|  |  |  |  |
| --- | --- | --- | --- |
| Version Control | | | |
| Version | Author | Comments | Date |
| V1 | Kevin Ibelgauptas |  | 10/01/2021 |
| V2 | Kevin Ibelgauptas | Fixed grammar and updated | 28/03/2021 |
| V3 | Kevin Ibelgauptas | Third version of the document, had to change formatting. | 06/04/2021 |

# Documentation

This document will discuss what components were added within iteration 1. Below you will see information regarding currently playing songs, artist names, tracks, albums, play buttons including the slider bar and lastly queuing. This document was made to provide some knowledge to other members in the group of what was done during iteration 1, so they have something to look back on when confused on how the front end operates.

# Client/src/Components

## Hoc (higher-order-component)

The term Hoc refers to a function which accepts another function as an argument (Map function).

### PlayerHoc

The “playerHoc” is a class which focuses on displaying the buttons which skip to a new song and previous song, pause a song, play a song and manipulate the song’s seeking position. This class uses props which contain the current song, its duration length, title, and artist along with whether the song is playing currently. Once a playback option is pressed, the data is sent using a WebSocket client to the room WebSocket server. The room updates its queue and playback data if required before the WebSocket server sends the data required for the appropriate Spotify API request to be called to every user in the room. Each user receives the data with their WebSocket client, and the data is then used to send the Spotify API request, thereby allowing each user to have synced playback with the host.

### StatusHoc

The “statusHoc” is similar to the “playerHoc” however it focuses on the song being able to play through the play/pause button. It uses the song’s URI, and whether it’s playing or not to retrieve information about the track and if it’s currently playing.

### UiHoc

The “uiHoc” is used to compose the props together.

## LobbyUsers

### LobbyUsers.jsx

This file contains the code regarding the lobby users and their connections. The file contains code relating to the WebSocket retrieving which users joined the room and displaying them. Code related to displaying users in the room will be found here along with the design of how they’re shown (colour, background and even viewing a user’s Spotify profile by clicking their username).

## lobbyTopTracks

### queueSection.jsx

This file contains the details of having songs pushed to the queue section in the left section of the application. This file focuses on having the WebSocket grab the songs from the middle of the page (room queue) and queue them in our own room’s queue list so it could connect with Spotify’s API and interact with different songs playing.

## songsTable/hoc

### songHoc.jsx

This file is what controls the tracks resetting back to 0:00 when “previous” control is pressed and skipping to the next song in the queue. This allows the room to render the next song and have all its details be displayed.

## Footer

The “playBox” contains a song player which is displayed to the user, providing the user with the buttons in which they can manipulate songs.

## songsPlayer/components

### songsPlayer.jsx

This file is what builds together the bottom section, where the song’s name, artist, album and album cover is shown along with the seeking bar which includes the ability to change the seeking position (where the song currently is in its duration).

### controlButtons.jsx

This designs the props (buttons) so they look normalised and do not clash with the overall style of the app, whilst also allowing them to be clicked on with the “onClick” JavaScript function.

### detailsSection.jsx

Displays the details with specific limitations. For instance, the song’s name length, or the artist’s name length, keeping it tidy and minimalised so if a song has a long title it does not fill up the whole screen and is instead cut off. This also holds some CSS (cascading style sheet code) for the artist’s name, album title and the song’s name, though this CSS is limited to the font size and colour.

### songSlider.jsx

This is the bar seen whilst the music plays including the seconds counters on either side of the bar. It also contains CSS which displays clicking on the bar, the bar highlighting, the progress of where the bar’s position is, the time and lastly the design of it.

### songsControl.jsx

The button icons for playing and pausing the song, going back to a previous song or going forward to another song. This is where the icons are placed on specific buttons (play, pause, skip, previous).

### volumeControl.jsx

Code on the volume control and how it changes once a position is clicked on in the progress bar. Allows the volume to be changed to any value, even 0, by just clicking. Any code relating to the volume bar will be found here.

### volumeSlider.jsx

Displays the volume bar, focuses on its design, positioning, shape, cursor hover and so on.

## Spotify

### webPlayback.jsx

This focuses on the SDK, playing device, current state and errors which identify if the user which is connected has successfully connected or if there is a problem with the SDK, authorization, or the device.

## TopBar

### Top.jsx

Displays the user’s Spotify username and their display image (as of December 2020, it is running on componentDidUpdate, which constantly refreshes and causes Spotify to cut out authorization due to Spotify limits on how many requests can be sent. A fix is in the works).

## TrackCover

### trackCover.jsx

The album cover art of the current song. If the song is not a part of any album, it’ll just display the song’s cover art.

## TrackTable

### queueButton.jsx

This class focuses on queuing tracks to the user’s queue list. It uses the data from the trackTable which is shown to the host of top tracks of all the users in the room, to which they can queue the track into their queue list.

### trackTable.jsx

Contains the track table which displays the top tracks of all the users in the room.

## UserDetails

### userDetails.jsx

Displays who the host is currently logged in as, displaying username and the display name. (You may see several different versions of these or similar classes which is due to an authentication error currently at hand. This will be fixed soon).

# Client/src/dataHandler

## Store/actions

Actions are just plain JavaScript objects which carry information from the current application to the store. This comes in hand with reducers, in which reducers use the actions to determine an application’s state change. Its sole purpose is to perform a specific “action”. Consider a scenario where, theres two asks, one is to turn off the music and the other is to turn on the music, these two “actions” are performing a specific task.

### Axios.js

Axios base instance setup which defines a URL and headers as configuration elements.

### libraryActions.js

This consists of constants (const) where the variables’ values do not change. We use Axios here to retrieve the data from Spotify. It retrieves the current song.

### spotify.js

This consists of consts using Axios to retrieve data from Spotify, such as nextSong, previousSong, playTracks, setActiveDevice, setToken and fetchUser.

## store/reducers

As mentioned previously, a reducer is a function which determines changes to an applications state.

### libraryReducer.js

Currently only holds a case which uses “containsCurrentSong” to change the data of which song is currently active.

### playerReducer.js

Fetches the current status of whether a song is actively playing.

### rootReducer.js

Combined several reducers as the app is continuously growing and it is getting more complex. This reducer focuses on calling every child reducer and gathering the results into one single state object.

### sessionReducer.js

This reducer is used for updating the current device the user is on, along with verifying it through the token.

### uiReducer.js

Reducer which changes the user interface regarding displaying songs, artists, art, display images and so on.

### userReducer.js

This reducer focuses on displaying the user’s data, such as the username, display picture and ID. (As stated before, currently going through some issues due to authentication but soon to be fixed.) This returns errors if the user data has not been successfully fetched.