Music Server with Group Listening

Ben Basten
Advisor: Andy Poe

Before COVID and when large gatherings were still allowed, attending concerts was one of my favorite pastimes. There is something magical about a large group all gathering in the same place with the common goal of having fun and enjoying the artistic craft of whoever is on stage. Now that social distancing and quarantining is in full effect, I would like a way to be able to connect with my friends and family by simultaneously streaming music from afar.

The purpose of this music server is to enable users to listen to music in real-time with other users of the server, so it will be designed with listening “rooms” that clients can join. Each room will have an independent queue of music playing. The rooms will be a collaborative format, so everyone has equal rights to play, pause, skip, and add to play queue.

There will be an API on the backend that will administer the creation of these rooms and streams to my personal local collection of music. I will most likely be using the Spring Boot framework, with Java WebSockets or MulticastSockets handling the streaming and room creation. The API will be able recursively index the music library. Through the various public endpoints, clients will be able to see a listing of the music collection and use these listings to add to the play queue within their room. To make this API more robust, I will be implementing Spock unit and integration tests along the way.

For the front-end, I will be creating a simple React.js interface. I would like to spend most of my development time building the API, so the styling will be simple yet consistent. The interface will provide users the option to create or join a room on the landing page. Upon successfully creating or joining a room, there will be asynchronous calls within the room to perform actions such as playing, pausing, skipping to the next song, or adding to queue.

Rubric

80pts: Backend

- 5: Unit Testing on controllers using Spock
- 2: File organization consistent with Spring Boot standards
- 10: Backend can index .mp3 and .ogg files that are available in a root directory
- 3: Backend can recursively index .mp3 and .ogg files that are nested (ex. [artist_name] > [album] > song.mp3)
- 8: Endpoint to retrieve listing of available song titles
- 5: Endpoint to provide current song information for a room
- 10: Users can create a room and generate a room code
- 5: Users can join a room by room code
- 15: A room can play a single predetermined song to multiple clients (multicasting)
- 5: A room can play a predetermined playlist of music
- 10: Clients within the room can play and pause
- 5: Clients within the room can skip to the next song
- 5: Clients within the room can add to queue for that specific room
- 2: Count of number of clients within the room

80pts: Frontend

- 5: Create/join room landing page
- 15: Successfully can create or join a room with API calls
- 5: Consistent look/feel
- 10: User can play/pause
- 5: User can skip
- 3: User can see a listing of available music to add to queue
- 2: User can add to queue
- 10: Audio plays on the client-side
- 15: Audio stream is the synchronized for multiple clients
- 5: Display current track information
- 5: Queue visible in UI

Total: 160pts

Grading Scale:

A	130-160
B	110-129
C	95-109
D	80-94
F	< 90