**SPOTIFY CLONE**

**CLIENT SIDE**

**LOGIN COMPONENT**- URL visited when the login is clickes. The code is extracted using t URLSearchParams(window.location.search).get('code'),

**APP.JS** - if code is available render dashboard with code prop, else render login page

**USEAUTH HOOK** -handles authentication. It is called in the Dashboard component when the code is sent after clicking the login button. It returns an accessToken for the spotify api.

Whenever the code changes, a useEffect hook is fired, in which a post request is sent to the server which is listening on port 3000. The code is sent with the post request in the body.A response from the server is received with contains an accessToken, a refresh token and an expiresIn Item.

Another useEffect is fired whenever the refreshToken, or expiresIn Properties change. It gets fired when the refreshToken and accessToken are set initially, and sets a 59 minute interval to serve a refresh post request. This hook sends a post request to the server and sends the refreshToken (after 59 minutes). The interval is cleared after the post request is served. The server sends back a new access token and a new expires in time.

**DASHBOARD**

Contains a search bar, music player and AREA for displaying lyrics. An instance of the Spotify Web API is also initialized on the front end in the Dashboard component, using the clientId & the accessToken.

1. Gets access token from useAuthHook

2. First useEffect hook to handle fetching search results. Only runs when there is and accessToken. A cancel token is initialized as false. The search method is called on the Spotify Web API. While the search term keeps changing the cancelToken is set to true. When the search term is finally set (doesn't change), the searchTracks function is called on the spotify web api item. For each search result, a few images are returned. The smallest image is selected using the reduce function by comparing the height of the images. Each result is mapped to return an object containing information about the returned track and set to the searchResults state.

3. Second useEffect sets the webApi access token after it is set through refreshing the expiresIn time.

4. The third useEffect is sends a get request to the server to retrieve the lyrics of the selectedTrack. The params sent are the track and artist. The results are set to the lyrics state.

**PLAYER**

SpotifyPlayer component is imported from the react-spotify-web-playback library. The player is initialised with the accessToken and the trackuri. It also takes the play property(boolean), uris, showSaveIcon and the callback property which sets play to false if state.isPlaying is false

**SERVER SIDE**

The server listens for requests on port 3001.

**LOGIN POST REQUEST**

**1. A code is received throug**h the request body, for authentication.

2. A new Spotify Web API object is created and the authorizationGrantCode method is called to authenticate the code. Upon successful authorization, a data object with accessToken, refreshToken is received and is sent to the front end via a json response.

**REFRESH POST REQUEST.**

1. Similar to the login post request, but a refresh token is received in the request body,

2. A new instance of the Spotify Web API is created, using the process.env properties, AND THE REFRESHTOKEN.

3. The refreshAccessToken() method is called on the Spotify Web API instance and a promise resolved with an object containing a new access token and a new expiresIn object.

**LYRICS GET REQUEST.**

When a get request is received, an async function is called (async (req, res). Where we await the result of the lyric finder which uses the req.query values sent via the get request params. The result is sent as the request. In order to parse the params, we use the app.use( bodyParser.urlencoded({extended:True}) middleware.

**MIDDLEWARE USED ON SERVER**

1. app.use(cors())

2. app.use(express.json())

3. app.use( bodyParser.urlencoded({extended:True})