

Front end component documentation



December 13, 2020

Kevin Ibelgauptas

Contents

[Documentation 3](#_Toc60428132)

[Client/src/Components 3](#_Toc60428133)

[Hoc (higher-order-component) 3](#_Toc60428134)

[PlayerHoc 3](#_Toc60428135)

[StatusHoc 3](#_Toc60428136)

[UiHoc 3](#_Toc60428137)

[playBox 3](#_Toc60428138)

[playBx/components 3](#_Toc60428139)

[controlButtons.jsx 3](#_Toc60428140)

[Details.jsx 3](#_Toc60428141)

[durationMs.jsx 3](#_Toc60428142)

[songsControl.jsx 3](#_Toc60428143)

[playBx 4](#_Toc60428144)

[playBx.jsx 4](#_Toc60428145)

[Spotify 4](#_Toc60428146)

[webPlayback.jsx 4](#_Toc60428147)

[TopBar 4](#_Toc60428148)

[Top.jsx 4](#_Toc60428149)

[TrackCover 4](#_Toc60428150)

[trackCover.jsx 4](#_Toc60428151)

[TrackTable 4](#_Toc60428152)

[queueButton.jsx 4](#_Toc60428153)

[trackTable.jsx 4](#_Toc60428154)

[UserDetails 4](#_Toc60428155)

[userDetails.jsx 4](#_Toc60428156)

[Client/src/containers 4](#_Toc60428157)

[currPlaying 4](#_Toc60428158)

[currPlaying.jsx 4](#_Toc60428159)

[mainSection 5](#_Toc60428160)

[mainSection.jsx 5](#_Toc60428161)

[Client/src/dataHandler 5](#_Toc60428162)

[Store/actions 5](#_Toc60428163)

[Axios.js 5](#_Toc60428164)

[libraryActions.js 5](#_Toc60428165)

[spotify.js 5](#_Toc60428166)

[store/reducers 5](#_Toc60428167)

[libraryReducer.js 5](#_Toc60428168)

[playerReducer.js 5](#_Toc60428169)

[rootReducer.js 5](#_Toc60428170)

[sessionReducer.js 5](#_Toc60428171)

[uiReducer.js 5](#_Toc60428172)

[userReducer.js 5](#_Toc60428173)

[Room-service/api/clients 6](#_Toc60428174)

[App.js 6](#_Toc60428175)

# Documentation

This document will the code in which provides our App to interact with Spotify. Below you will see information regarding currently playing songs, artists name, song, track and album, the play buttons along with the slider bar and lastly the queueing.

# 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, to a previous song, pause a song, play a song and manipulate the songs duration bar. This class uses props which contain the current song, its duration length, title, and artist along with whether the song is playing currently.

### 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 songs 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.

## playBox

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

## playBx/components

### controlButtons.jsx

This designs the props(buttons) to which they look normalised and do not stand out to the style of the app, whilst also providing them to be clicked on with the “onClick” javascript function.

### Details.jsx

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

### durationMs.jsx

This is the bar seen whilst the music plays, and the second’s count. It also contains css in which displays clicking on the bar, the bar highlighting, the progress of where the bars position is, the time and lastly the design of it.

### songsControl.jsx

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

## playBx

### playBx.jsx

This is the class, which is responsible for displaying the artists name, songs name, album artists and lastly the durationMs (which is the length of the song which is going by in seconds). The css file can be found in a separate file called playBx.css.

## 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, and the device.

## TopBar

### Top.jsx

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

## TrackCover

### trackCover.jsx

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

## TrackTable

### queueButton.jsx

This class focuses on queueing tracks to the user’s queue list. It uses the data from the trackTable which is shown to the host of their topmost listened songs, to which they can queue the track into their queue list.

### trackTable.jsx

Contains the track table in which displays the user’s (hosts) top tracks.

## 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, this is due to an authentication error currently at hand. This will be fixed soon).

# Client/src/containers

## currPlaying

### currPlaying.jsx

Displays the currently playing song, which uses the props from previous classes which hold the album art, track name and artist.

## mainSection

### mainSection.jsx

This class is called in the main room files to display the username and display image (as stated, currently in the works) and the buttons. The buttons are composed in <playBox/> and the username and display image in <Top/>.

# Client/src/dataHandler

## Store/actions

Actions carry information from the current application to store. This comes in hand with reducers, in which reducers use the actions to determine an applications state change.

### Axios.js

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

### libraryActions.js

This consists of contant (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 const’s with use 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 contains current song 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 a bit 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 when it comes to 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 hasn’t been successfully fetched.

# Room-service/api/clients

## App.js

The App.js portrays everything mentioned to the user. It also makes sure the token is caught, and the device is checked. As long as these two work fine, the data should be displayed to the user.