CSCI-3321

Introduction to Software Engineering

Project Title: CaribBeats

Group Name: Caribbean Chaos Collective

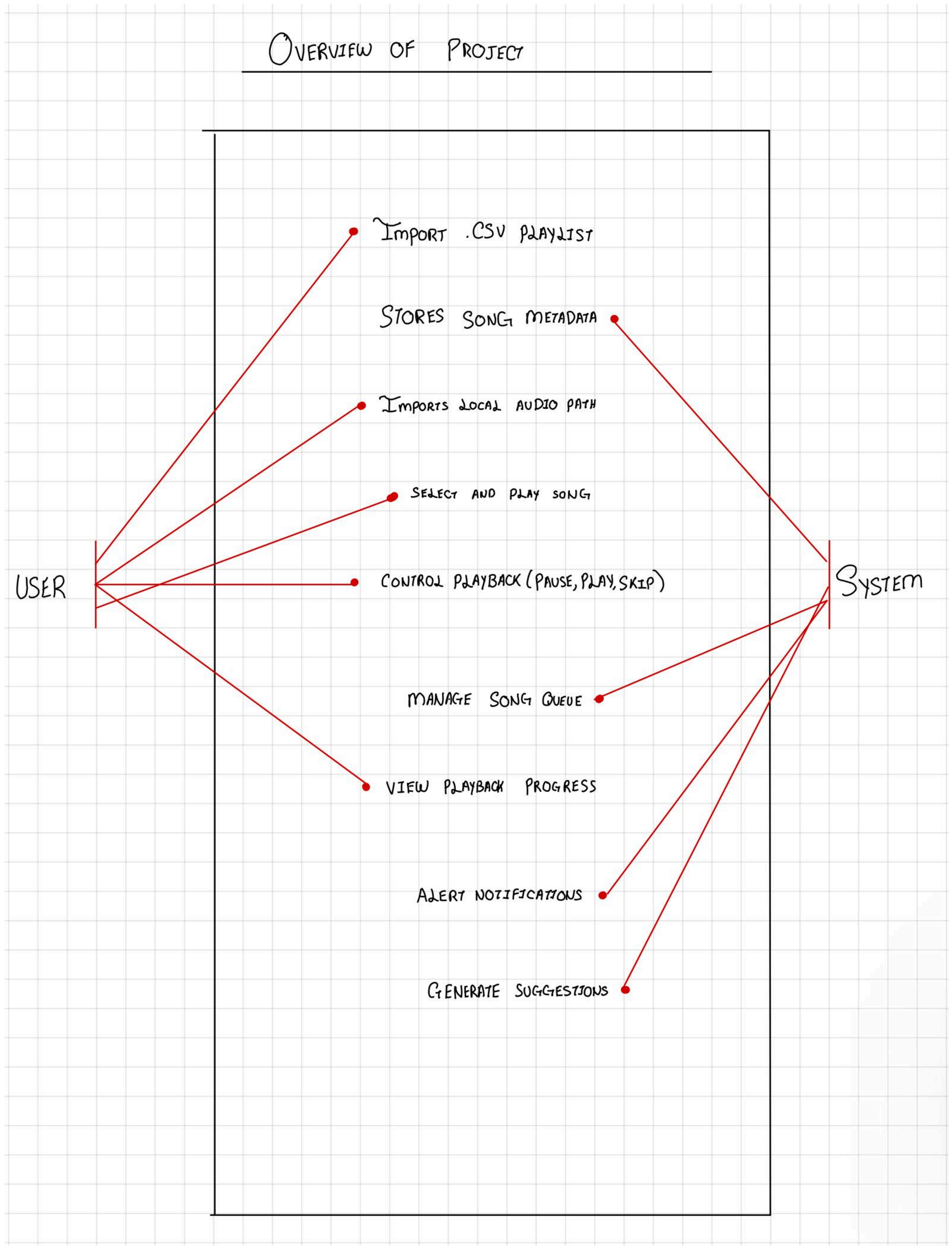
Team Lead: Abigail Riley; ar21583@georgiasouthern.edu

Team Member 1: Michaella Zias; mz02156@georgiasouthern.edu

Project Summary

The program is designed to create a unique and engaging application that organizes and plays songs specifically within the soca and dancehall genres, using the tempo (beats per minute) of each track as the primary organizing principle. These genres are known for their energetic rhythms, and many songs within these genres share similar tempos, which creates an opportunity to enhance the listening experience by carefully curating the flow of music. The application will store essential song metadata in an SQL database, with a specific emphasis on the "Tempo" column, which will be key in determining the order of songs. This data will also include file paths to the local storage locations of the audio files, allowing the backend Java program to access and play the files directly from the user's computer. The backend will use conditional statements to analyze the tempo of each song and select the next track based on a close match in tempo, ensuring that the songs transition smoothly without jarring changes in speed. For example, if a user selects a song with a certain tempo, the system will automatically queue other songs that share a similar tempo, maintaining a cohesive and consistent vibe throughout the session. Additionally, the program will allow users to import playlists from popular music services like Spotify by exporting them to CSV format and then importing this data into the SQL database. This process makes it easy for users to bring their existing playlists into the application, streamlining the integration of their personal music libraries. By organizing and sorting songs based on tempo, the program offers a dynamic, genre-specific playlist experience that not only reflects the energy of the soca and dancehall genres but also provides a seamless listening experience. As the tempo of each song influences the sequence of tracks played, users will enjoy a personalized experience that adapts to their preferences, providing a rich and smooth musical journey from one song to the next, all while ensuring that the tempo transitions naturally without abrupt shifts that could disrupt the flow.

USE CASE DIAGRAMS



Use Case: Play Songs Based on Selected Song's Tempo

Trigger: The user selects a song from the playlist.

Preconditions:

- P-1: The user is logged into the system.
- P-2: The song audio path and its tempo information are stored in the SQL database.

Basic Path:

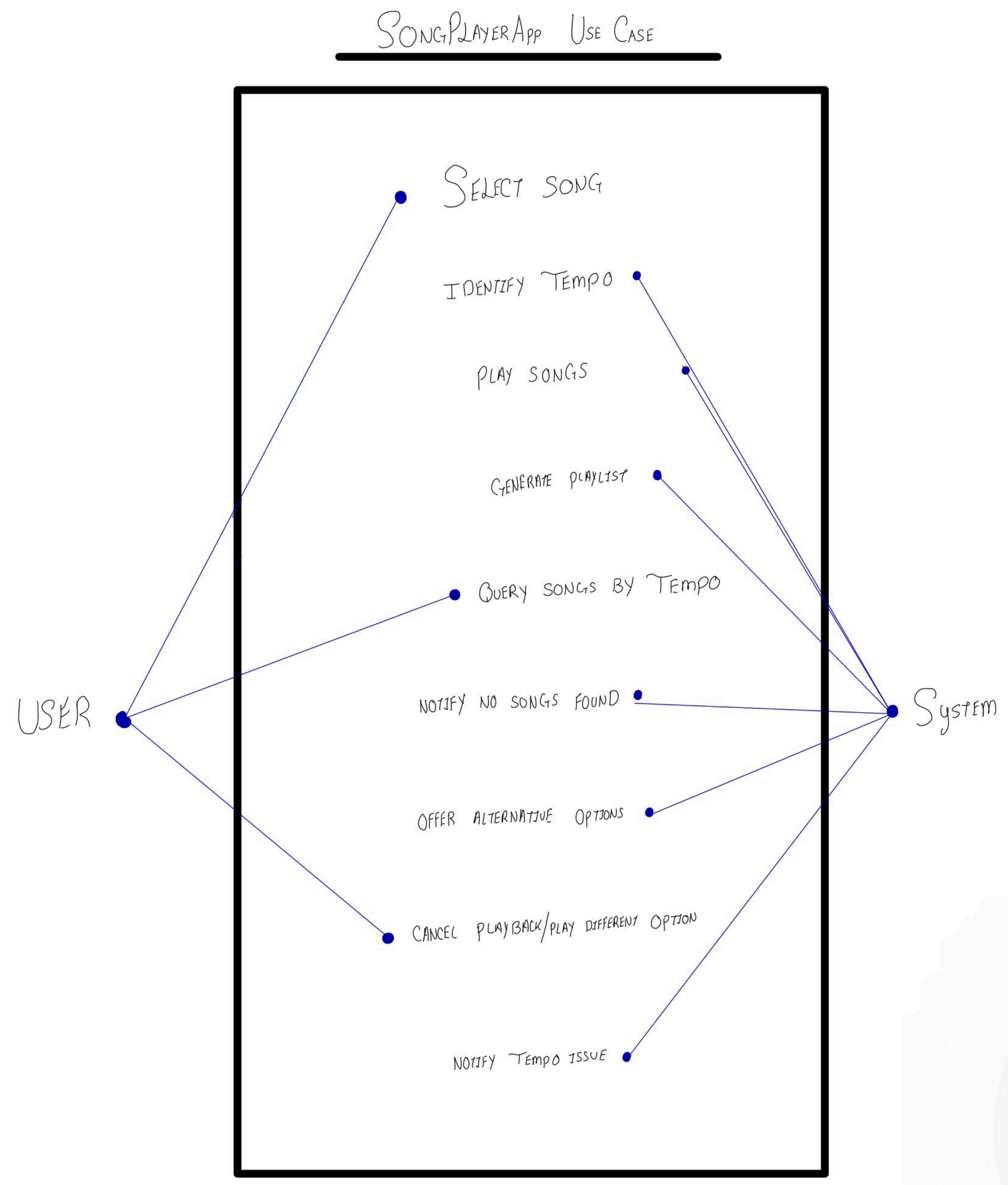
- BP-1: The system shall provide the user with a method to select a song from the playlist.
- BP-2: The system shall identify the tempo of the selected song.
- BP-3: The system shall query the SQL database for all songs that share similar tempos of the selected song.
- BP-4: The system shall continually parse the database for songs with a similar tempo, that has not yet been played.
- BP-5: The system shall play the songs in the generated playlist array in the order of their appearance.

Alternative Path 1:

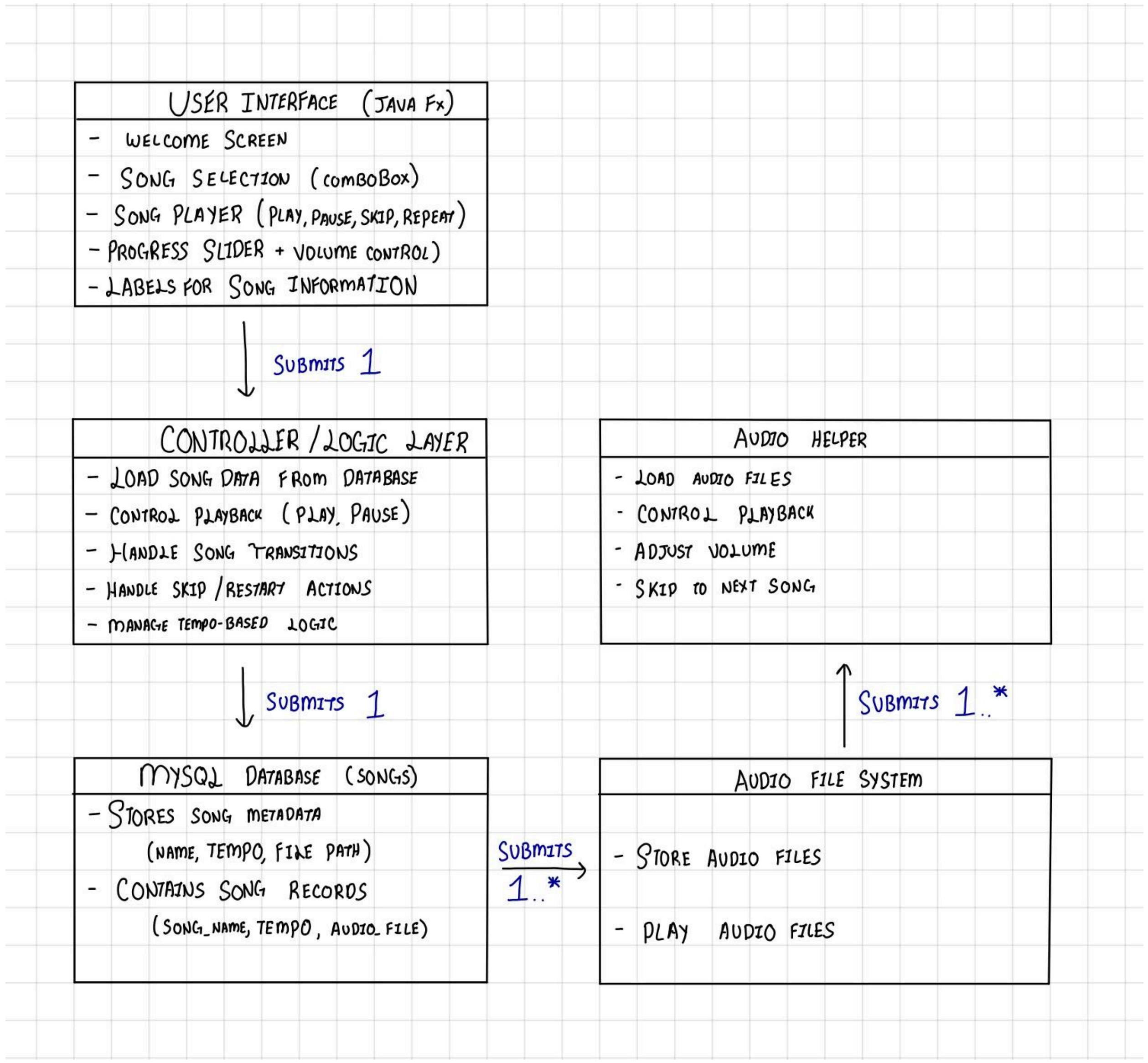
- After BP-3, if no additional songs are found with similar tempos
    - AP1-1: The system shall notify the user that there are no other songs with similar tempos
    - AP1-2: The system shall offer options for the user to select a different song or playlist.

Exception Paths:

- EP-1: The system shall provide the user with the ability to cancel playback at any time before BP-5.

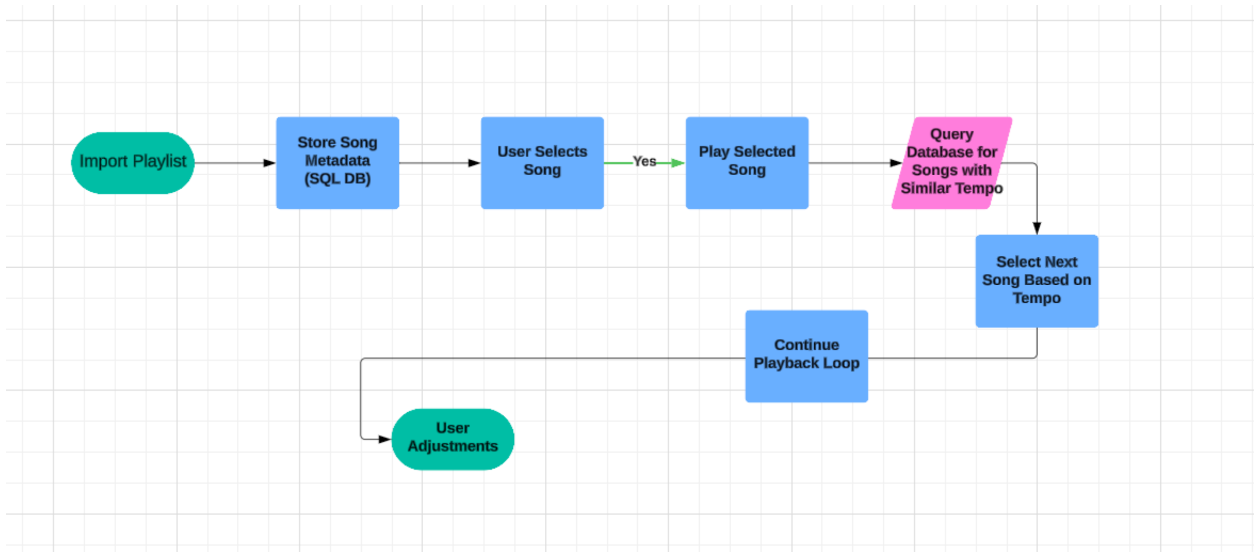

SongPlayerApp Use Case
Select song
Identify Tempo
Play songs
Generate playlist
Query songs by Tempo
Notify no songs found
Offer alternative options
Cancel playback/play different option
Notify Tempo issue
USER
System

CLASS DIAGRAMS


USER INTERFACE (JAVA FX)
- WELCOME SCREEN
- SONG SELECTION (COMBOBOX)
- SONG PLAYER (PLAY, PAUSE, SKIP, REPEAT)
- PROGRESS SLIDER + VOLUME CONTROL)
- LABELS FOR SONG INFORMATION
SUBMITS 1
CONTROLLER / LOGIC LAYER
- LOAD SONG DATA FROM DATABASE
- CONTROL PLAYBACK (PLAY, PAUSE)
- HANDLE SONG TRANSITIONS
- HANDLE SKIP / RESTART ACTIONS
- MANAGE TEMPO-BASED LOGIC
SUBMITS 1
MYSQL DATABASE (SONGS)
- STORES SONG METADATA (NAME, TEMPO, FILE PATH)
- CONTAINS SONG RECORDS (SONG_NAME, TEMPO, AUDIO_FILE)
SUBMITS 1..*
AUDIO FILE SYSTEM
- STORE AUDIO FILES
- PLAY AUDIO FILES
SUBMITS 1..*
AUDIO HELPER
- LOAD AUDIO FILES
- CONTROL PLAYBACK
- ADJUST VOLUME
- SKIP TO NEXT SONG

Represents the structure of a system by showing its classes, their attributes, methods, and the relationships between them.

WORK FLOW DIAGRAM


Import Playlist
Store Song Metadata (SQL DB)
User Selects Song
Yes
Play Selected Song
Query Database for Songs with Similar Tempo
Select Next Song Based on Tempo
Continue Playback Loop
User Adjustments

1. User Imports Playlist

The user imports a playlist from an external service like Spotify. The user exports the playlist to a CSV format. If the user already has local audio files, they can manually ensure that the file paths in the CSV match the correct locations on their computer.

2. SQL Database Stores Song Metadata

The SQL database stores song data, including:

- **Song Name**
- **Artist**
- **Tempo (BPM)**
- **File Path** (points to the local location of the audio file on the user's computer)

3. User Selects a Song

The program retrieves the song's metadata (including tempo and file path) from the database.

4. Java Backend Queries the Database

The program queries the database for other songs with a similar tempo (within a set range of ±0.600 BPM).

5. Playback of the Selected Song

The song is played through the application, and the tempo is tracked for the next song selection.

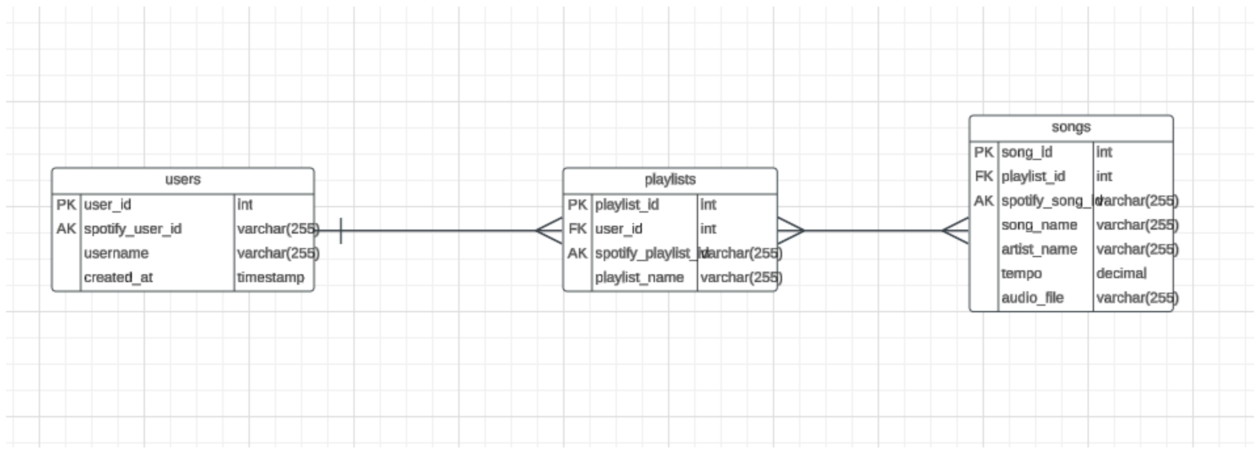
6. Next Song Selection Based on Tempo

The backend queries the database for the next song with a similar tempo (or the next closest match). The file path of the next song is retrieved, and the audio is played.

7. Optional Adjustments

- Users can adjust their preferences, such as:
    - **Skipping songs**

DATABASE ER DIAGRAM



A song can appear in multiple playlists, and a playlist can contain multiple songs. This is a many-to-many relationship.

Each playlist can be associated with a single user (i.e., the user who created it), but a user can have multiple playlists. The Playlist table has a many-to-one relationship with the User table.

TECHNOLOGY USED

Java: The core programming language used for the backend logic and overall application structure.

JavaFX: Utilized for building the graphical user interface (GUI) of the application, providing an interactive and visually appealing experience for managing playlists and controlling playback.

javax.sound.sampled: Used for audio playback functionality, allowing the application to load and play audio files directly from local storage. This API is employed for lower-level audio processing, providing control over playback, volume, and audio format handling.

MySQL: Chosen for its reliability and performance in storing and querying song metadata, such as song title, artist, tempo, and file paths.

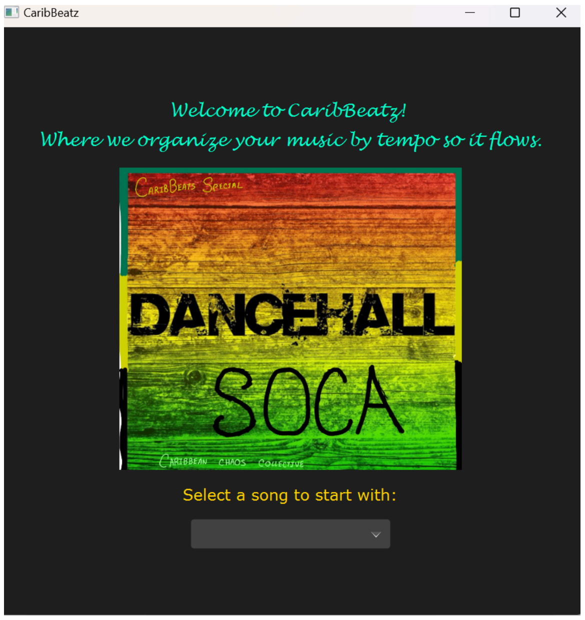
JDBC: Implemented to establish a connection between the Java application and the MySQL database, allowing for seamless data retrieval and management of song information.

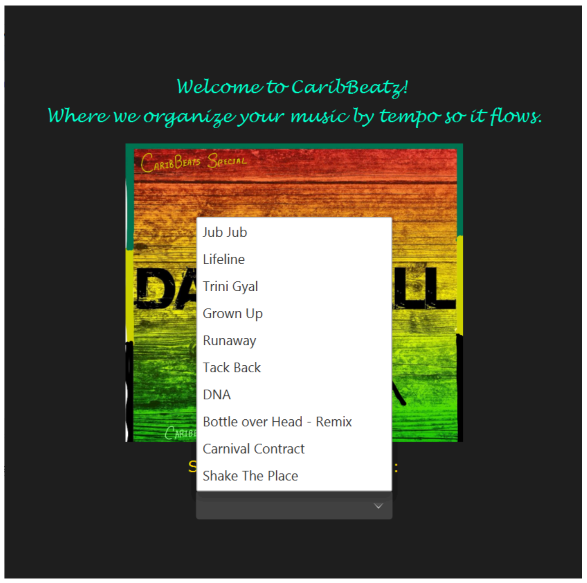
CSV Parsing Libraries: Used to import playlists from Spotify (or other music services) by parsing CSV files and converting them into a format suitable for insertion into the SQL database.

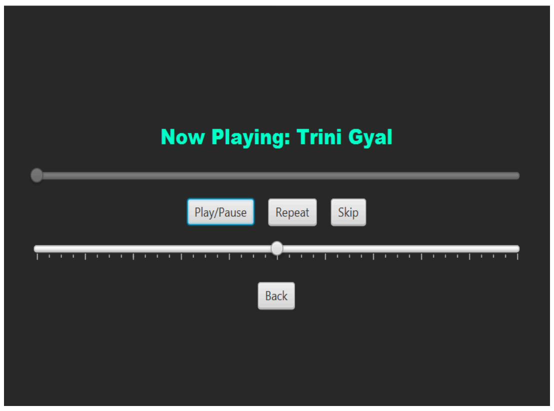
GitHub: Utilized for version control, allowing for team collaboration and managing codebase revisions throughout the development process.

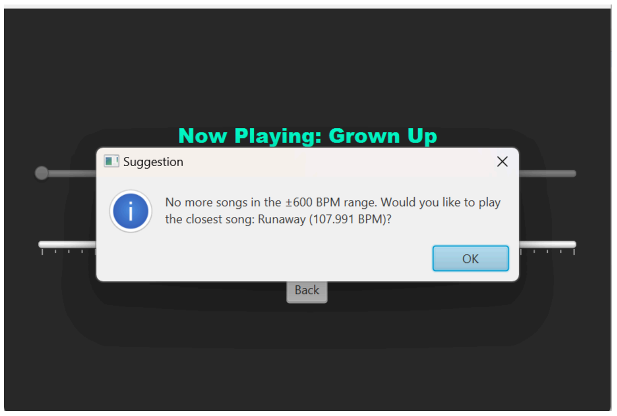
Exportify: Used to export Spotify playlists into CSV format, enabling easy integration of external playlists into the application

PROJECT SCREENSHOTS


CaribBeatz
Welcome to CaribBeatz!
Where we organize your music by tempo so it flows.
CaribBeats Special
DANCEHALL
SOCA
Caribbean Chaos Collective
Select a song to start with:







No more songs in the ±600 BPM range. Would you like to play the closest song: Runaway (107.991 BPM)?

TEAM MEMBERS CONTRIBUTIONS

The team was lead by Abigail Riley and the tasks were distributed as below:

Abigail - Design and create the SQL database schema to store playlist data, create the necessay SQL tables and relationships, develop the interface that allows users to upload CSV files from their local system, and store audio file path, Integrate a media player within the GUI to handle song playback, create SQL queries to call songs based on conditional statements.

Michaella - Design GUI layout, design and develop the user interface for the application, create screens for playlist management (importing, viewing, and editing playlists), as well as playback controls (play, pause, skip), implement conditional statements in Java that determine the next song to play based on the current song's tempo.

Integration Testing & Documentation / Code Review - both