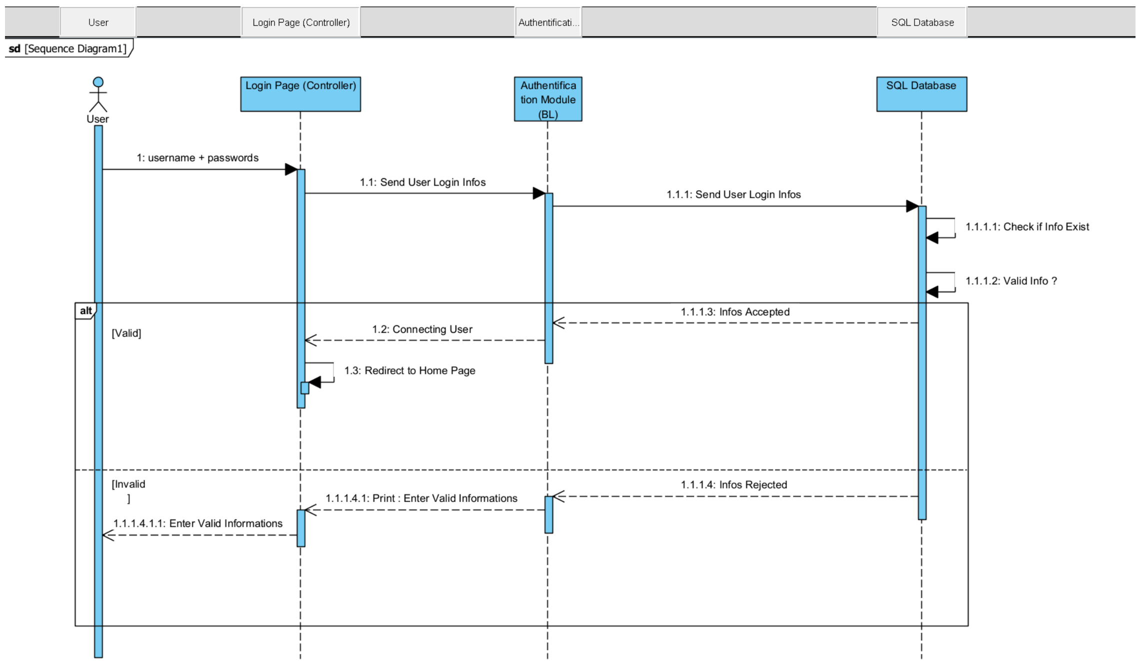
Activity Diagram (Login) :

This diagram shows step-by-step the process of user login. It starts with the user choosing whether to login or to register. For login, the system validates the email and password. For registration : the system checks if the email is unique (not already used) and if the password meets the necessary security.



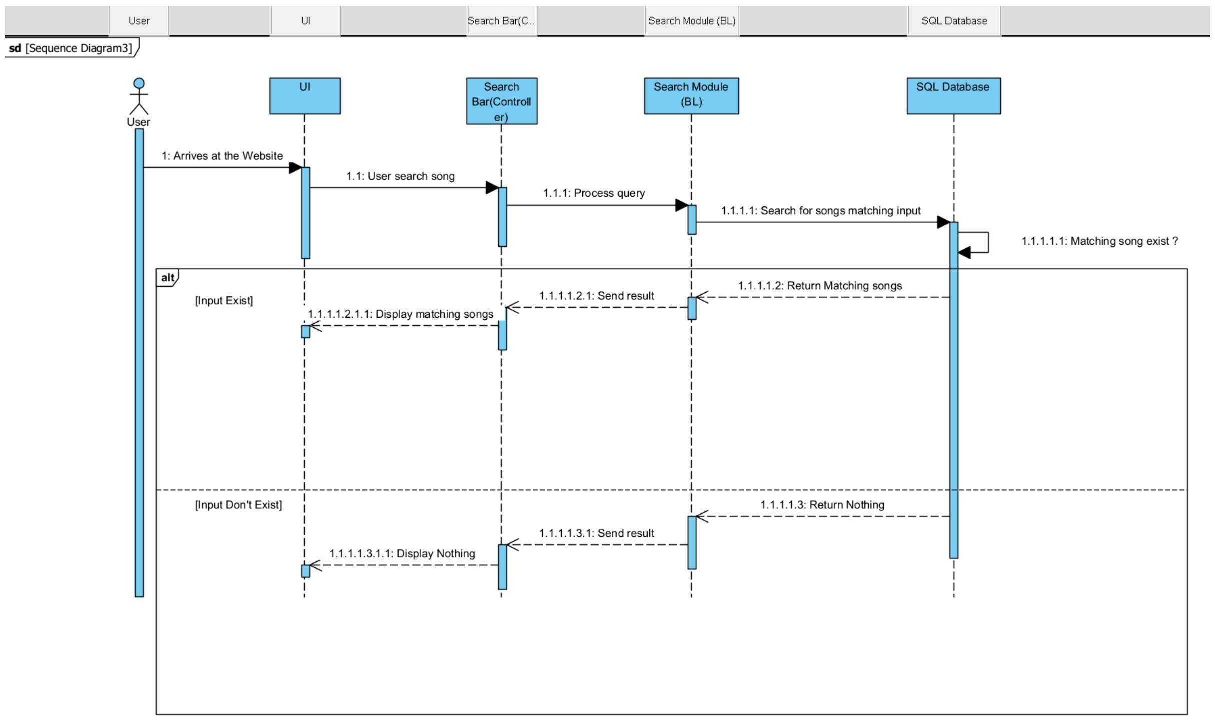
Sequence Diagram (Login) :

This diagram details the sequence of interactions between the user, login page, authentication module, and database:The user provides login credentials. The authentication module validates the credentials using the database. The system either grants access or rejects the login attempt. It showcases the system's logic flow during user login.



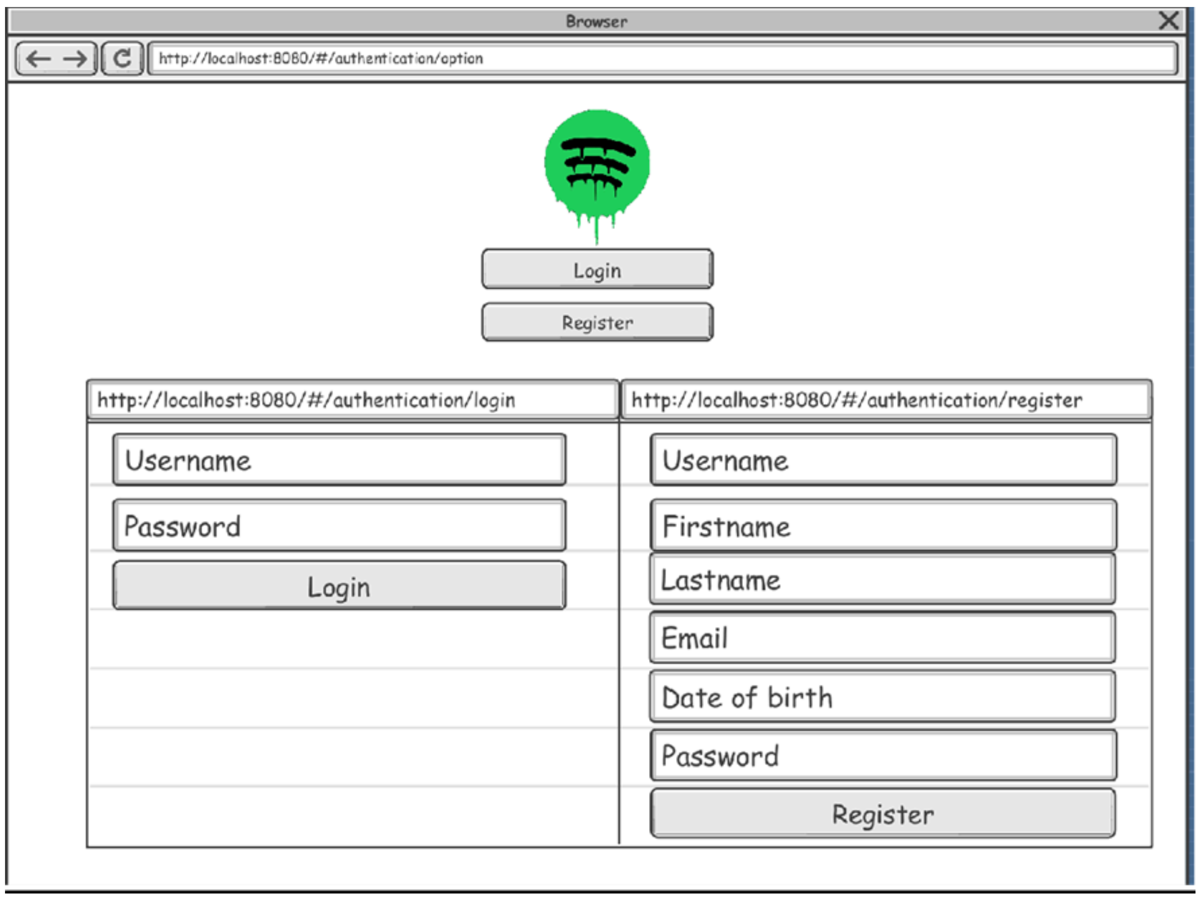
Sequence Diagram (Search Song) :

This sequence diagram illustrates how a song is being searched from the UI to the backend. If a matching song exists in the database it will display the song in the UI, else it will display nothing.



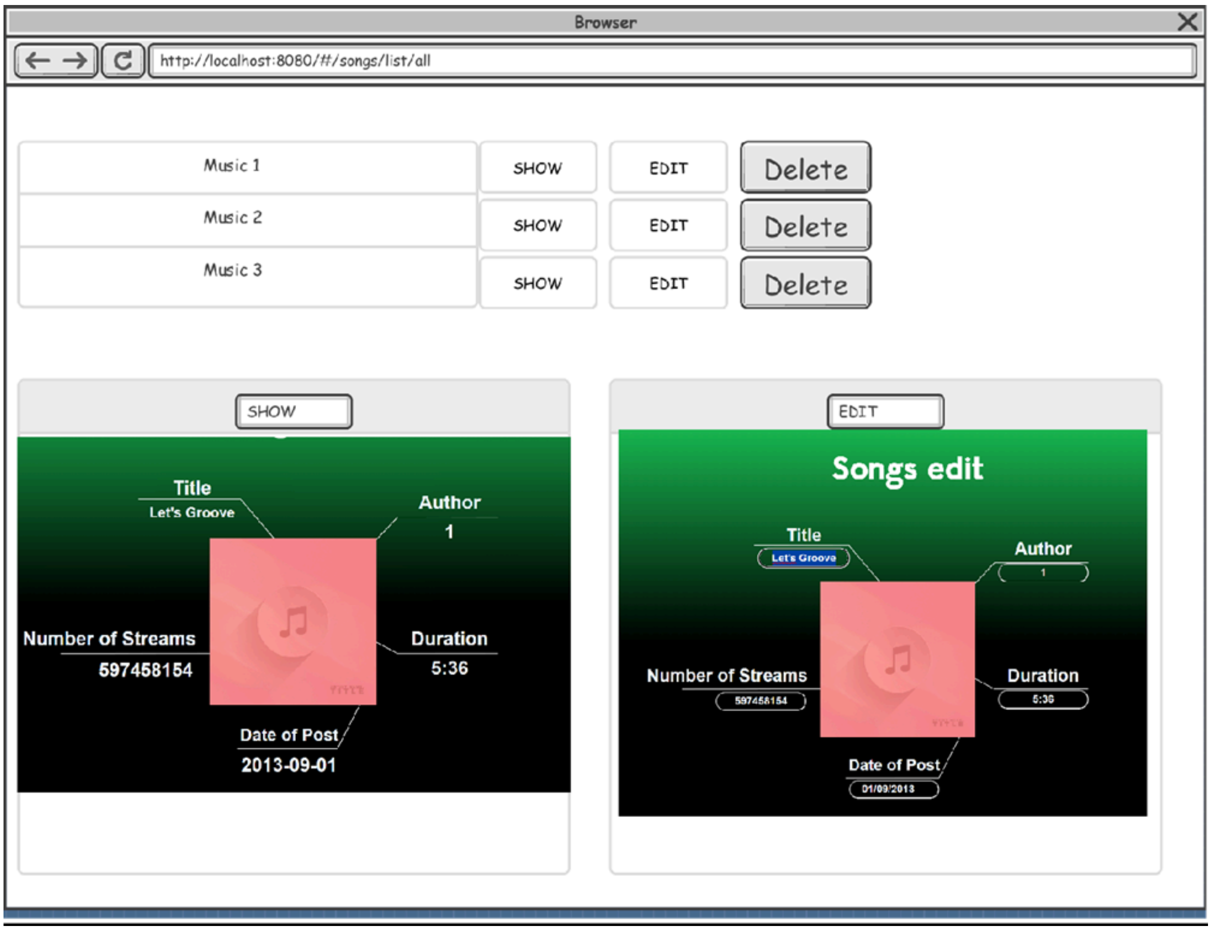
Wireframe Diagram (Login + Register) :

The wireframe shows the user interface for login and registration: Users can enter credentials to log in or fill in details (name, email, password) to register.It provides a visual layout of the forms and options available on the authentication pages


Browser
http://localhost:8080/#/authentication/option
Login
Register
http://localhost:8080/#/authentication/login
Username
Password
Login
http://localhost:8080/#/authentication/register
Username
Firstname
Lastname
Email
Date of birth
Password
Register

Wireframe Diagram (Songs) :

This wireframe depicts the interface for managing songs: Users can view a list of songs, edit details, or delete songs. Detailed views for individual songs include information like title, author, and duration.
This design focuses on usability and efficient song management.

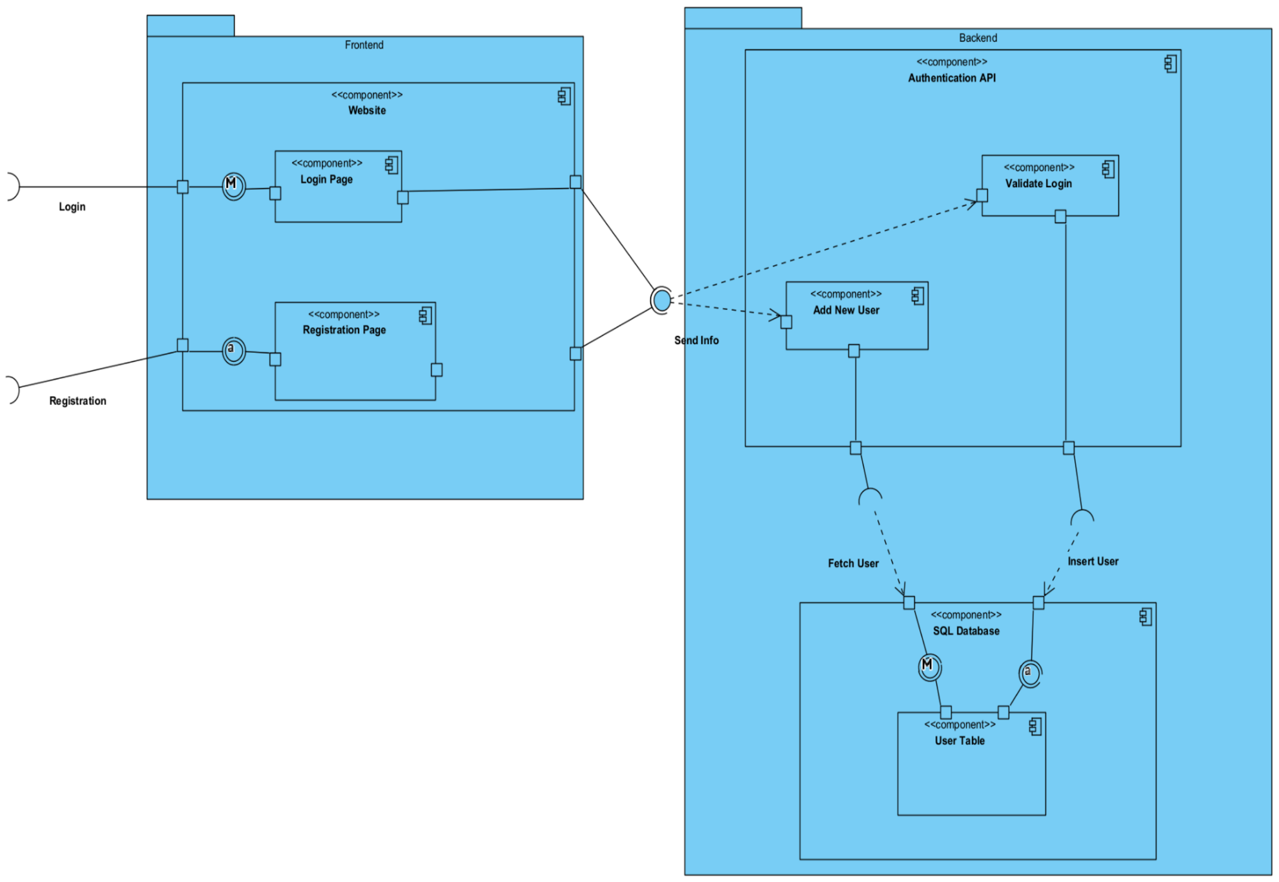


Component Diagram Authentication:

This component diagram illustrates the authentication process within the system. The Frontend Module includes two main components: the Login Page, where users enter their credentials to access the system, and the Registration Page, where new users can register by providing their details. These frontend components interact with the Authentication API, which contains two key services: Validate Login, responsible for verifying user credentials, and Add New User, which handles the registration process.

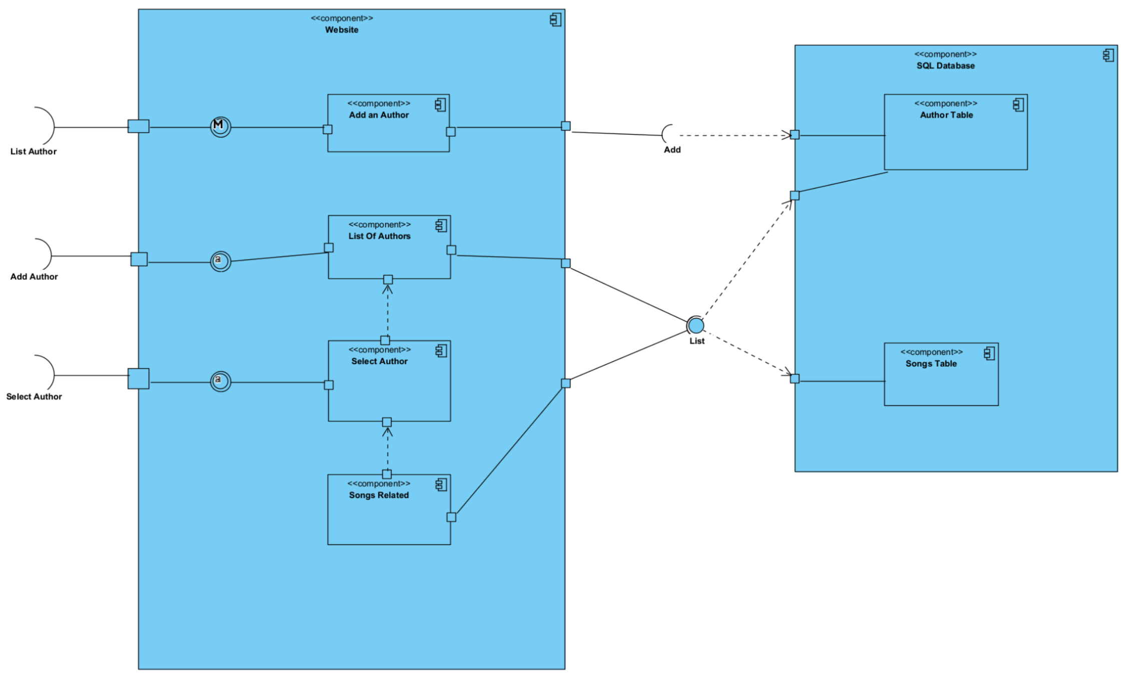
The Authentication API interacts with the Database Module, specifically the User Table, to perform operations such as inserting new user data during registration or fetching user records to validate login credentials. Connections like "Insert User" and "Fetch User" clearly represent these operations.

This diagram demonstrates a well-organized and modular authentication flow, highlighting how the frontend, backend, and database collaborate to handle user authentication securely and efficiently.



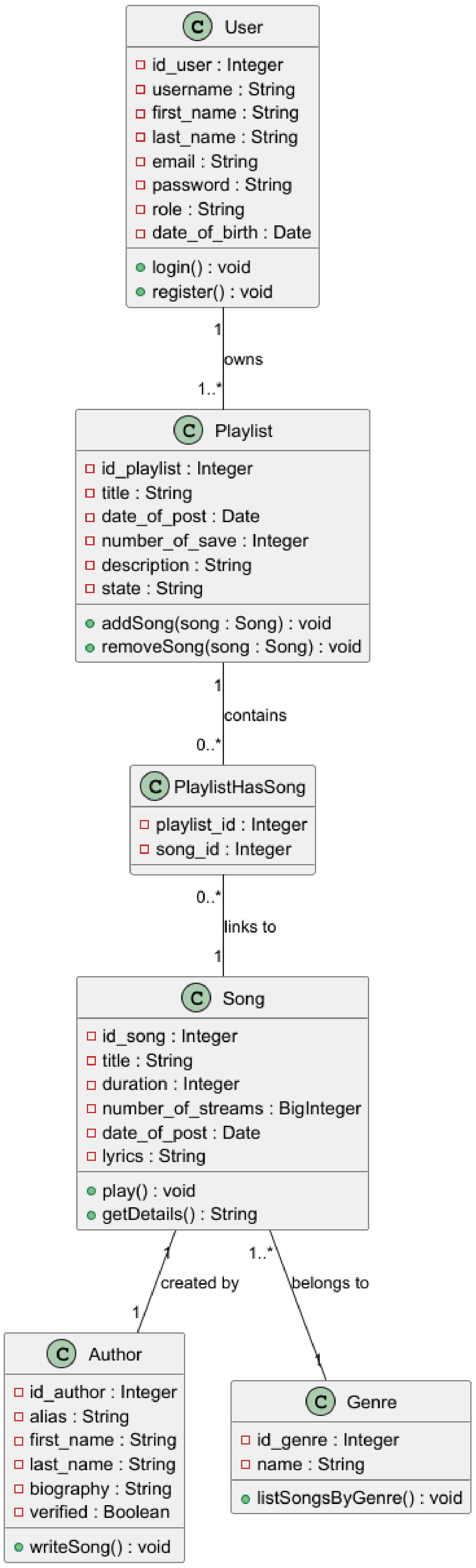
Component Diagram Author :

This component diagram illustrates how the system manages authors and songs. The frontend module includes several components: "Website," which acts as the main interface for users; "Add an Author," allowing users to add new authors to the database; "List of Authors," which fetches and displays all authors from the database; "Select Author," enabling users to choose a specific author to view or manage their details; and "Songs Related," which retrieves songs associated with a selected author. The backend module includes the "SQL Database" that stores all data, with specific components like "Author Table," which handles operations such as adding or fetching author details, and "Songs Table," which manages song data and retrieves songs related to authors. Interactions between the frontend and backend components are well-defined, ensuring smooth communication for actions like adding authors, fetching lists, and retrieving songs



Class Diagram :

- This class diagram represents the structure of the playlist management system, showcasing the relationships between the core entities:
- **User**: Represents the users of the system, who can create and manage playlists. Each user can own one or more playlists.
- **Playlist**: Represents a collection of songs managed by a user. Each playlist belongs to one user and can contain multiple songs.
- **Song**: Represents an individual song with attributes like title, duration, and lyrics. Each song is associated with one author and belongs to one genre.
- **Author**: Represents the creator of a song. An author can create one or more songs, and each song must have exactly one author.
- **Genre**: Represents the category of a song. Each song belongs to one genre, and a genre can group multiple songs.
- **PlaylistHasSong**: Represents the many-to-many relationship between playlists and songs. It allows a song to appear in multiple playlists and a playlist to contain multiple songs.



User
id_user : Integer
username : String
first_name : String
last_name : String
email : String
password : String
role : String
date_of_birth : Date
login() : void
register() : void
1
owns
1..*
Playlist
id_playlist : Integer
title : String
date_of_post : Date
number_of_save : Integer
description : String
state : String
addSong(song : Song) : void
removeSong(song : Song) : void
1
contains
0..*
PlaylistHasSong
playlist_id : Integer
song_id : Integer
0..*
links to
1
Song
id_song : Integer
title : String
duration : Integer
number_of_streams : BigInteger
date_of_post : Date
lyrics : String
play() : void
getDetails() : String
1
created by
1
1..*
belongs to
1
Author
id_author : Integer
alias : String
first_name : String
last_name : String
biography : String
verified : Boolean
writeSong() : void
Genre
id_genre : Integer
name : String
listSongsByGenre() : void