Feature Request: Playlist Features

**April 3rd 2023**

# **OBJECTIVE**

(A brief description of the overall goal of this feature request, summed up in one to two sentences.)

To give users the ability to rearrange their Spotify playlist tracks after they have been added from the search, and before the playlist is saved.

# **BACKGROUND**

A more thorough explanation of why this feature is important to implement and all of the essential functionality of the feature. This section could contain research findings, user stories, examples of similar features, or anything else that explains why this feature has been prioritized and why the technical design laid out below implements the pieces of functionality that it does.

An integral feature for building customized playlists is the ability to tweak the playlist by moving individual playlist tracks until the track order is just right.

Currently, the Jammming App functionality is limited; once a user adds tracks from the search results to the playlist the track order is fixed. By adding a modification function that shifts individual playlist tracks up or down, we give the user the ability to rearrange individual playlist tracks before saving and fine-tune their music selection.

# **TECHNICAL DESIGN**

This section should lay out all of the information needed to implement this feature. In the context of React, this could include new components and their functionality, existing component updates, how to address edge cases, and any other information an engineer would need before implementing this feature.

For front-end changes, this section will often include design mocks or wireframes to specify how the design of the application needs to be updated.

**Moving Tracks saved in App.js state**

In App.js, we will add two methods for moving playlist tracks, one to move the track up and one for moving the track down. These methods should update the App state to display the new playlist order. These two new App.js methods: moveTrackUp(track) and moveTrackDown(track) will accept an argument of track that will be used to create a copy of the playlistTracks array, remove the track, then reinsert the track at the specified location.

To save the original track position for reinsertion, moveTrack\*(track) methods use Array.prototype.findIndex() on playlistTracks state to find the index of the supplied track and save it to a variable named origTrackPosition that is to be used when reinserting the track.

Next, the callback of Array.prototype.filter() will use the id property of the current track to create a copy of the playlistTracks array named newPlaylistTracks that contains all of the original tracks minus the track we want to move.

Using Array.prototype.splice() with a start parameter of an offset origTrackPosition variable; the track, depending on move direction, is either spliced into the newPlaylistTracks array one unit less than its original position, or one unit greater than its original position. However, before insertion we ensure the new track position is not out of the arrays bounds.

At this point moveTrack\*() methods should be bound to the current instance of the App component.

With mutation of the new array now complete, moveTrack\*() will use setState() to overwrite the original playlistTracks array with the updated newPlaylistTracks array resulting in an updated playlistTracks state that can be used to display the moved tracks new location.

**Displaying Track Move Buttons on the Track component**

To display up/down buttons on each individual playlist track we will use the Track components renderAction() method. renderAction() is currently responsible for rendering the plus/minus buttons that add or remove playlist tracks and is therefore a good place for our move button logic.

Not all playlist tracks require both up/down buttons. Tracks at the beginning of the playlist don’t need up buttons and tracks at the playlists end need no down button, having Tracklist pass props firstTrack={true} or lastTrack={true} to the first and last tracks will provide values renderAction() can use to avoid rendering unneeded buttons.

**Adding Functionality to The Track Move Buttons**

To add the needed functionality to the Track Move buttons the App.js moveTrack\*() methods have to be passed as props to Playlist, which will pass them to Tracklist, which will pass them down to Track the same way the App.js removeTrack() method is passed to the Track component as an onRemove prop.

Once the moveTrack\*() methods are available to the Track component as props, event handlers: trackUp() and trackDown() can be created that call moveTrackUp() and moveTrackDown() on track respectively.

The event handlers will then need to be bound to the current instance of the Track component and added to the renderAction() up/down buttons as onClick attributes.

# **CAVEATS**

This section is used to lay out alternative solutions and their respective drawbacks, as well as potential drawbacks to the proposed solution above. This is used to make it clear why the technical implementation detailed previously was chosen instead of alternatives. It additionally allows stakeholders or other developers to consider those drawbacks and choose one of the alternate solutions if they prefer it. This may occur if they feel the benefits or drawbacks of that solution are more desirable than the current solution, or if they can identify other benefits and drawbacks not currently listed.

**Two Move Track Methods**

The moveTrackUp() and moveTrackDown() methods are very similar, and could be combined into one method. For clarity, and to avoid having tracks pass a direction parameter, two separate methods are used.