MusiQ Design Document

The interactive music player for friends to collaborate on what they're all listening to.

Overview

The MusiQ web application is a hub for people to join and collaborate on playing songs. A user will join or log in on their own device with their Spotify Premium account to create or join a joint listening group. Once connected to a group, they can contribute to what song might play next and when. Thus, multiple users from anywhere in the world can listen to the same playlist simultaneously.

A user can search for songs from Spotify's library and add them to their queue. If they don't like what order the song was added in, they're able to rearrange the order of the songs at any time. When a song is added, removed, or rearranged, all users currently listening will receive a live update of the change too.

Users can pause, play, or skip the songs in the queue at any time and the commands will live propagate to every user connected. Thus, if you pause a song on MusiQ, everyone else in your group will have the song paused too. MusiQ's queue creation and editing features create a universal music environment for friends, family members, or even strangers from anywhere to experience together. Just log in with Spotify and create or join a listening group to get started.

User Stories

As a person without an account or who is not logged in, I can look at the index page for MusiQ to see what it is. Since the features of MusiQ are all specific to users who have Spotify Premium accounts, I cannot see or interact with queues without logging in or creating an account first. However, from the main page, I will be able to learn what MusiQ is and what I can do if I was a user. I will also have the option to log in or create an account from the main page so I know how to get started.

As a new user, I will be able to log in to MusiQ with my Spotify Premium account credentials. If I do not have a Spotify account, I will be able to create one to log in to MusiQ with from the main application. Once I'm logged in, I can join a pre-existing listening group by entering the name and creator of it or I can create one myself. Once I'm in a joint listening group, I can search for songs to add to the playlist, rearrange the songs in the playlist, pause or play the current song, and remove songs. I will only be able to listen to the song playing if I have my Spotify app or web player open since songs can only be streamed through Spotify's own

application.

As a user, I will be able to see what groups I've been a part of before. When I view my previously joined groups, I can re-join them to access all features of the queue just as I had previously. If I did not create a group and it was deleted by it's creator, then I will not be able to see it in my list at all.

As a user who created a listening group, I will have the ability to edit the information about the group I created. I will be the only one who can rename the group's name and description or delete the group entirely. If I delete the group, all users who had joined it before will no longer be a part of it. I will be able to see it in my previously joined groups list, however, users who were in it will not.

As an admin, I will be able to have full control of every group and user created. I will be able to delete users from MusiQ, delete groups created by anyone, and edit the information of groups. If I delete a user's account who has created groups before, the user's groups will also be permanently deleted.

As a user, I will be able to search for other users too see the groups they have created or been a part of. I will be able to search for their Spotify/MusiQ username and if they have an account, I will be brought to the page that shows what groups they've previously joined or created. I can then join any of the groups listed since if a group has been deleted, I will not be able to see it at all.

Data Design

User

Name	Data Type	Description	Relationships
Spotify User ID	int	A user's Spotify User ID is the unique account ID key they're given upon creation of their Spotify account.	
		A user's username that	

Username	string	they set when they first sign up with their Spotify account.	
Email Address	string	A user's email address is the email address they have connected to their Spotify account.	
Admin?	boolean	A boolean that indicates if the user is an admin user of MusiQ or not.	
Created Groups	MusiQ Group	The groups that a user has created.	"has_many": A user can have 0 or more MusiQ Groups attached to their user account. If they have created a group, they will be linked to their account here.
Joined Groups	MusiQ Group	The groups that a user has joined before.	"has_many": A user can have 0 or more MusiQ Groups attached to their user account. If they join a group, the group will be connected to their user account here. Groups a user have created will be in Created Groups and not here.

MusiQ Group

Name	Data Type	Description	Relationships
Name	string	The name of the group created.	
Group ID	int	A unique ID created for the group.	

Description	string	An optional description that can be set for the group by the creator only.	
Creator	User	The MusiQ user who created the group. The creator is able to edit the group's name, description, and delete it.	"belongs_to": A MusiQ group only has one Creator that it belongs to.
Listeners	User	The users who are currently joined and listening to the MusiQ group.	"has_many": A group can have many users listening/joined at once.
Songs	Song	The songs that are currently in the queue to play in the group.	"has_many": A group can have 0 or more songs in the queue.

Song

Name	Data Type	Description	Relationships
Spotify Song ID	int	The unique Spotify ID of a specific track.	
Track ID	int	The ID created by MusiQ for a song. Any time a song is added to a group, a track ID is created for it.	
Order	int	The order the song is set to play in in it's group.	
Group	MusiQ Group	The MusiQ group that the song is set to play in.	"belongs_to": The track ID of the MusiQ representation of a song only belongs to 1 group.

App Interface

The public landing page of the application will have a brief yet detailed description of what MusiQ is and what a user can do with it so users who have not visited before may feel compelled to create an account. Users who *have* visited before but whose account information is no longer cached can log in directly from this landing page.

The index page of a logged in user instructs them to either create a group, join a pre-existing group, or view their previously joined groups. If they choose to create a group, they will be prompted to enter a group name and optional description. If they choose to join an already existing group, they enter the name and username of the creator of the group and press join. If they view their previously joined groups, they will see a list of groups that they have created before and groups that they have been a part of before; all will have an option to rejoin. Once a user is in a group, they will see the title, description, and creator of the group at the top of the page. There will be a queue of songs that are scheduled to play next and a player bar which shows the song that is currently playing or paused. A user can drag and drop songs to rearrange their order in the queue. They can also search for songs at the top to add to the queue. As for the play bar, they can pause/play the current song or skip to the next one in the queue from the buttons there.

Each page in the app will have a menu bar at the top that will allow users to switch quickly between their group, their previously joined groups page, and the index page. The menu bar will also have a logout button if logged in or a button to log in if on the public landing page. Overall, the users can log in, join a group, and start playing in one easy user flow.

Experiments

Experiment 1 - OAuth and Spotify Search API

- Source Code: /spotifyExperiment/spot/

The first experiment was a test of using the Elixir wrapper spotify_ex for the Spotify API and implementing OAuth. The library was written by Jason Cummings and can be found at https://github.com/jsncmgs1/spotify_ex. A basic Phoenix web application was created and which demonstrates the ability to log in to an existing Spotify account using OAuth, and to then utilize the Spotify search feature. Through this experiment we learned what the specifications of the Spotify API, most importantly that our users most have access to a Premium Spotify account, as many features of the API are not available to free accounts. This discovery made it clear that our app would have to be marketed only towards those with Premium accounts. We also learned the limitations of the spotify_ex library. While it does an excellent job and implementing OAuth and making API calls, it only provides us with a small

subset of the Spotify API functionality. This discovery led us to conduct our second experiment, which involved extending the spotify_ex library in order to utilize the functionality we need in the Spotify API.

Experiment 2 - Pause and Play

- Source Code: /spotifyExperiment/extra/
- Forked Wrapper: /spotifyExperiment/spotify_ex/

As mentioned in the previous experiment, the spotify_ex wrapper library does not provide us with everything we need out of the Spotify API. Most importantly, it does not allow us to control playback of music, which is clearly and integral part of MusiQ. To that end, this experiment consisted of an extension of the spotify_ex library. We forked the original wrapper from github and added access to missing endpoints and functions as needed, matching the style and structure of the original library as much as possible. For this experiment we tested the play and pause features of the API. Through this process we realized that Spotify does not allow the playing of their music through external audio players, meaning that all playback has to occur in an official Spotify application. For our application to work as intended, the user must have a separate Spotify application open. The play and pause commands tested in this application control the playback of music on the Spotify application the user has open.

Experiment 3 - React Drag and Drop

- Source Code: /reactExperiment/

For the front end of MusiQ, we decided to see if React would be useful in reordering the music queue. We want users to be able to be able to rearrange the order of the songs set to play next by dragging and dropping them. We set up a bare elixir app with a react front end and found the react-dnd library to implement drag and drop. We quickly learned that our version of babel requires installing extra plugins, like backwards compatibility support for decorators (`transform-decorators-legacy`). Once all the necessary plugins were installed to get react working in the browser with a sample project implementing drag and drop, we were able to create a miniature prototype of what the "Up Next" feed will look like. Since the state, and thus the ordering, of the songs is maintained entirely in the react front end and the server will need to know what songs to play next to play them, we now know that we'll be implementing web sockets to handle some of the updated state logic queues in each music group.

Project Status

With the OAuth, search, and pause and play implemented, a lot of the backend functions are completed. However, we still have to get our database set up, add users, support re-ordering, and implementing web sockets for the live updates. We've run into problems in noticing that

the Elixir wrapper for the Spotify API does not contain every functionality so we have to write a few of our own. We also noticed that Spotify requires the web player to be open to play songs and for a user to have Spotify Premium to connect with it. Thus, we've changed our app to have to require these two things of users. One other major change will be implementing the react front end to communicate with the server for telling Spotify which song to play next. The next steps for our application will be connecting the front end with the server via web sockets, adding our own database objects described above, and completing the frontend for all our app interfaces.