\*English is not my first language, please forgive me for any mistakes. I would appreciate if you might want to paint them out for me so I could correct them in the future.

Feature Request: Callback information

**November 13th 2017**

**OBJECTIVE**

To give users a callback information about application working progress and accomplished processes.

**BACKGROUND**

This feature accomplishes the following:

When the user actives any action like search or save a playlist, a loading/saving icon will appear.

When the playlist is saved in the Spotify account a message box will appear with text – your ***playlist has been saved successfully***.

When jamming meets any difficulties to accomplished the order – message box will show an appropriate feedback information explaining what went wrong and advised the best solution to solve the problem.

For example:

When the app isn't able to find the track due to a typo or name mistake – message box will display information that it could not found the specified song and will suggest a solution - try with different track name.

When a user tries to save a playlist without typing it a name or adding any track into it, a message box will inform that to save new playlist there should be at least one track attached and some playlist name specified.

When a user saves a new playlist, the message box will inform whether the saving process has accomplished successfully or not.

In ordered to hide a message box user will have to click on the OK button in the message box window, which will be located just under the text message.

**TECHNICAL DESIGN**

**Show loading/saving icon**

A new component, **Button**, should be created. This button will trigger Spotify AJAX methods and show animated loading and save icon. The **Button** component will have a state object with two keys: ‘**name’** and **‘isActive.'**

Inside the **Button** component in the render function we will add **<a> </a>** tag with the **onClick** attribute that will call one of the parent’s method - **this.props.onSave()**.

Inside the **SearchBar** and **Playlist** component, we will have to replace the **<a>** tag with the **<Button />** element. Our **Button** should occur in two versions. A first one is a version that will trigger the **Spotify.search()** method, the second one will fire the **Spotify.savePlaylist()** method. Each version will have a different name passed through props from higher components - **name=’SAVE’** and **name =’Save to Spotify.'** The **name** will refer to default name of the button and will be stored in a Buttons' state object under the key of a **name**. **This.state.name** will be rendered as a text inside the <Button/> element.

While waiting for the AJAX response Buttons' name will switch to the loading/saving animated icon. To show loading or saving icon inside the button we will add a new method to a Button component called **showIcon()** that accepts one parameter **active**. This method will update the Buttons’s state **name** and change it from string into the URL address of loading/saving icon. As soon as the AJAX Promise resolves, **name** will turn back to it’s default name.

Inside of **showIcon(active)**, we will create a const variable called **defaultName** and store the button’s default name inside.

**defaultName = this.props.name**

Under the **defaultName**, we will create an if statement that checks if the **active** parameter is valid.

If the condition is right, we have to update the state using setState(). Inside we will assign the **name**: to the URL address of our animated loading/saving icon. If the initial condition were not true, we would update the state’s **name** to the value of const variable **defaultName** created at the top of the **showIcon** method.

To run our logic, we will have to modify a bit the App component. First of all, we need to add two more keys to the state object:

This.state = {

…………………,

…………………,

**searchInProgress :fasle,**

**savingInProgress: false**

}

Now in the body of App component we will create a method called **.activeProgressState** that accepts two parameters **progress** and **active.**  We will use it to update newly created states. Inside the method we will take a **progress** parameter that will stand for a **key** that we want to update while **active** will refer to key’s new **value**.

**activePogressState(progress, active) {**

**this.setState({[progress]:active});**

**}**

The purpose of **.activeProgressState()** is to change the proper state **(searchInProgress or savingInProgress)** from **false** to **true.** Now we can add **.activeProgressState()** to the beginning of the methods that ivokes after the **button** is clicked. (And those will be **search()** and **savePlaylist()**.)

Let’s add the **.activeProgressState()** at the very beginning of **savePlaylist()** method with two parameters, first would be a string that refer to the app’s state obejct key that we want to update. Because we are in savePlaylist method , we will use ‘**savingInProgress**’ string as a parameter. The second parameter will stand for the **savingInProgress ‘s** value, and this would be a boolen - **true.** When the button is clicked, the **savingInProgress** state will change to **true** and this will be a signal for button to change it’s defalutName to the loading/saving icon. To achieve that effect last thing that we have to do is to pass this state through props from **<App.>** down to the **<Button/>**.

We will give the **<searchBar/>** andthe **<playList/>** components an atribute of **isActive.** In **<seachBar/>** will pass **this.state.searchInProgress** while in the **<playList/>** **this.state.saveInProgress**.

Once we have activated our **savingInProgres** process we have to choose a place inside **savePlaylist()** where we will switch it back to a false value. The best place to do that is inside the Promise:

**Spotify.savePlaylist(playlistName,tracksToSave)**

**.then((resposne) => {**

**this.activePogressState(‘savingInProgress’, false)**

**}**

When the response is received we know that the process is completed. This the time when the button will turn back to it’s previous displayed default name.

In the search() method we will have to repeat accordingly same steps so the ‘SEARCH’ button could display searching icon. Thats all for know that had to be done is App component, is time to go back to **<Button/>** and put power to our **shwoIcon()** method.

Meanwhile in the **Button component…**

To call **shwoIcon()** at time of updated props arrives we should add **componentWillReceiveProps(nextProps)** lifecycle method to **Button**.

Inside the method we will start form **if statement** that check is if newly send data are different form present.

**if(nextProps.isActive !== this.state.isActive)**

If condition is true we will update state’s **isActive** to the value from **nextProps.isActive** and will call **shwoIcon(nextProps.isActive)** with **nextProps.isActive** as a parameter.

Finally when the button is clicked, animated searching/saving icon will show and will switch back to the buttons’ default name when the AJAX response process is complete.

\*remember to bind the current value of this to the newly created methods wherever it is necessary .

**Feedback message**

Once we have finished loading/saving icon issue, it’s time to work on feedback message. Again, a new component called **<MessageBox/>** should be created.

Inside the render function, after the return statement of newly created **MessageBox** component we will add a **<div></div>** container with **<h3></h3> and <a></a> tags** inside. H3 tag will receive a text from this.props.text, while inside <a> tag we will just type ‘OK’ text. <a> tag will play a role of confirmation button that reset the MessageBox when clicked – make it disappeared.

To achieve that we will give the <a></a> an onClick attribute equal to = {this.handleClick}. Next, we will add handleClick() to the body of the component. Inside the method we will trigger a function that comes through props:

**this.props.resetMessage()**

Last thing that need to be done before moving on is to bind the current value of this inside to constructor body. Now let’s move to the <App/> to put everything together.

Inside the <App/> first thing we have to do is to give App’s state a new parameter - **message: ‘’** that is equal to an empty string. Then we need to add two methods **resetMessage()** and **showMessage()** to the App’s body. That second one accepts one parameter **message**. Inside the **resetMessage()** we want to reset a **message** state into the empty string **‘’**. In the **showMessage()**, on the other hand, we will update the state’s **message key** to the string that comes in with the **message** parameter. Once two methods are created it’s time to place it in the code wherever we want to give a user a text feedback message.

Let’s put our newly created method inside **savePlayslit()** method. We want to inform the user whether his playlist was saved or not. Inside the promise chain, when the **Spotify.savePlaylist()** call resolves we should add an **if statement** that checks if **response.sanphot\_it** is true. If so, it means that playlist had been saved and we can trigger our **showMessage()** method with a string-message as a parameter - ***'Your playlist has been saved successfully'***. If the condition is not true we can inform the user showing an appropriate string-message passed as a parameter to **showMessage()**.

Using same logic we may add **showMessage()** in search() method. I think that it would be a good idea if we informed the user about the following situations:

- If no term had been provided before search button was used, we might display messageBox advising ***'Please specify what we are looking for.'*** When the **Spotify.search(term)** resolved with an empty array because of typo mistake, we would display a message '***Could not found the song you are looking for, please try with different name.'***

- Once we know where we want to display our **messageBox**, we should use it in the render method and set it to appear only when necessary. We want to render out **<MessageBox/>** depending on the state of the App’s message.

Inside the render method but before the return statement we will declare a let variable **message** equal to **null**. Then we will write an **if statement** that checks for **(this.state.message**). If condition evaluated to **true**, then we would assign the message to the **MessageBox** component.

message = <MessageBox

text={this.state.message}

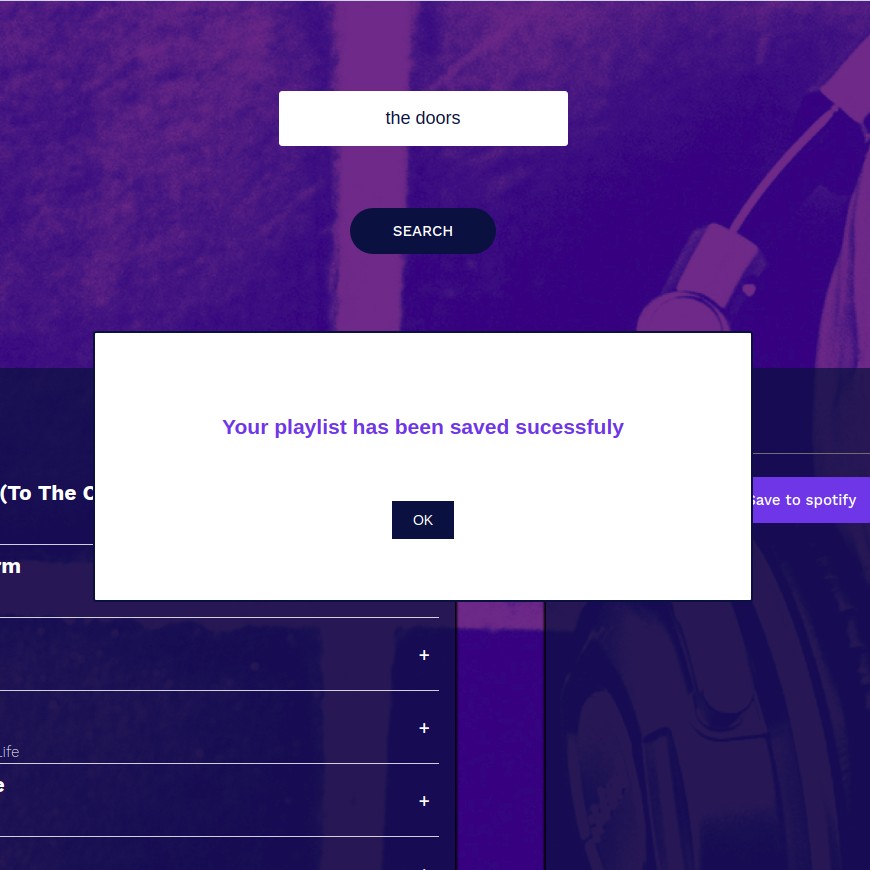
resetMessage={this.resetMessage}/>

}

MessageBox component will consist two attributes called **text** and **resetMessage** where we will pass **this.state.message** and **resetMessage()** down to the MessageBox component.

Finally, we will place the **{message}** variable inside the return statement, just after the <div> opening tag. This logic will display our **MessageBox** when message state is set to a string and will hide it when message is empty.

This component should look as follows upon implementation:



When we click OK button, **messageBox** will disappear.

**CAVEATS**

**Button implementation**

In our example, we use the button component only for search and save tracks purpose. There are still few components that use just the <a></a> tag with onClick attribute. If we decide to develop the functionality of the jammming app by adding more components end expand existing ones we should consider creating universal button component. It might require rebuilding the button component accordingly. We could also extract two main groups of buttons for those that use animation and for those that are static.

**MessageBox**

Currently, we have got just a few message to announce during user activity, so the MessageBox should not appear very often. But in case of application extension message announcements might increase. In that situation popping quite frequently, message window could become annoying. To avoid that we could rebuild the MessageBox to behave more subtly, and try to insert it into the primary application structure.