Diagrams Spotiflop

Chitam Harone
Leyx-valade Hugo

Table of contents :

Introduction :

Music is a subject that can be created by many artists with their own style. Furthermore, since music is made and listened to worldwide, it was necessary to categorize them into different categories and genres to make it easier for listeners to search for them. Our project aims for a user to create multiple playlists and add 1 or many songs to them. Below you will find multiple diagrams to explain all the features our website could do.

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.

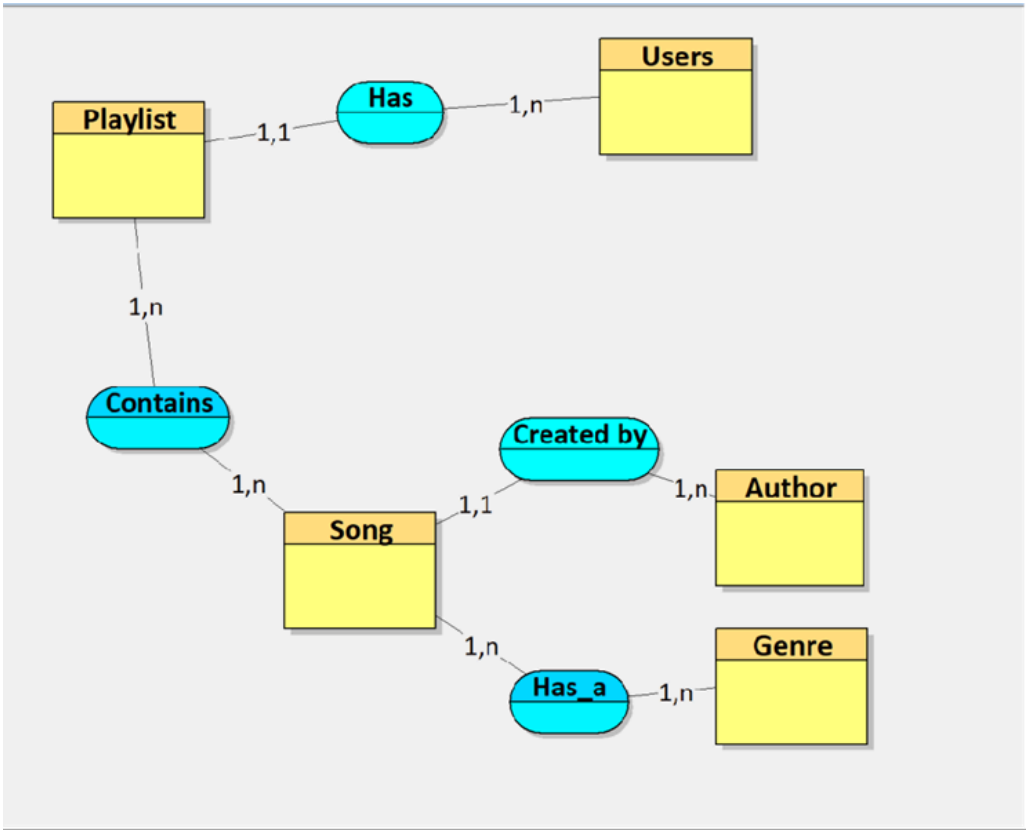
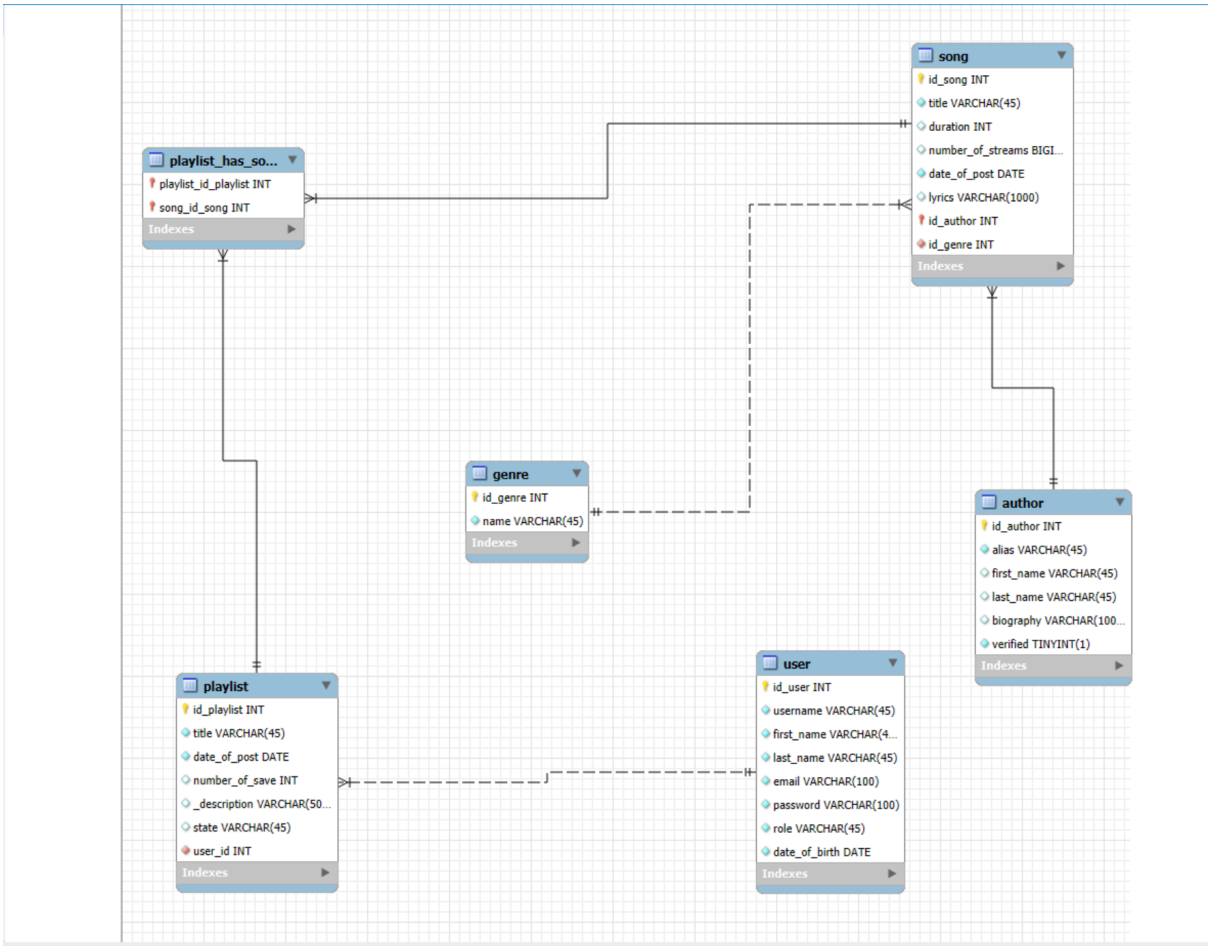


Table Structure Diagram :

This ER diagram represents the structure of the playlist management system, showing the relationships between key entities. Users can create multiple playlists, with each playlist belonging to one user and containing multiple songs through a many-to-many relationship managed by a junction table. Songs are created by a single author, belong to a specific genre, and can appear in multiple playlists. Authors can create multiple songs, while genres group multiple songs into categories. The PlaylistHasSong table links playlists and songs, facilitating the many-to-many relationship and ensuring modular data management.



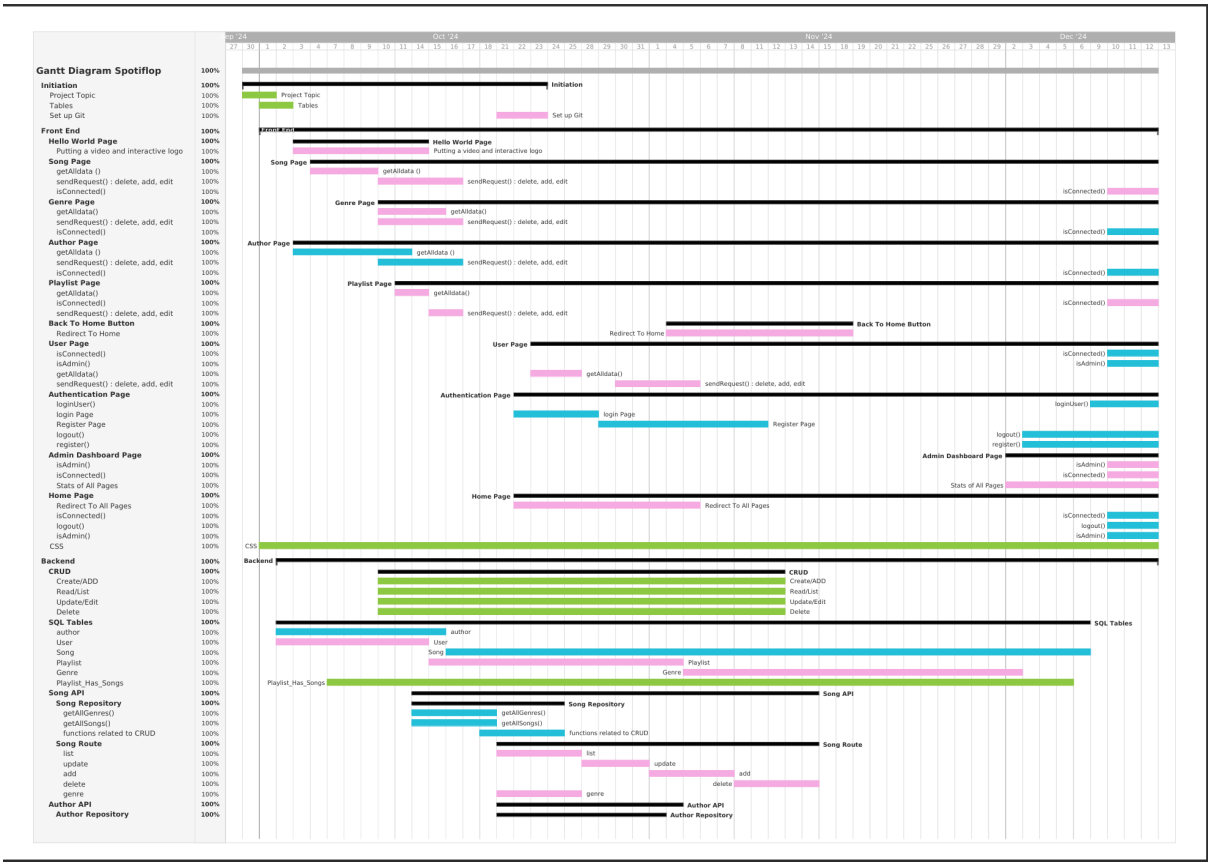
Gantt Diagram :

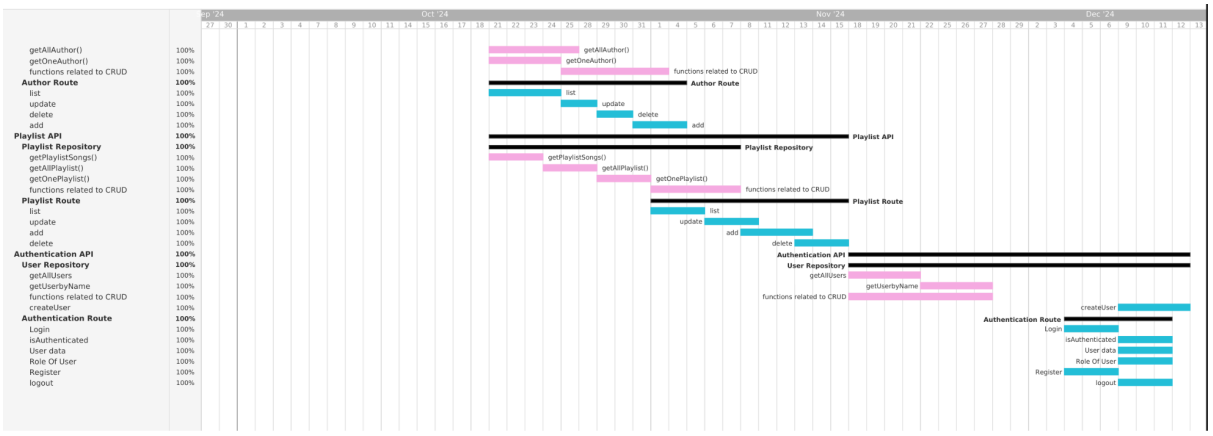
The Gantt Diagram provides a timeline view of the project's tasks and milestones. It includes:Tasks: Each task is represented as a horizontal bar with a start and end date (e.g., GIT Setup, Frontend development). Dependencies: Some tasks are linked to indicate that one task must finish before another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

The green color represents both of US
Blue color : Harone

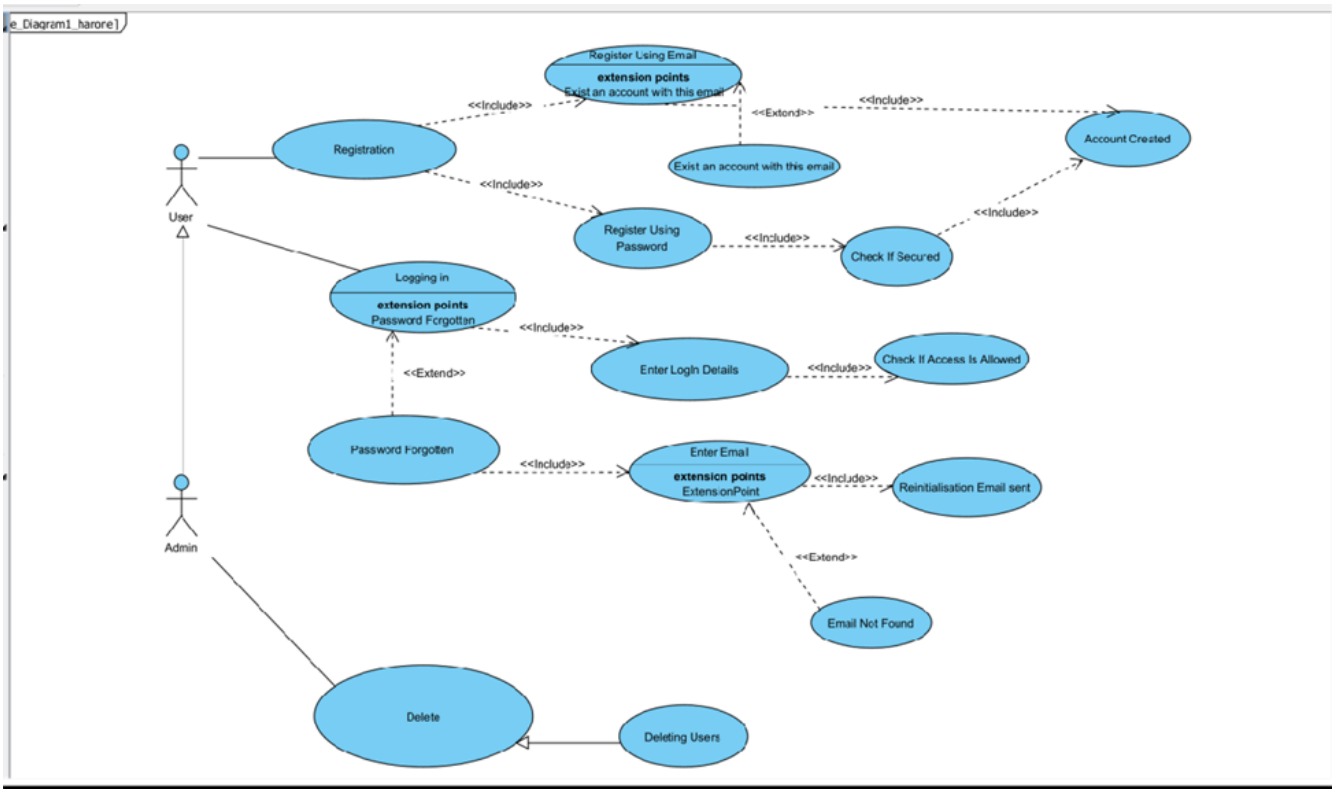
Pink color : Hugo





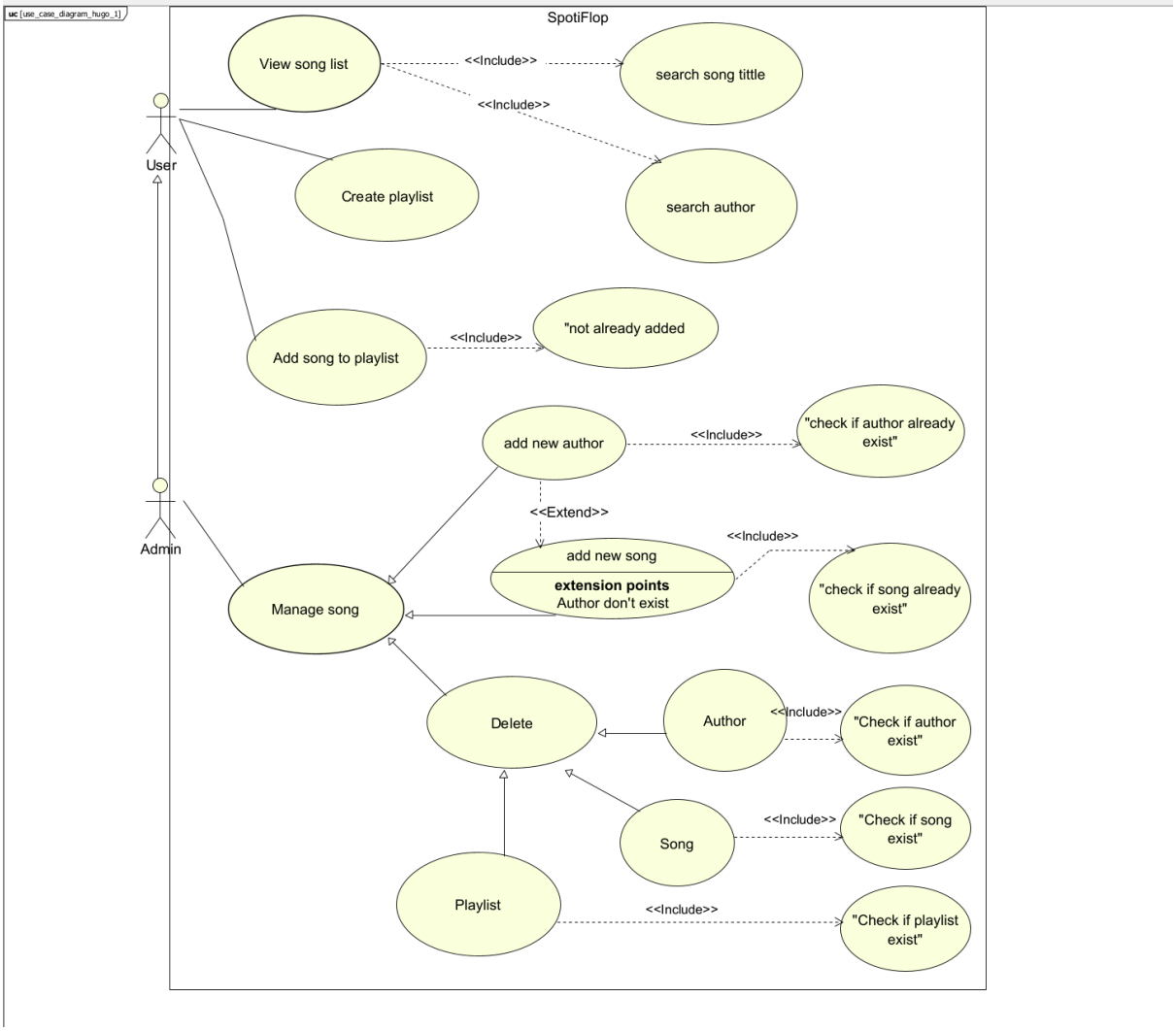
Use Case Diagram (Authentication) :

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) :

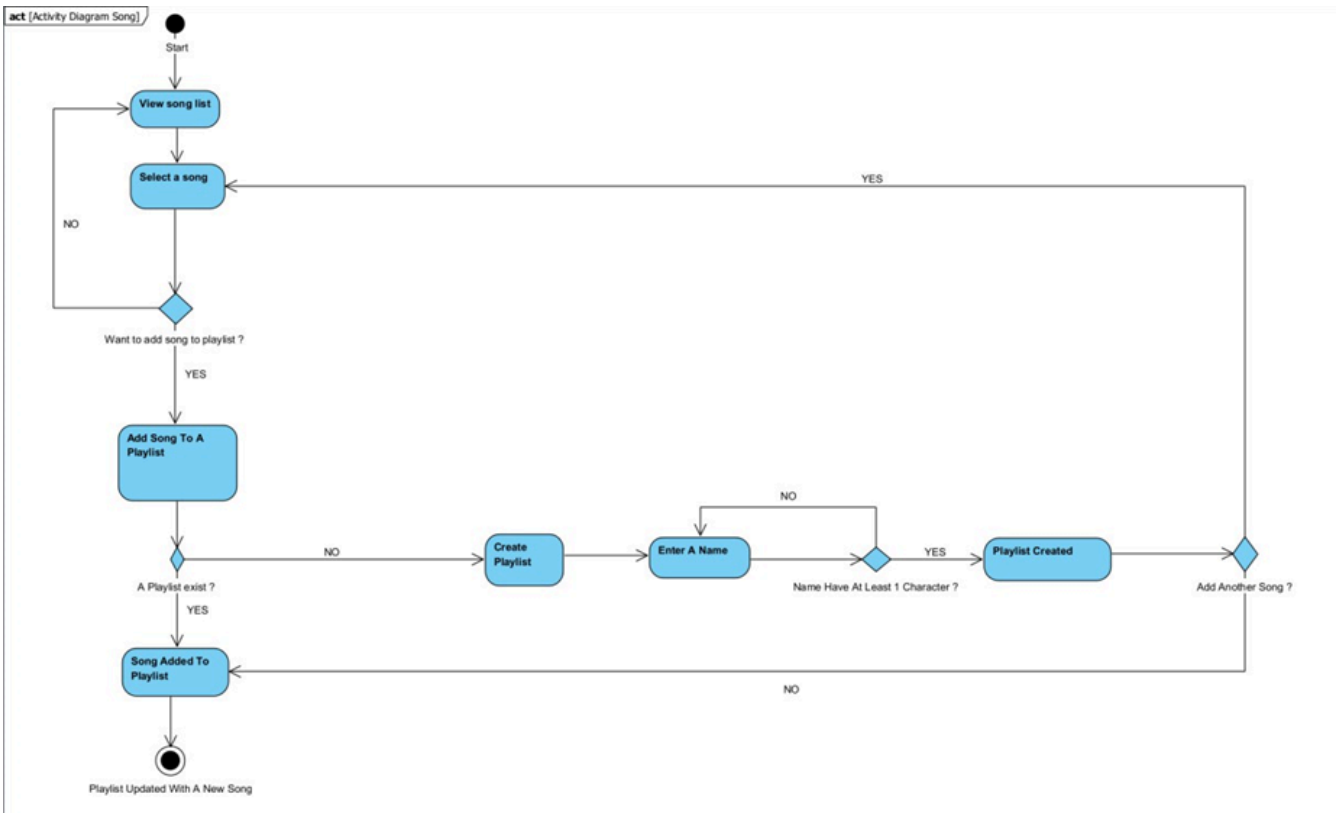
The diagram represents how a user and an admin can manage a song. Both have distinct managing of IT:
-users can view a list of song, search one by the title and search an author, create a playlist and add a song to a playlist
-admin manages by adding one, adding an author, delete an author a song or a playlist.



Activity Diagram (Adding Song To Playlist) :

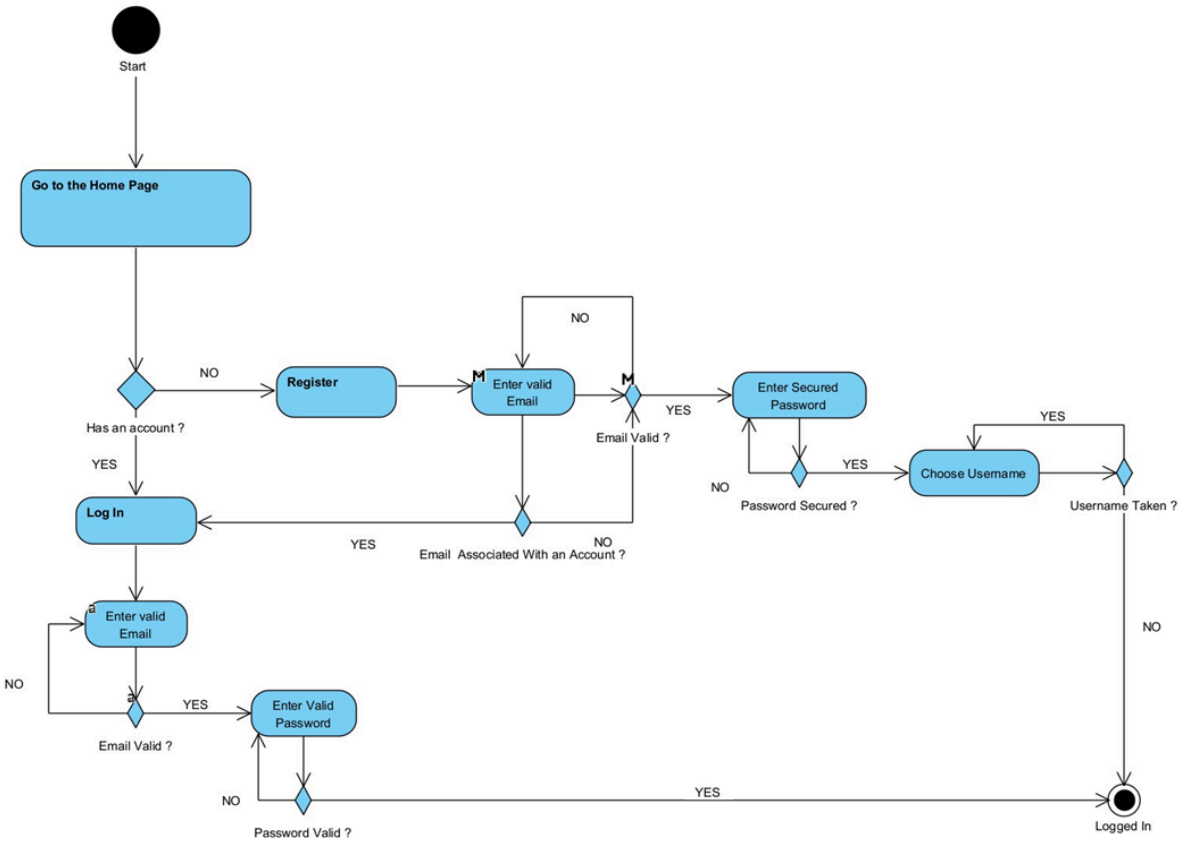
This activity diagram illustrates the process of adding a song to a playlist. The flow begins when a user selects a song:

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.



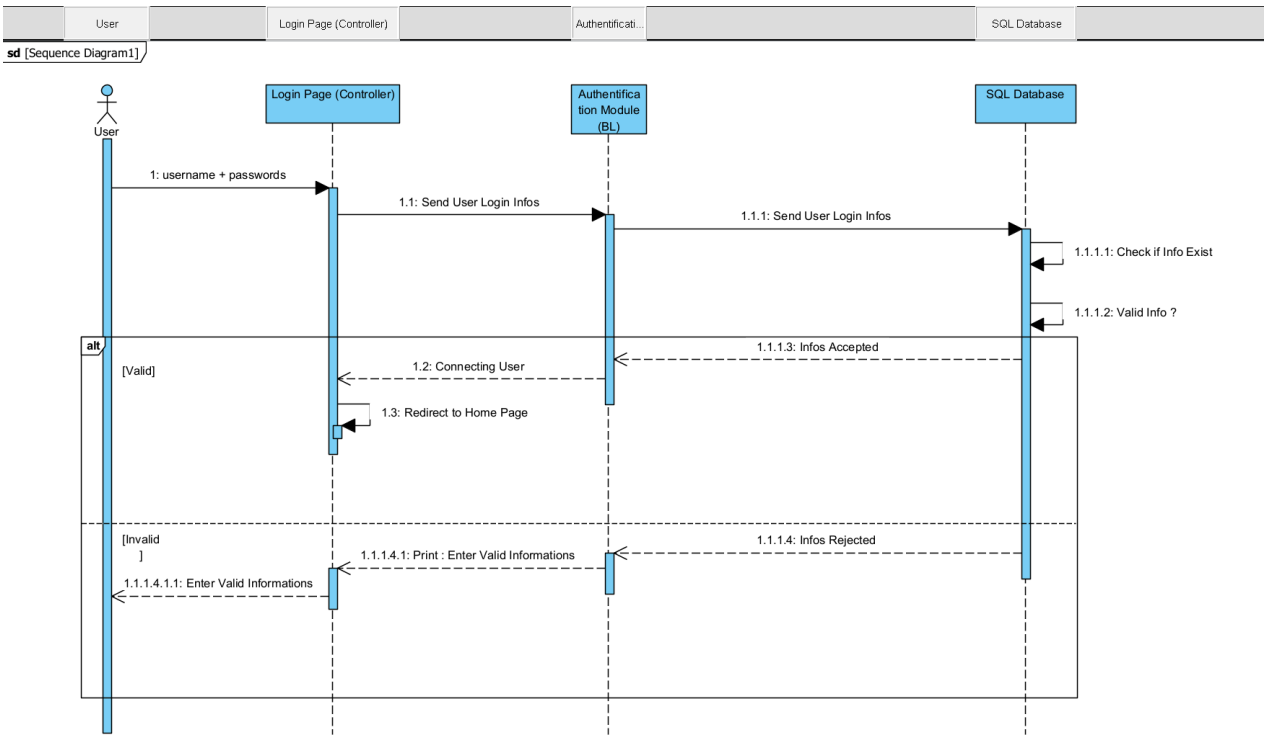
Activity Diagram (Login) :

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.



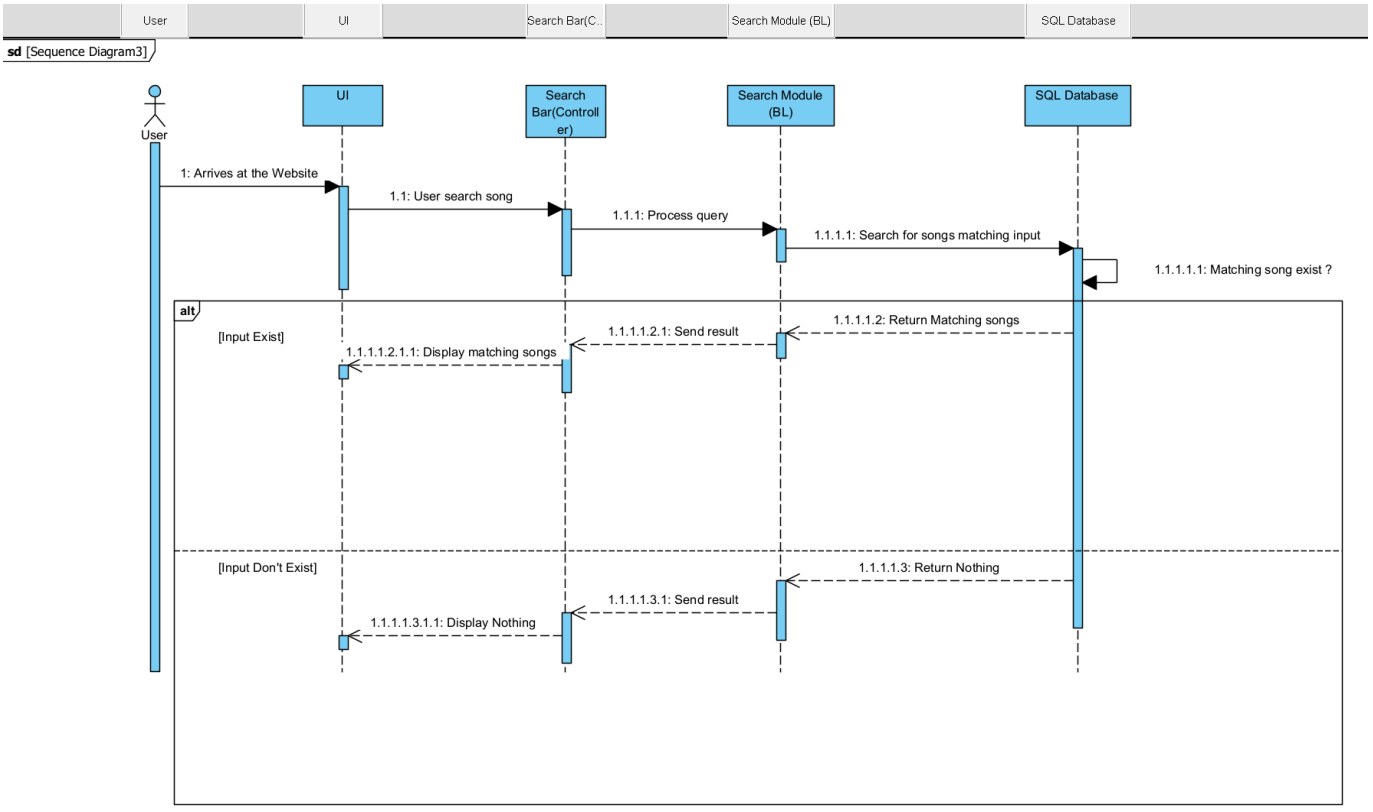
Sequence Diagram (Login) :

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.



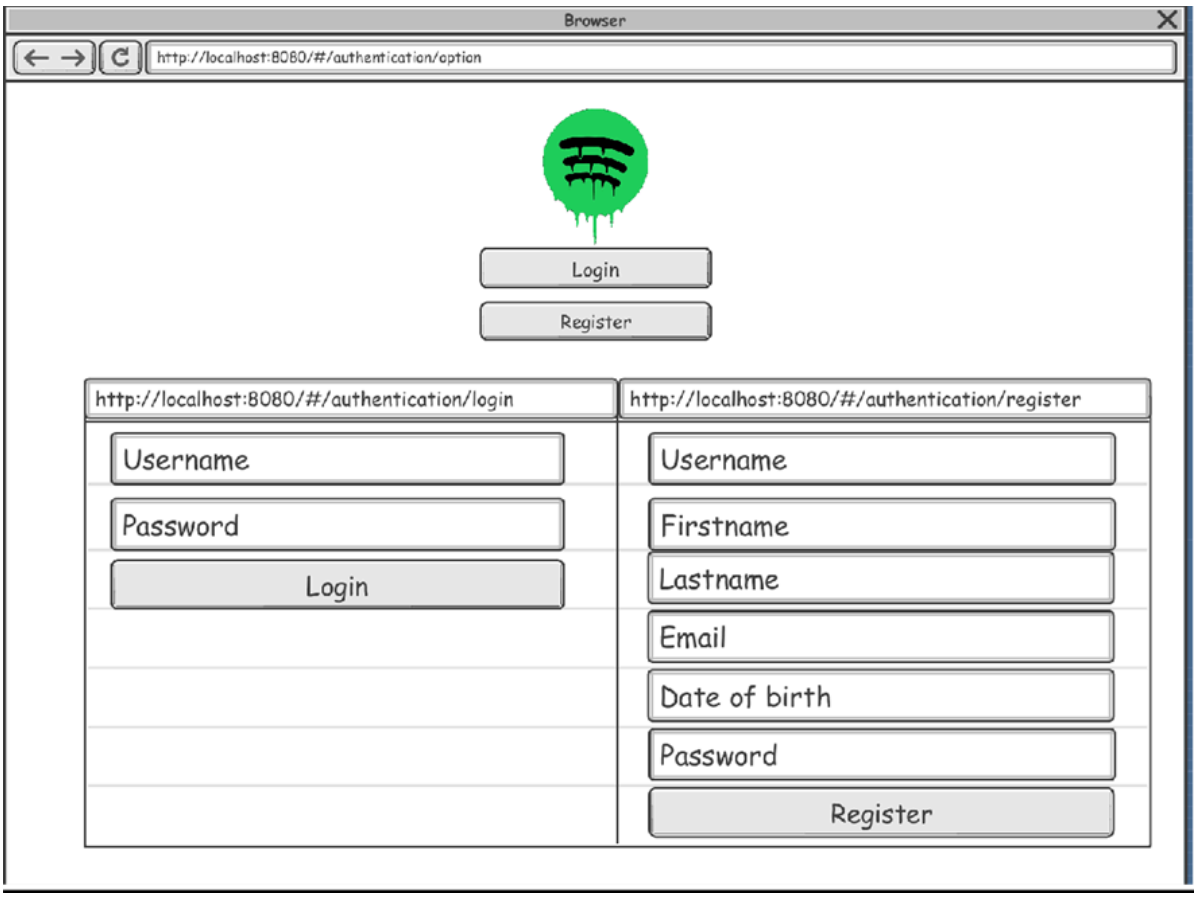
Sequence Diagram (Search Song) :

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.



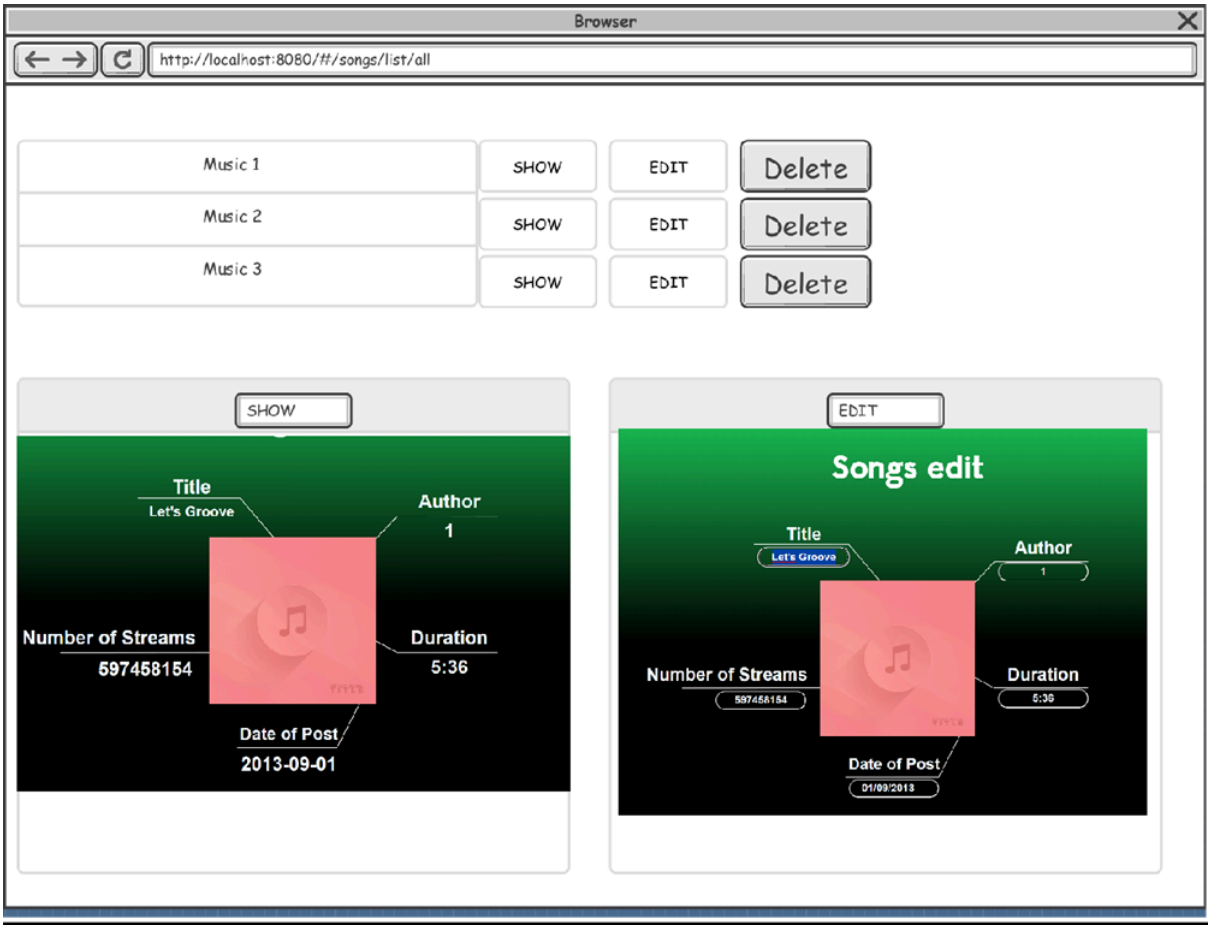
Wireframe Diagram (Login + Register) :

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



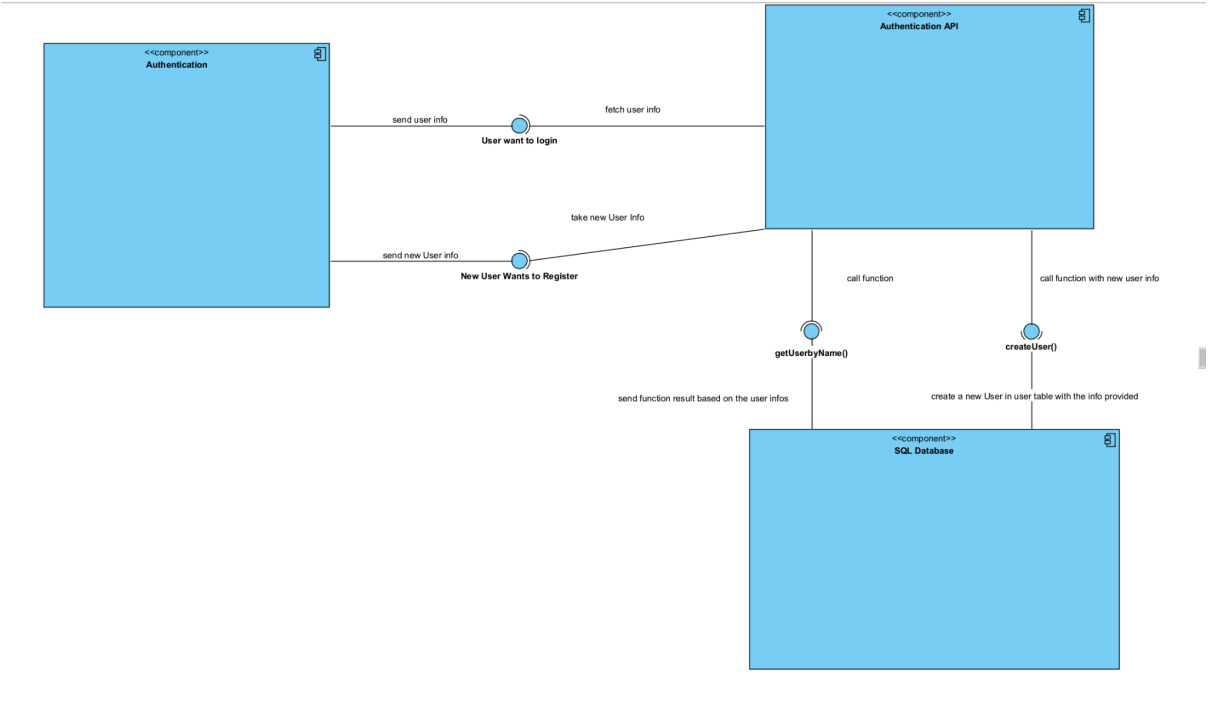
Wireframe Diagram (Songs) :

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.



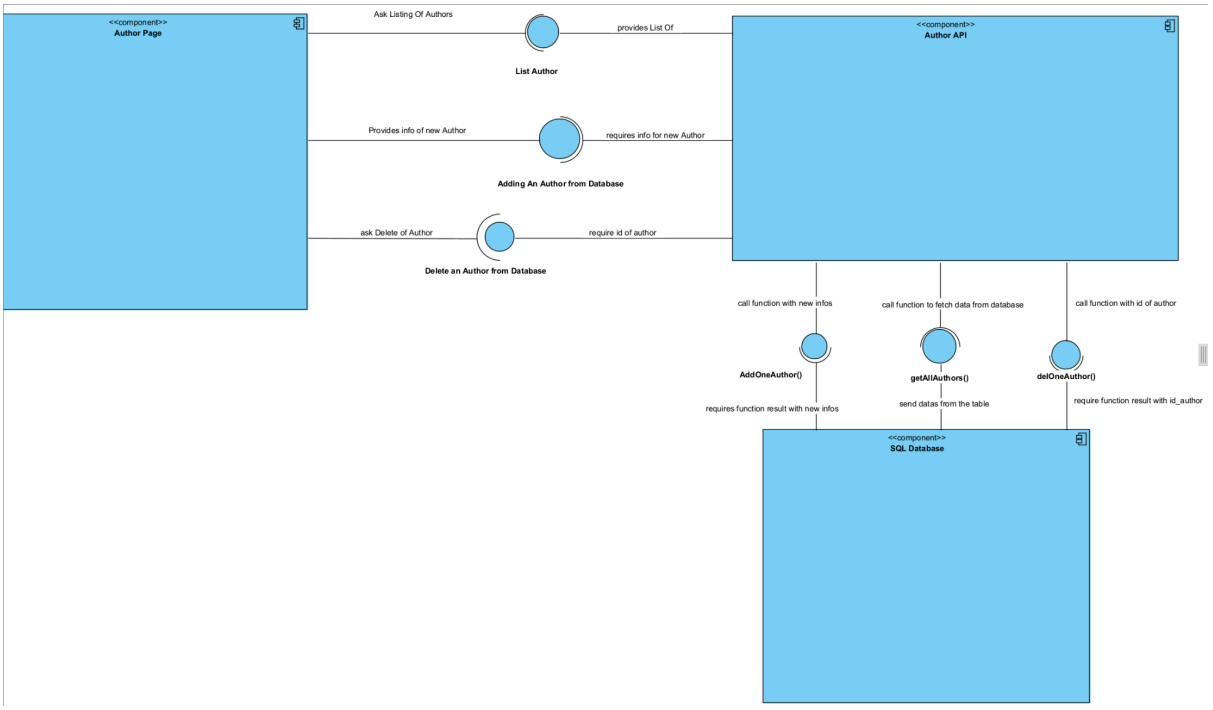
Component Diagram Authentication:

This component diagram illustrates the authentication process within the system. The Authentication Component contains the login, register pages and management of users. These frontend components interact with the Authentication API, that contains user repository, user.auth.js and finally auth.route.js. When the user wants to login or to register it sends a post request that is being sent to the auth.route.js, it will then go to the repository to fetch the user info from the database (in case of login) or to give these infos to the database in case of registration. Then it will go to the user.auth.js to authorize the request and keep a session with the user logged in. In case of registration it will no go to user.auth.js



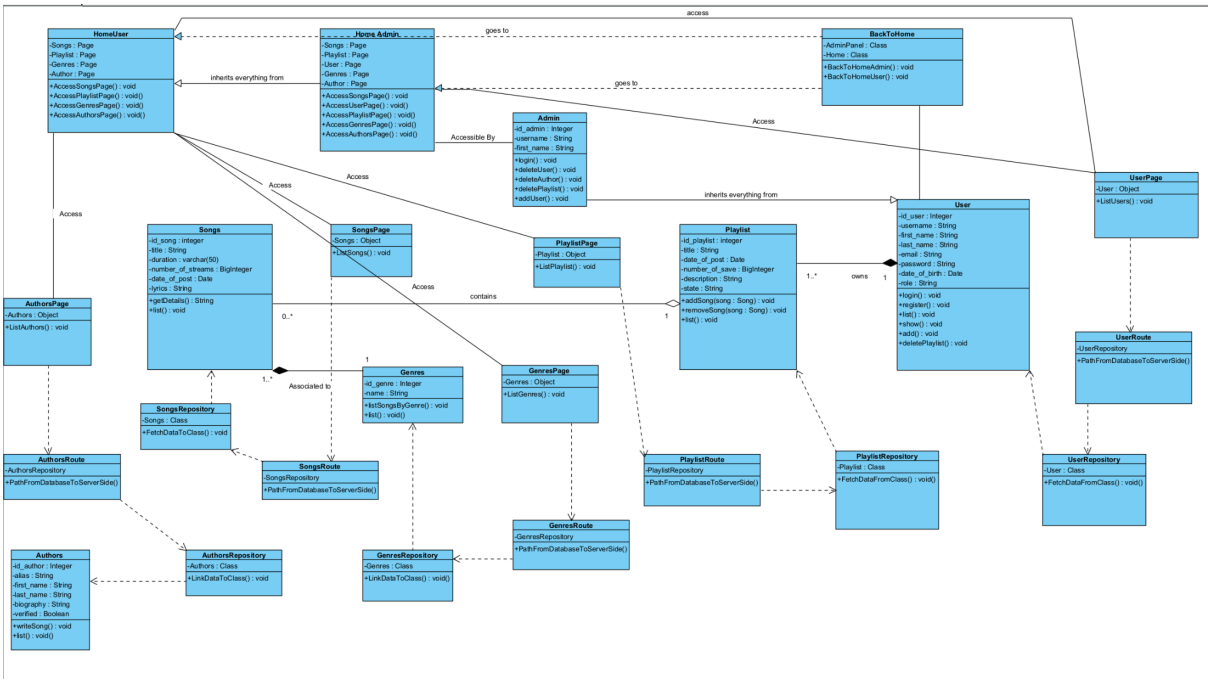
Component Diagram Author :

This component diagram illustrates how the system manages authors. The user interface, represented by the Author Page, allows users to view the list of authors, add a new author, or delete an author by providing/fetching the necessary information. These actions are sent to the Author API, which acts as an intermediary with the SQL database. When asking for the list of authors, the user send a get request that will ask the API to provides the informations by retrieving the infos from the database, if it's a post request like adding an author, the API will then provides the infos that were used in the post request to send them to the sql database that will retrieve and add a new author based on the infos.



Class Diagram :

This class diagram represents the architecture of the playlist management system. Users own playlists, which contain songs. Songs are associated with authors and genres. The administrator can manage entities through the AdminPanel interface. Repositories and routes connect entities to the database, ensuring persistence and linking the backend to the frontend. Pages like SongsPage and AuthorsPage provide user interfaces to access and manipulate entities. It is showing all of the pages, repositories, classes and entities that makes the website working from server side to client side.


access
HomeUser
-Songs : Page
-Playlist : Page
-Genres : Page
-Author : Page
+AccessSongsPage() : void
+AccessPlaylistPage() : void()
+AccessGenresPage() : void()
+AccessAuthorsPage() : void()
goes to
inherits everything from
Access
Accessible By
BackToHome
-AdminPanel : Class
-Home : Class
+BackToHomeAdmin() : void
+BackToHomeUser() : void
Admin
-id_admin : Integer
-username : String
-first_name : String
+login() : void
+deleteUser() : void
+deleteAuthor() : void
+deletePlaylist() : void
+addUser() : void
UserPage
-User : Object
+ListUsers() : void
User
-id_user : Integer
-username : String
-first_name : String
-last_name : String
-email : String
-password : String
-date_of_birth : Date
-role : String
+login() : void
+register() : void
+list() : void
+show() : void
+add() : void
+deletePlaylist() : void
1..*
owns
1
contains
AuthorsPage
-Authors : Object
+ListAuthors() : void
Songs
-id_song : integer
-title : String
-duration : varchar(50)
-number_of_streams : BigInteger
-date_of_post : Date
-lyrics : String
+getDetails() : String
+list() : void
0..*
1..*
SongsPage
-Songs : Object
+ListSongs() : void
PlaylistPage
-Playlist : Object
+ListPlaylist() : void
Playlist
-id_playlist : integer
-title : String
-date_of_post : Date
-number_of_save : BigInteger
-description : String
-state : String
+addSong(song : Song) : void
+removeSong(song : Song) : void
+list() : void
Genres
-id_genre : Integer
-name : String
+listSongsByGenre() : void
+list() : void()
Associated to
GenresPage
-Genres : Object
+ListGenres() : void
UserRoute
-UserRepository
+PathFromDatabaseToServerSide()
SongsRepository
-Songs : Class
+FetchDataToClass() : void
AuthorsRoute
-AuthorsRepository
+PathFromDatabaseToServerSide()
SongsRoute
-SongsRepository
+PathFromDatabaseToServerSide()
PlaylistRoute
-PlaylistRepository
+PathFromDatabaseToServerSide()
PlaylistRepository
-Playlist : Class
+FetchDataFromClass() : void()
UserRepository
-User : Class
+FetchDataFromClass() : void()
Authors
-id_author : Integer
-alias : String
-first_name : String
-last_name : String
-biography : String
-verified : Boolean
+writeSong() : void
+list() : void()
AuthorsRepository
-Authors : Class
+LinkDataToClass() : void
GenresRepository
-Genres : Class
+LinkDataToClass() : void()
GenresRoute
-GenresRepository
+PathFromDatabaseToServerSide()