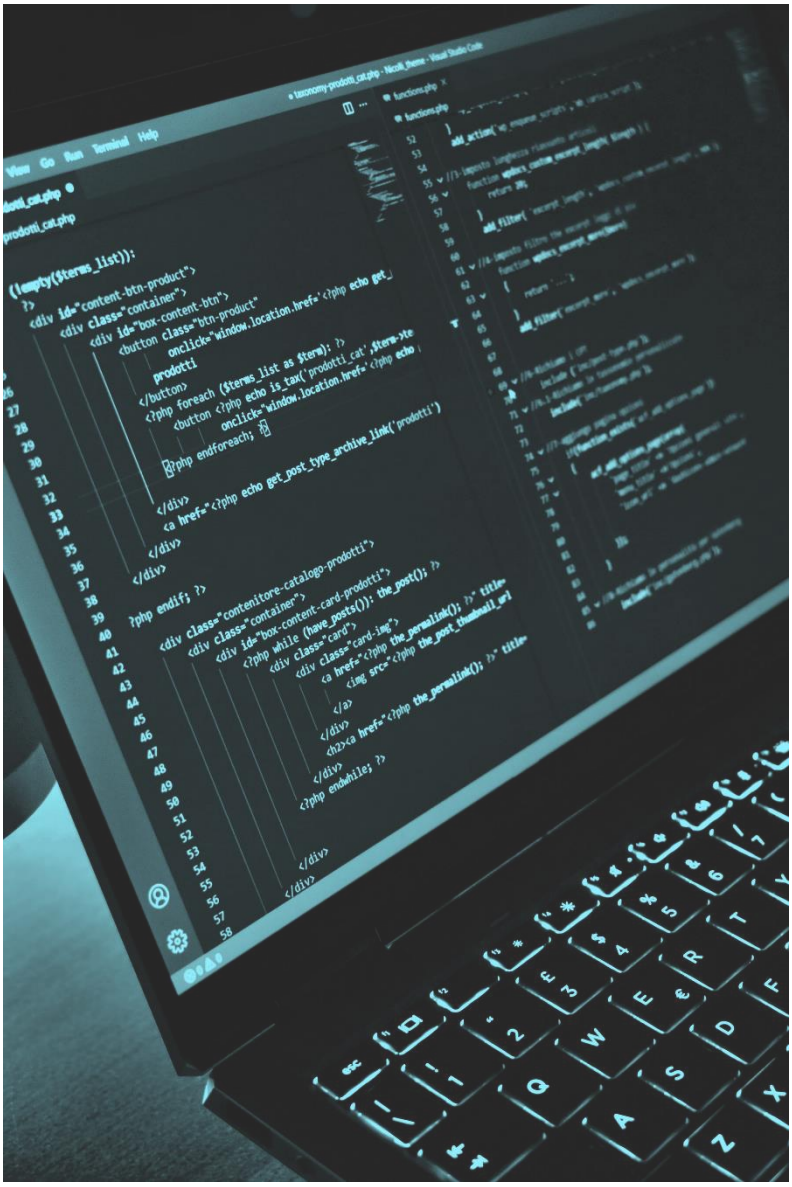


GCMO

GESTORE COMUNITÀ MUSICALI ONLINE

Roberto Tallarini | Applicazioni Web & Cloud | UNIMI

Febbraio 2023

Sommario

Introduzione al progetto

GCMO (gestore comunità musicali online) è una piattaforma web che offre una soluzione semplice e intuitiva per gestire, creare e modificare delle playlist musicali utilizzando le API di Spotify. I suoi utenti possono accedere tramite il loro account Spotify, selezionare le loro preferenze musicali, salvare artisti, tracce, album e playlist nel loro profilo. La piattaforma offre anche la creazione, modifica e cancellazione di playlist pubbliche o private, la ricerca avanzata di brani, album, artisti e playlist, nonché la visualizzazione e la possibilità di seguire i profili di altri utenti.

FUNZIONALITÀ DELL'APPLICAZIONE

L'applicazione GCMO è stata progettata per offrire una vasta gamma di funzionalità che soddisfano le necessità degli utenti che usano un sito di gestione delle playlist musicali.

Qui di seguito una lista di tutte le funzionalità implementate:

- Registrazione, Login e Logout
- Scelta delle preferenze Musicali dell'utente
- Caroselli di Playlist e Tracce Suggerite per l'utente
- Visualizzazione dell'elenco di Playlist dell'utente (con Filtri)
- Ricerca delle Playlist dell'utente
- Ricerca di Playlists, Tracce, Album e Artisti (con Filtri)
- Ricerca di Playlists (per Categoria/Tag)
- Creazione di Nuove Playlists (con titolo, immagine, descrizione e tag, visibilità)
- Modifica di Playlists
- Eliminazione di Playlists (unfollow)
- Aggiunta di Tracce musicali ad una Playlist
- Rimozione di Tracce musicali da una Playlist
- Follow/Unfollow di Playlists pubbliche
- e Albums in una nuova Playlist personale
- Follow/Unfollow di Importazione di Playlists Artisti, Albums, Tracce musicali
- Visualizzazione delle Informazioni relative ad una Playlist e delle Canzoni in elenco
- Visualizzazione delle Informazioni relative ad un Album e delle Canzoni in elenco
- Visualizzazione delle Informazioni relative ad un Artista
- Link verso l'applicazione di Spotify per ogni Traccia, Album o Playlist
- Visualizzazione delle Informazioni relative ad un altro utente e delle Playlists in elenco
- Follow/Unfollow di un utente
- Visualizzazione delle informazioni dell'Utente corrente
- Modifica delle informazioni relative all'Utente corrente
- Cancellazione Account / Revoca delle autorizzazioni dal sito da parte dell'utente corrente
- Visualizzazione di alcuni suggerimenti di Spotify (playlists, canzoni..)

Tecnologie utilizzate

Il progetto GCMO Gestore Comunità Musicali Online è stato realizzato con una combinazione di tecnologie web, alcune delle quali sono state scelte per semplificare e velocizzare il processo di sviluppo, mentre altre sono state scelte per fornire un'esperienza utente più gradevole e personalizzabile. In questa sezione, verranno descritte e motivate le tecnologie utilizzate nel progetto.

JAVASCRIPT

JavaScript è stato utilizzato come linguaggio di scripting principale del progetto. Questo è stato scelto perché è uno dei linguaggi di scripting più popolari e flessibili, ed è supportato da molti framework e librerie web. Inoltre, JavaScript è un linguaggio dinamico e di alto livello che permette di creare funzioni complesse e interattive in modo semplice e rapido.

REACT

React è una libreria JavaScript progettata da facebook basata su Node.js per la costruzione di interfacce utente dinamiche e interattive. Si basa sulla filosofia della single page application; il che significa che ogni azione avviene all'interno della stessa pagina web senza bisogno di ricaricare la pagina.

In una single page application il codice è incapsulato in componenti riutilizzabili; ogni componente può essere utilizzato più volte in diverse parti dell'applicazione web. Questo modello di sviluppo rende il codice più pulito, ordinato e facile da mantenere, migliorando la scalabilità del progetto.

Il linguaggio utilizzato con React si chiama JSX, ed è un'estensione di JS che combina il linguaggio di markup HTML e JavaScript. Ogni componente quindi può essere scritto in JSX, inserendo il codice HTML direttamente nel codice JavaScript. Combinato con la libreria Bootstrap e il linguaggio CSS, JSX consente di costruire la struttura, lo stile e lo script di ogni componente singolarmente.

React utilizza un Virtual DOM (document object model virtuale), che differisce dal DOM HTML. Il DOM è una rappresentazione in memoria della pagina web, quando un elemento della pagina viene modificato, il browser deve apportare le modifiche al DOM, e l'intera pagina viene ri-renderizzata. Il Virtual DOM di React invece offre un modo più semplice di gestire la modifica dei componenti da renderizzare all'interno di una pagina poiché è una rappresentazione virtuale in memoria del DOM; quando un elemento viene modificato, React confronta la versione attuale del Virtual DOM con la versione precedente per determinare quali parti devono essere effettivamente modificate nel DOM reale. Questo rende il processo di rendering molto più efficiente, poiché solo le parti che devono essere effettivamente modificate vengono apportate al DOM e ri-renderizzate. Questo permette un facile scambio di componenti all'interno di una single page application.

I componenti utilizzati nel progetto sono Functional Components, sono stati scelti come approccio di sviluppo poichè hanno come vantaggio rispetto ai componenti Class di avere un codice più snello e una facile gestione dello stato tramite Hook. Un hook è una funzione in React che permette di gestire lo stato e gli effetti collaterali in un componente. Gli Hook principali utilizzati sono useState che permette di gestire lo stato locale di un singolo componente e useEffect che permette di eseguire un effetto collaterale in un componente al cambiamento del suo stato o di una variabile. Infine, per gestire uno stato condiviso in ogni componente, compare nel progetto anche uno useContext che fornisce un modo semplice per condividere informazioni tra componenti padre-figlio senza il passaggio di props manualmente.

In sintesi, React è stato scelto perchè si tratta di una libreria concorrenziale e richiesta sul mercato del lavoro per lo sviluppo web e perché grazie ai suoi componenti riutilizzabili e a un codice più pulito, ha migliorato la gestione del progetto con un rendering più efficiente e una semplice gestione dello stato dei componenti tramite gli Hook.

REACT ROUTER DOM

React Router DOM è una libreria che semplifica la gestione delle route in un'applicazione React, permettendo di creare una navigazione fluida tra componenti di pagina diversi basandosi sull'URL corrente. Ogni URL viene associato ad un componente pagina, che può contenere un insieme di componenti secondari. In base all'URL corrente, una pagina viene selezionata e renderizzata all'interno dell'applicazione, garantendo una single-page experience all'utente.

CSS E BOOTSTRAP

CSS, e Bootstrap sono stati utilizzati per progettare e dare forma all'interfaccia utente del progetto. Bootstrap è una libreria di componenti CSS pre-costruiti che permette di creare rapidamente interfacce utente con stili e animazioni pronti all'uso, comunemente utilizzati e riconosciuti dagli utenti, con un'interfaccia piacevole e intuitiva.

CSS è stato utilizzato per personalizzare ulteriormente l'aspetto e la funzionalità dell'interfaccia utente rendendo la pagina graficamente adattata allo stile complessivo del progetto.

NODE.JS

Node.js è stato utilizzato come piattaforma per la costruzione di un piccolo backend che consente di gestire l'autenticazione con Spotify. Questo è stato fatto per nascondere il client secret delle API di Spotify dal codice sorgente pubblico della pagina dell'utente. Node.js fornisce un'infrastruttura potente e scalabile per la costruzione di backend, consentendo di eseguire operazioni e gestire richieste HTTP con semplicità. Inoltre, la sua popolarità e la vasta quantità di librerie disponibili hanno reso questa scelta la più logica per questo progetto.

AXIOS & SPOTIFY-WEB-API-NODE

Axios e Spotify-web-api-node sono due librerie JavaScript che sono state utilizzate per la gestione delle richieste HTTP all'interno dell'applicazione web. Axios è una libreria che semplifica la creazione di richieste HTTP, ed è stato utilizzato principalmente per la comunicazione tra il backend e il frontend dell'applicazione per quanto riguarda la gestione della login e del refresh dei token.

Al contrario, Spotify-web-api-node è una libreria ufficiale specifica per le richieste all'API di Spotify, che astrae le richieste e fornisce un'interfaccia semplificata per lavorare con l'API di Spotify. Queste due librerie sono state scelte perché offrono una soluzione semplice e intuitiva per gestire le richieste HTTP all'interno dell'applicazione web.

L'utilizzo di tutte queste tecnologie permette di ottenere una soluzione più avanzata e personalizzabile rispetto a una soluzione basata solo su HTML, CSS e JavaScript. Questo è importante soprattutto per progetti che richiedono funzionalità complesse, un'interfaccia utente accattivante e un'esperienza utente fluida.

Architettura dell'applicazione

GCMO (gestore di comunità musicali online) è un'applicazione frontend che comunica tramite http con le API REST di Spotify per raccogliere tutti i dati e le informazioni che alimentano la piattaforma. Questo approccio di sviluppo presenta sia vantaggi che limitazioni che sono state gestite nel modo più opportuno in base alle possibilità offerte dagli endpoint di Spotify API.

AUTENTICAZIONE

L'autenticazione dell'utente alla piattaforma GCMO avviene tramite Spotify API.

Spotify fornisce diversi metodi per effettuare l'autenticazione; alcuni di essi sono:

1. **Client Credentials Flow**: Server-To-Server
   Questo metodo non richiede l'autorizzazione da parte dell'utente, vengono invece sfruttati client ID e client Secret dell'applicazione per accedere ai servizi delle API. Non essendoci autorizzazione ne interazione dell'utente non permette quindi una gestione delle playlist dell'utente loggato.

2. **Implicit Grant Flow**: Autorizzazione temporanea
   Questo metodo non richiede l'utilizzo di una secret key e autorizza temporaneamente l'applicazione a gestire i dati dell'utente restituendo direttamente un access token (durata 1 ora), tuttavia non permette il refresh dell'access token ottenuto.

3. **Authorization Code Flow**:
   L'authorization Code Flow è un processo a due fasi in cui l'utente viene prima reindirizzato alla pagina di autenticazione di Spotify, dove può autorizzare o negare l'accesso alle proprie informazioni, poi, se l'autorizzazione viene concessa, Spotify restituirà un codice che potrà essere scambiato con un token di accesso attraverso una richiesta API. Questo metodo è utilizzato per garantire che solo l'applicazione autorizzata abbia accesso alle informazioni dell'utente e permette di eseguire il refresh del token una volta che quest'ultimo sarà scaduto garantendo un'autorizzazione "permanente" dell'utente.

   La più corretta e sicura implementazione dell'Authorization Code Flow prevede l'utilizzo di un server secondario adibito alla fase di scambio del codice di autorizzazione con l'Access Token e alla fase di Refresh del Token. Entrambe queste fasi hanno infatti bisogno di uno scambio con le API di Spotify del Client Secret, ovvero di un codice segreto relativo all'applicazione che per ragioni di sicurezza vogliamo conservare nel Server e non condividere mai con il codice sorgente del lato Client.

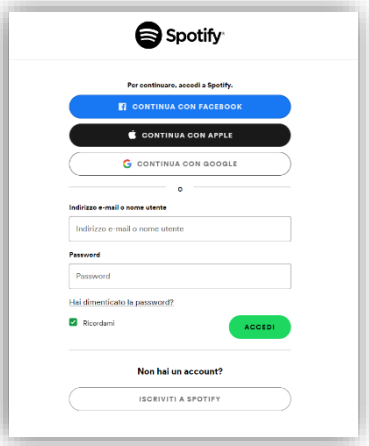
La scelta implementativa del progetto verte quindi per i precedenti motivi sull'Authorization Code Flow e l'utilizzo di un server backend di supporto all'autenticazione dell'applicazione; l'autenticazione e la registrazione sono delegate a spotify, mentre il server backend gestisce le chiamate necessarie per l'ottenimento dei token.

Siccome il progetto scelto per l'esame richiedeva di concentrarsi sul frontend dell'applicazione ho deciso complice della poca esperienza con il backend di affidarmi ad un Server Node preimpostato rilevato da GitHub. Il server predispone due endpoint che offrono la possibilità di ottenere e refreshare l'access Token sfruttando la stessa libreria utilizzata per il frontend per le comunicazioni con le API di Spotify (Spotify-web-api-node).

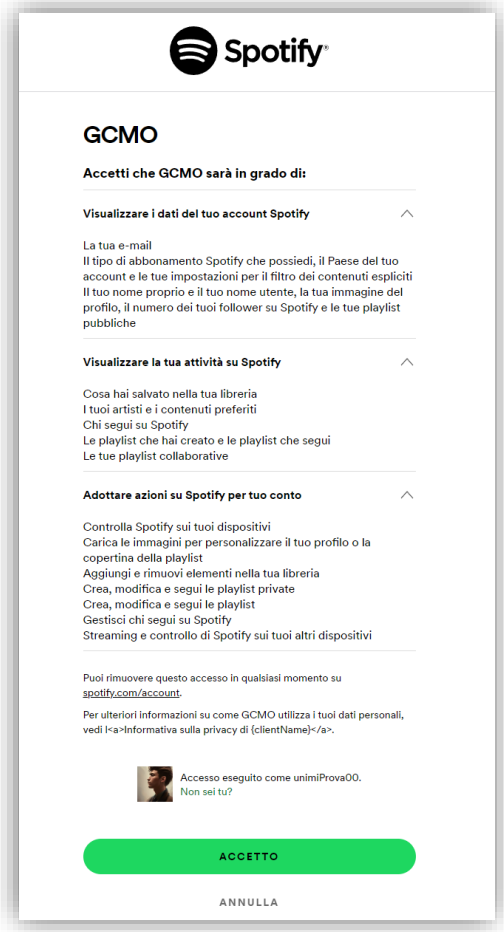
L'oggetto restituito a seguito delle chiamate del Server è un oggetto che include l'Access Token, il Refresh Token e l'Expires_in ovvero la durata massima del token restituito (1 ora).

A partire dal lato backend descritto ho quindi iniziato a sviluppare il progetto frontend in React che è in grado di comunicare con le due REST API (il back-end e le Spotify-API) per garantire una completa gestione delle informazioni degli utenti loggati.

Descrizione dei Passaggi di Login / Registrazione



1. L'utente clicca sul pulsante di Login nell'applicazione GCMO e viene reindirizzato alla pagina di **autenticazione** di Spotify; l'autenticazione dell'utente avviene tramite le credenziali di Spotify.



2. **L'utente autorizza o nega l'accesso**: Una volta reindirizzato alla pagina di autenticazione ed effettuata l'autenticazione con Spotify, l'utente viene reindirizzato alla pagina dove può autorizzare o negare l'accesso del sito alle proprie informazioni. L'URL di reindirizzamento include il Client ID dell'applicazione e gli Scopes (informazioni dell'utente etc..) di cui l'applicazione richiede l'accesso.

3. Se l'autorizzazione viene concessa, Spotify restituirà un codice univoco di autorizzazione nell'URL che reindirizza l'utente alla pagina web

4. **Scambio del codice con un token di accesso**: Il codice di autorizzazione viene estratto dall'URL e viene inviato nel body di una richiesta http con Axios all'endpoint di login del server backend.

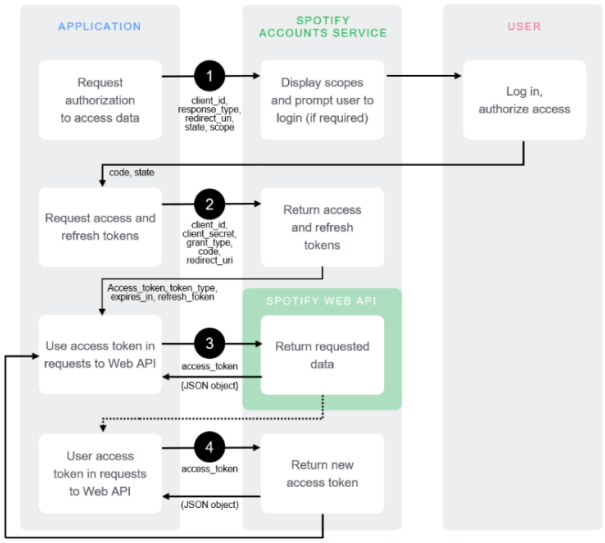
5. Il server esegue una chiamata alle API di Spotify con spotify-web-api-node inviando Client_ID, Client_Secret e il Code ricevuto dal frontend. Come risposta riceverà un oggetto contenente L'accessToken, il refreshToken e l'expires_in.

6. Il server invia cosi la risposta al client ed il client archivia l'access token e il refresh token nel Local storage. Ogni volta che il frontend vorrà chiedere delle informazioni alle API di Spotify attingerà al token e lo invierà nella richiesta.

Refresh dell'accessToken

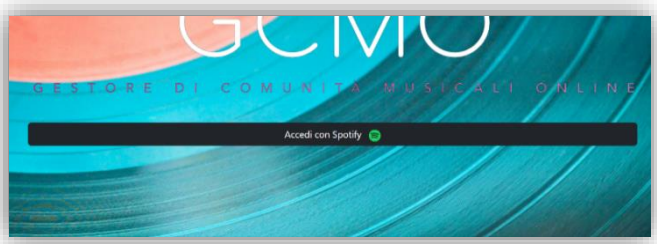
Se il l'access Token scade (dopo 1 ora) le chiamate alle API di Spotify restituiscono l'errore con codice 401, in questo caso il client esegue la funzione refreshToken(); questa funzione invia una richiesta http con Axios compresa di refreshToken all'endpoint di refresh del server backend. Il server allora invierà una richiesta di Refresh alle API di Spotify contenente Client_ID, Client_Secret e refreshToken.

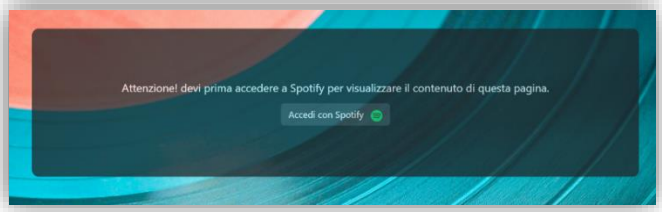
Le API risponderanno al backend con un oggetto contenente L'accessToken, il refreshToken e l'expires_in che verranno a loro volta restituiti al Client.



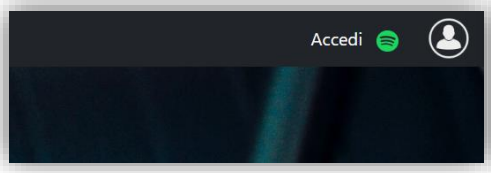
Scelte Implementative e di design

Le scelte di design per la registrazione, login e logout sono limitati ai pulsanti che permettono di accedere o uscire dal sito web, poiché l'accesso avviene attraverso la pagina di login di Spotify e il logout consiste semplicemente nella rimozione del token di accesso dal local storage. Ho implementato i pulsanti di login sia nella pagina principale che nella NavBar in alto a destra vicino all'immagine del profilo utente. Inoltre, i pulsanti di login sono presenti anche nei banner visualizzati nelle pagine quando l'utente non è loggato.





Il pulsante di logout si trova sempre e solo all'interno della navbar e sostituisce quello di login una volta eseguito l'accesso.



Se l'utente che accede al browser è già loggato in Spotify e ha già dato consenso agli scopes, l'accesso all'applicazione lo porta direttamente alla personalArea senza passare dalla index.

Se l'utente invece ha già dato accesso agli scopes ma è uscito dalla login, necessiterà di fare click sul pulsante accedi nella indexPage o in un qualsiasi altro pulsante di login che lo porterà direttamente alla personalArea.

Se per l'utente è invece la prima volta che accede alla pagina dal browser, verrà indirizzato alla pagina di scelta delle preferenze musicali. Solo in seguito approderà alla personalArea.

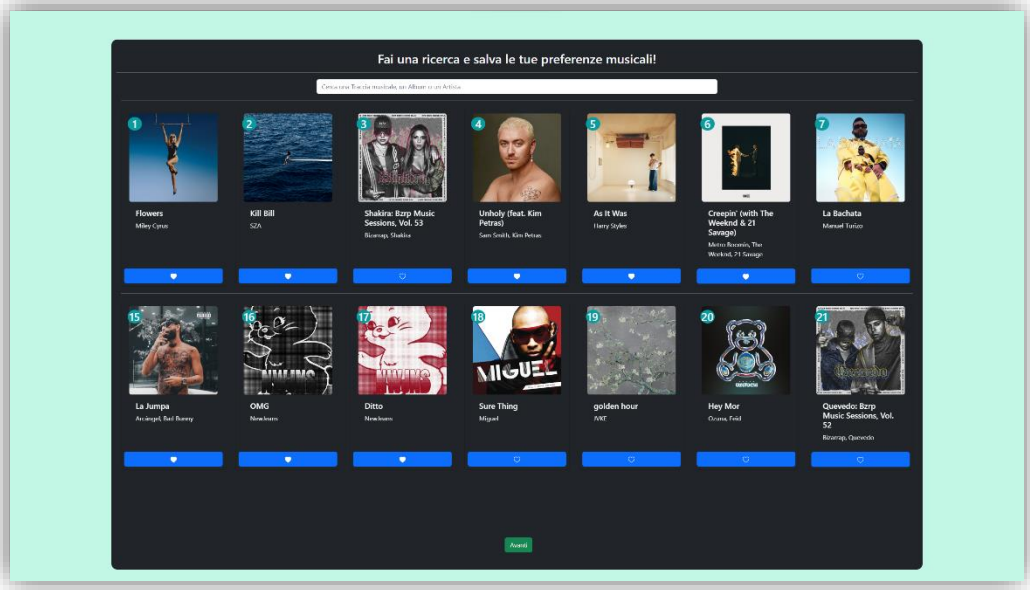
Questa differenza di percorso viene gestita tramite un valore speciale ("visited": "true") che viene salvato nel local storage dell'utente al primo accesso alla piattaforma. Se nei futuri accessi nel local storage è presente il valore speciale l'utente non passa dalla pagina per la scelta delle preferenze musicali.

Struttura dell'applicazione web

L'applicazione web include 4 percorsi di navigazione

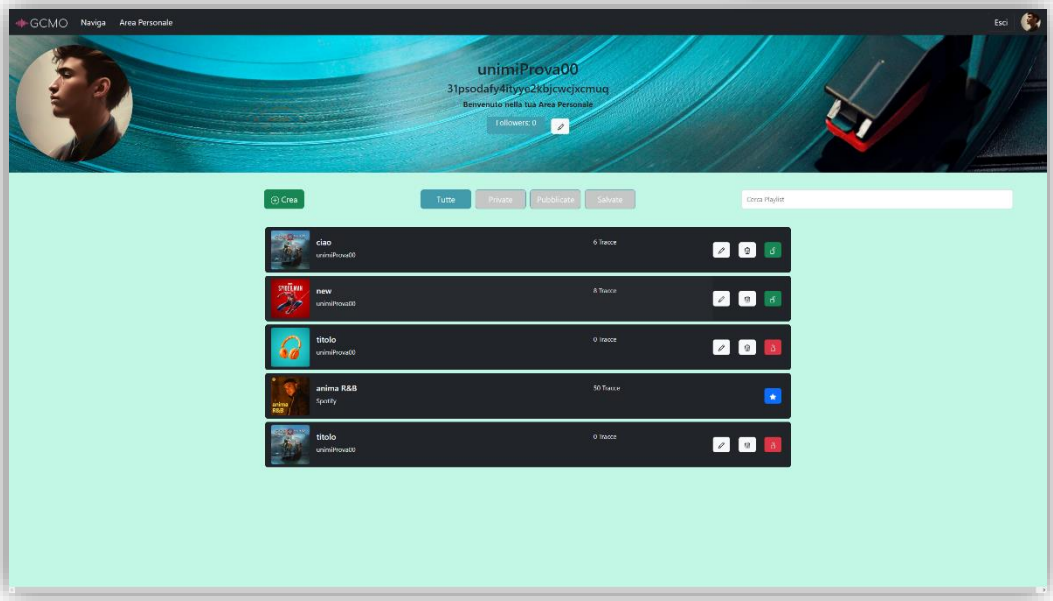
7. **Index Page**
   (la index Page è la prima pagina che si incontra nella navigazione, è una pagina minimale e presenta il logo del sito, la NavBar e il pulsante di Login)

8. **Pagina per la Scelta delle preferenze Musicali dell'utente**
   (la Pagina per la Scelta delle preferenze Musicali permette di fare ricerche e salvare le preferenze musicali dell'utente intese come artisti, albums e tracce musicali, la particolarità della pagina è che compare solamente al primo accesso browser al sito GCMO…)



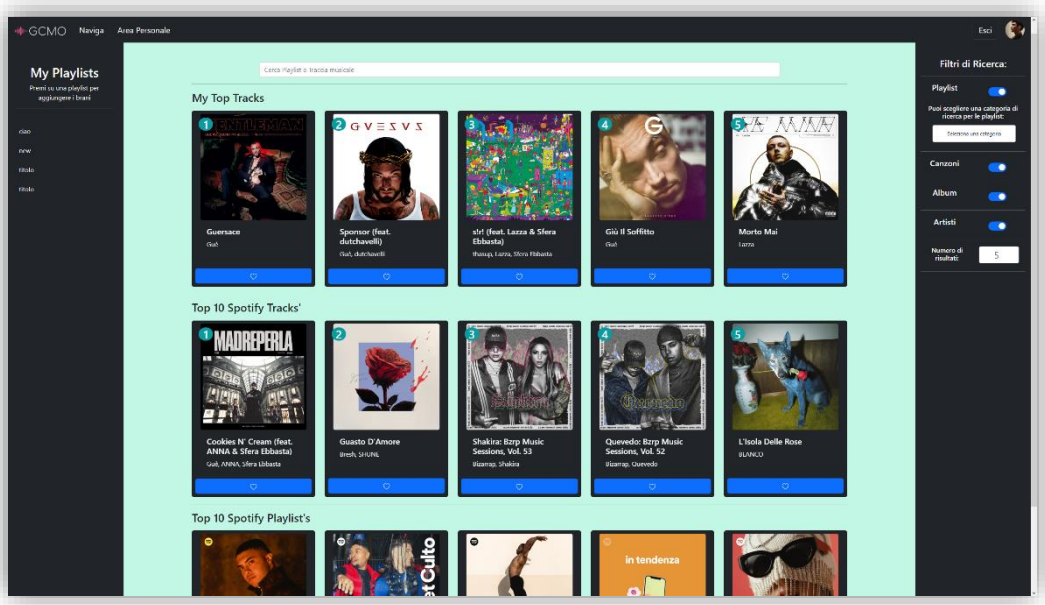
9. **Personal Area**

(se non si è loggati la personal Area presenta solamente un banner che avvisa l'utente di loggarsi, altrimenti permette di visionare le proprie playlists, crearne di nuove, modificarle, eliminarle, gestire le informazioni utente e molto altro..)



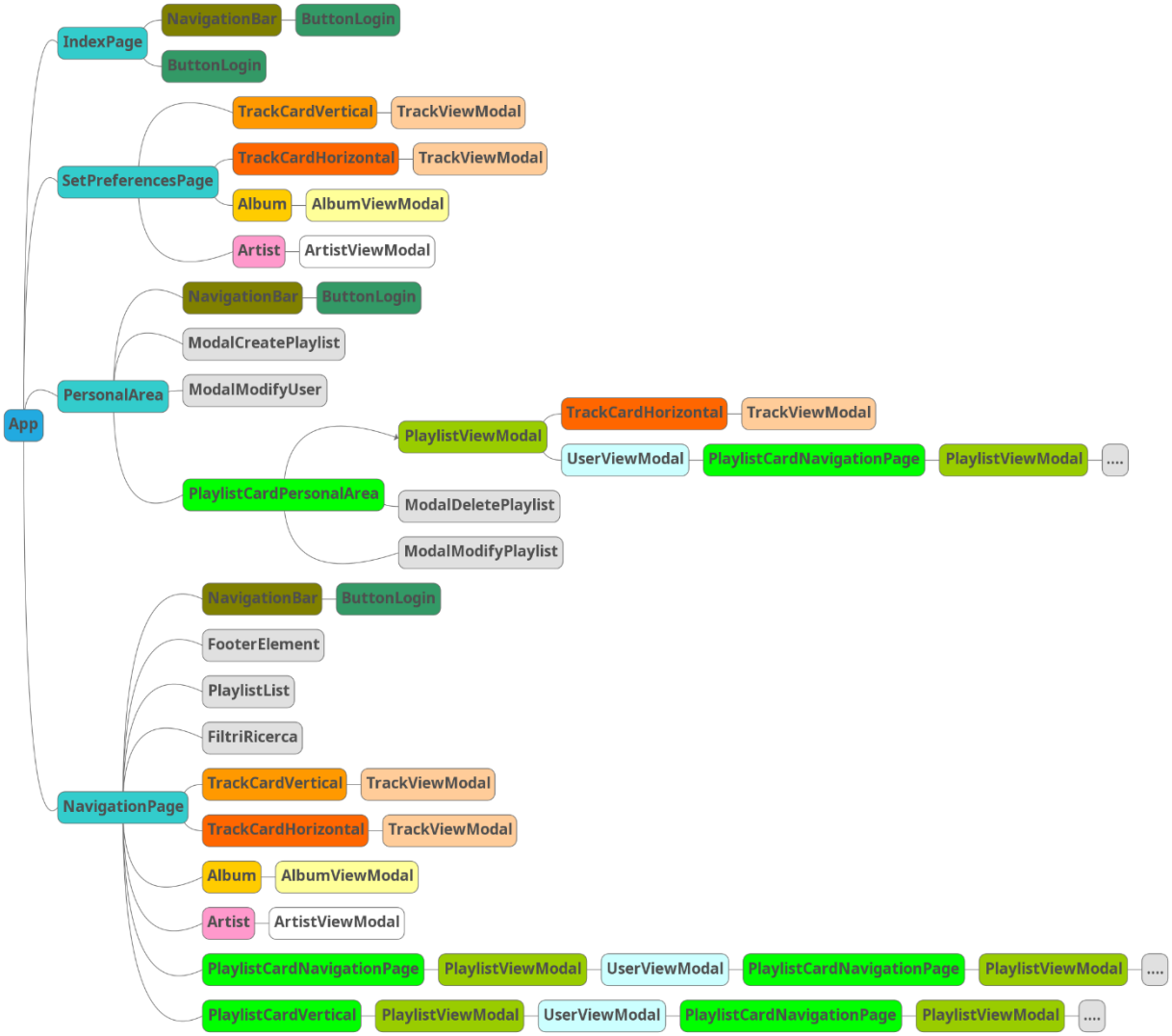
10. **Navigation Page**

(se non si è loggati la Navigation Page presenta solamente un banner che avvisa l'utente di loggarsi, altrimenti permette di fare ricerche, visualizzare suggerimenti, aggiungere tracce alle playlists personali e molto altro...)



ALBERATURA DEI COMPONENTI

Di seguito riporto l'alberatura dei componenti funzionali nelle pagine:



Elenco:

- App.js
- ToastNotify
- IndexPage
- NavigationBar
- ButtonLogin
- SetPreferencesPage
- TrackCardVertical
- TrackCardHorizontal
- TrackViewModal
- Album
- AlbumViewModal
- Artist
- ArtistViewModal
- PersonalArea
- ModalCreatePlaylist
- ModalModifyUser
- PlaylistCardPersonalArea
- PlaylistViewModal
- ModalDeletePlaylist
- ModalModifyPlaylist
- NavigationPage
- PlaylistCardVertical
- PlaylistCardNavigationPage
- FiltriRicerca
- PlaylistList
- FooterElement

Approfondimento sulle Funzionalità

SCELTA DELLE PREFERENZE MUSICALI DELL'UTENTE

Per la scelta delle preferenze musicali dell'utente ho predisposto una pagina apposita, la pagina viene visualizzata solo al primo accesso al sito web da un determinato browser; nella pagina di scelta delle preferenze è possibile salvare, visualizzare e ricercare tracce, album e artisti.

Oltre alla form per effettuare la ricerca è presente un box con 2 caroselli, ogni carosello permette di visualizzare 14 tracce, in totale vengono visualizzate quindi 28 tracce. Le tracce visualizzate nel carosello sono le prime 28 tracce presenti nella playlist top 50 creata da spotify, quindi in sostanza sono le tracce più ascoltate del momento.

La pagina viene strutturata assemblando diversi componenti utilizzati per le altre pagine ad esempio le modali e le cards per visualizzare artisti, album e tracce.

La ricerca è stata implementata in una variante differente rispetto alla ricerca nella navigation page, in questo caso infatti non si possono ricercare playlists, non ci sono filtri e i risultati di ricerca saranno sempre 10 per ogni categoria di oggetti.

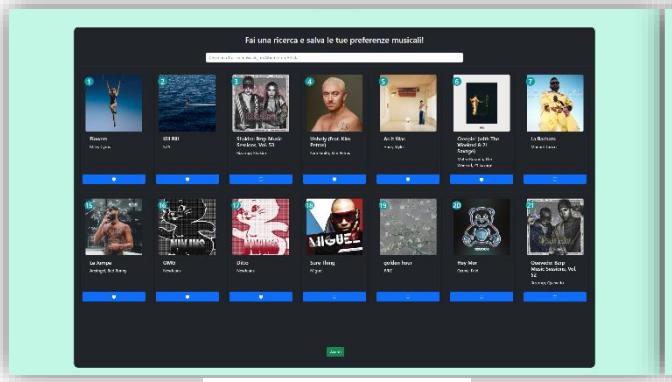
In fondo alla pagina troviamo un pulsante che permette di proseguire la navigazione, al click l'utente viene indirizzato alla personal area.

È importante specificare che operazioni quali l'import di album in una nuova playlist, quando l'url corrente corrisponde a quello della pagina di scelta delle preferenze vengono disabilitate.

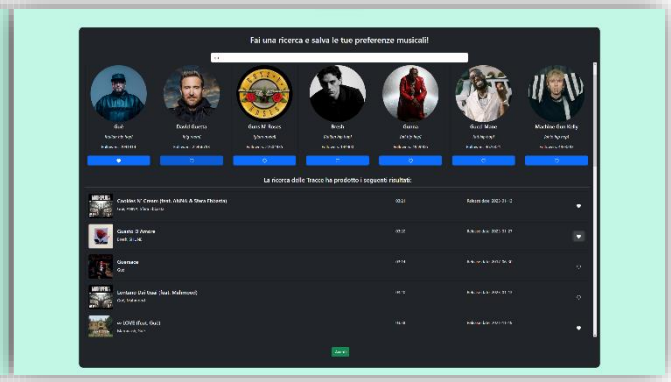
In seguito verranno approfonditi i componenti utilizzati nella pagina.

Scelte di design

Il design di questa pagina riprende i colori dell'intero sito web, lo sfondo è chiaro, in centro alla pagina compare un container dark che racchiude la form di ricerca e i caroselli delle top tracks. Quando viene effettuata una ricerca le top tracks scompaiono e al loro posto compaiono i risultati di ricerca che è possibile scorrere in verticale (scroll).



Caroselli Top tracks



Risultati di Ricerca

Il click su tracce, album o artisti apre delle modali ad essi relative. Ogni artista, album o traccia presenta un pulsante sulla card a forma di cuore vuoto per poter salvare la preferenza musicale. Se si clicca sul cuore quest'ultimo si riempie.

Implementazione

Come prima cosa viene impostato uno stato nel componente chiamato searchWord con una stringa vuota e uno state AccessToken. Ogni volta che l' accessToken o searchWord cambiano, viene eseguito l'Hook useEffect. L'Hook verifica se searchWord è una stringa vuota. Se lo è, viene effettuata una chiamata per recuperare le prime 28 tracce della playlist top50:

```
spotifyApi.getPlaylistTracks('37i9dQZEVXbMDoHDwVN2tF', {limit: 28, offset: 0})
```

verrà restituito un oggetto che al suo interno avrà una lista di tracce musicali, quindi viene eseguita un'iterazione sull'oggetto estraendo per ogni traccia le informazioni necessarie e componendo un nuovo oggetto. Ogni nuovo oggetto è inserito in ordine in un nuovo array. Dopodichè l'array verrà assegnato allo state topTracks.

Se topTracks non è nullo e searchWord è vuota vengono renderizzati 2 caroselli, ciascuno con 2 pagine, in ogni pagina viene eseguito un ciclo di 7 posizioni sull'array topTracks (0-6, 7-13, 14-20, 21-27), i 7 elementi estratti per ogni pagina vengono passati ognuno ad un componente CardTrackVertical per renderizzare la card della canzone.

Quando qualcosa viene scritto nella form, ogni cambiamento al suo interno viene intercettato come evento ed esegue il setter setSearchWord cambiando il contenuto di searchWord, quando il suo contenuto cambia l' Hook intercetta il cambiamento della searchWord e verifica che non sia vuota, se non lo è viene eseguita la funzione ricerca(); questa funzione esegue una chiamata alle api di spotify per recuperare 10 artisti, 10 tracce e 10 album basandosi sulla searchWord corrente:

```
spotifyApi.search(searchWord, ["track", "artist", "album"], {limit: 10})
```

la risposta conterrà varie liste di oggetti, vengono quindi eseguiti 3 cicli, uno per estrarre tutte le informazioni necessarie di ogni traccia e comporre l'array delle tracce, uno per gli artisti e uno per gli album.

Inoltre essendo la searchWord non vuota viene cambiato il rendering, in questo caso vengono renderizzati 1 carosello composto da 2 pagine con 5 artisti per ogni pagina, e 2 container con 10 risultati di tracce e album. Il procedimento è sempre un ciclo .map sull'array per estrarre ogni elemento. Gli album vengono passati al componente Album, gli artisti al componente Artist, e le tracce al componente TrackCardHorizontal.

Se la searchWord cambia nuovamente e diventa vuota ogni array viene svuotato e viene rieseguita getTopTracks().

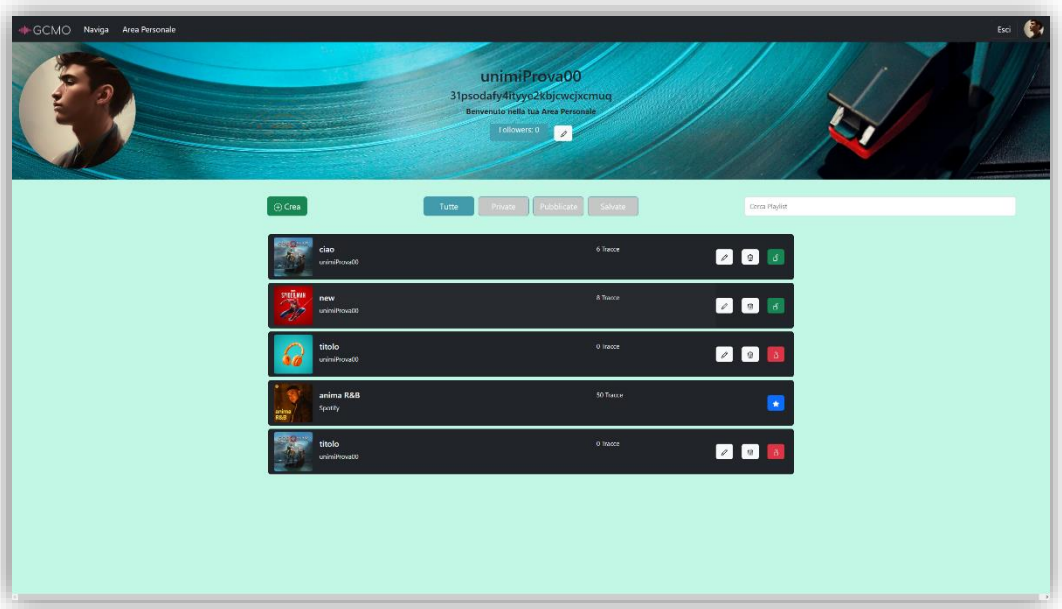
Ad ogni chiamata API se viene restuito un errore il catch esegue la funzione ErrorStatusCheck, che verifica il tipo di errore e visualizza degli alert, stampa a console, imposta un timeout o esegue il refresh del token.

Il pulsante avanti esegue una funzione che inserisce nel local storage il valore “visited”:”true”, dopodichè reindirizza l’utente alla personalArea. Il valore memorizzato non verrà rimosso al logout, e ai successivi accessi dal browser, se il valore è presente nello storage l’utente verrà indirizzato direttamente alla personalArea senza passare dalla SetPreferencesPage.

Tale controllo viene eseguito nel componente ButtonLogin.

PERSONAL AREA

La Personal Area è un'area riservata all'utente. Dopo il primo accesso, sarà sempre la prima pagina su cui si atterra una volta effettuato il login. Al suo interno, si possono trovare le informazioni relative all'utente, una barra di navigazione per passare da una sezione del sito all'altra e per effettuare il logout, un pulsante che permette di aprire una finestra modale con pulsanti che reindirizzano a Spotify per effettuare modifiche all'account o al profilo, un elenco di playlist che si possono filtrare utilizzando i filtri Tutte, Pubbliche, Private, Salvate, un modulo di ricerca per cercare tra le playlist dell'utente, infine c'è un pulsante che apre la finestra modale per creare una nuova playlist. Ogni playlist può essere visualizzata attraverso la sua finestra modale, modificata e cancellata. In ogni playlist è presente il nome dell'utente che l'ha creata, quindi si può anche visualizzare il creatore e le sue playlist. In ogni playlist si possono visualizzare le canzoni contenute al suo interno e le relative modali.



Le operazioni si basano sui dati forniti dagli endpoint delle Spotify API, quindi i limiti dell'applicazione sono legati ad essi:

- La cancellazione delle playlist nei server di Spotify non esiste, cancellare una playlist equivale a fare "unfollow" della playlist.

- L'API di Spotify non fornisce alcun endpoint per effettuare modifiche sul profilo utente o sull'account Spotify, pertanto la soluzione più logica è stata quella di collegare l'utente direttamente alla pagina di Spotify per effettuare le modifiche.

- Allo stesso modo, non esiste un endpoint per la cancellazione dell'account Spotify o per la rimozione degli scopes dall'app, per questo si rimanda l'utente alle pagine di Spotify per svolgere queste azioni.

- Durante la creazione di una playlist, alcune informazioni possono essere in ritardo, soprattutto l'upload delle immagini degli album. In questo caso, spesso è preferibile gestire le modifiche in locale, inviando le modifiche a Spotify e modificando/creando la playlist direttamente in locale in caso di risposta positiva da parte delle API.

- Ogni richiesta "GET" di playlist può avere al massimo 50 playlist come risposta. In caso di molte playlist, una buona soluzione implementativa potrebbe essere quella di effettuare richieste con una funzione iterativa con un offset per richiedere l'intera lista di playlist, anche se questo approccio può essere rischioso per un grande set di playlist utilizzando le API gratuite a causa dei limiti di rating che potrebbero causare l'errore "429" (troppe richieste).

Implementazione

In primis, quando la pagina si apre viene rimosso "createdPlaylist" dal localStorage, che verrà discusso successivamente e viene impostato lo state accessToken con il valore di accessToken nel localStorage.

Viene subito eseguito un "useEffect", che verrà rieseguito ogni volta che lo state accessToken cambia o quando la finestra di modifica delle informazioni dell'utente viene chiusa. Ciò avviene perché, se l'accessToken scadesse prima di inviare le principali richieste della pagina verrebbe refreshato e cambiato. Noi vorremmo poi che le richieste vengano rieseguite. Inoltre, si suppone che le informazioni dell'utente possano essere cambiate alla chiusura della finestra di modifica, quindi vengono richieste nuovamente; questo avviene poiché non facendo una chiamata alle api ma reindirizzando l'utente a Spotify non possiamo effettivamente sapere se ci sono state delle modifiche.

Le funzioni eseguite dal "useEffect" sono "getInfoUtente" e "getAllPlaylist".

La prima richiede alle API le informazioni dell'utente tramite la chiamata "spotifyApi.getMe()", che vengono salvate in un nuovo oggetto chiamato "currentUser" e inserite anche nel localStorage e renderizzate.

la seconda invece è una funzione iterativa che esegue una serie di chiamate per recuperare a blocchi di 50 (l'offset viene incrementato di 50 ad ogni chiamata) ogni playlist. Ogni richiesta crea un nuovo array di playlist con le informazioni necessarie e lo accoda all'array di playlist precedente. Quando il valore "next" della risposta ad una richiesta è uguale a null, significa che è l'ultimo blocco e il ciclo si interrompe, impostando l'array nello stato "playlistsResults". Inoltre, lo stato "update" (booleano) viene cambiato con la sua negazione.

Ogni volta che "update" cambia, viene eseguito un altro "useEffect", che viene eseguito anche ogni volta che cambia il valore dello stato "filterName". Questo effect calcola, in base a "filterName", un nuovo array di playlist chiamato "playlistFiltered". Questo array è uguale a "playlistsResult" quando "filterName" è uguale a "ALL". Negli altri casi, "playlistFiltered" assume il valore di un sottoinsieme delle playlist di "playlistResult". In particolare:

- playlistFiltered filtra tutte le playlist pubbliche se filterName è “PUBLIC”. Una playlist è pubblica se l’attributo public della playlist è true e l’utente corrente è l’utente creatore della playlist;
- playlistFiltered filtra tutte le playlist private se filterName è “PRIVATE”. Una playlist è privata se l’attributo public della playlist è false.
- playlistFiltered filtra tutte le playlist salvate se filterName è “FOLLOWED”. Una playlist è salvata se l’attributo public della playlist è true e l’utente corrente non è l’utente creatore della playlist.

Le playlist vengono quindi rifiltrate ogni volta che cambia filterName ovvero ogni volta che si clicca sul pulsante di una categoria; inoltre vengono rifiltrate ad ogni nuovo set di iterazioni che richiede nuove playlist.

Quando playlistResult cambia inoltre vengono filtrate tutte le playlist che hanno come creatore e come utente corrente lo stesso id (ovvero le playlist modificabili), e inserite nel local storage, questo permetterà di riprenderle da altre pagine senza chiederle nuovamente.

Le playlist che vengono renderizzate sono quelle inserite nell’array playlistsFiltered ogni volta che esso cambia. Ciclando su playlistsFiltered si passa ogni oggetto playlist come prop ad un componente CardPlaylistPersonalArea.

Per la cancellazione di una playlist si passa come prop a CardPlaylistPersonalArea anche una funzione, tale funzione verrà usata all’interno di CardPlaylistPersonalArea in seguito ad una chiamata per l’unfollow della playlist e permetterà di rimuovere la playlist da playlistResult impostandola semplicemente a null, quando questo accade le playlist vengono rifiltrate e ri-renderizzate, se una playlist è null non viene renderizzata.

Ci sono anche altre due funzioni che permettono di modificare "playlistResult", come la modifica della proprietà "public" di una playlist o il cambiamento di un'intera playlist. Anche queste due funzioni vengono passate a "CardPlaylistPersonalArea" e verranno utilizzate a seguito delle chiamate di modifica di una playlist alle API.

Per quanto riguarda la ricerca, ogni volta che searchWord cambia, uno useEffect viene eseguito, lo useEffect verifica che searchWord non sia una stringa vuota, se lo è, lo state searchResult viene impostato a null, quando è null non viene mai renderizzato.

Se invece non è una stringa vuota viene ciclato e filtrato ulteriormente l’array playlistFiltered, vengono estratti dall’array solamente gli oggetti playlist per cui il campo name contiene la sottostringa searchWord. L’array estratto viene impostato in searchResult cosi da poter essere renderizzato. Nella pagina viene quindi renderizzato

l'array searchResult ciclando su ogni playlist e passando ognuna ad un componente PlaylistCardPersonalArea.

Da Notare che nelle funzioni di modifica e cancellazione delle playlist negli array locali, viene eseguita anche la modifica delle playlist su searchResult (quando esiste) oltre che su playlistFiltered; questo poiché le modifiche devo essere visibili subito anche nel risultato di ricerca.

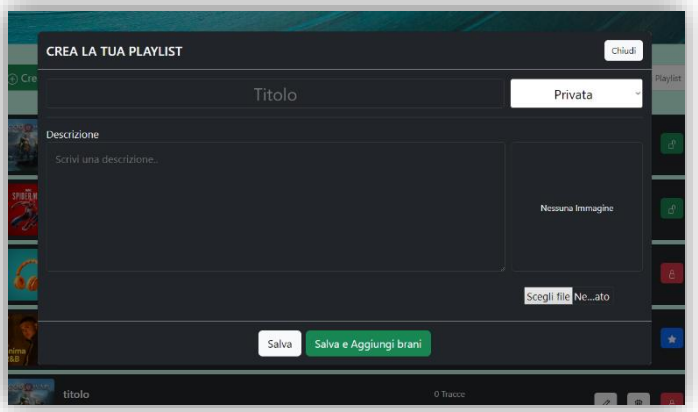
Infine ci sono 2 pulsanti nella pagina che permettono di impostare 2 states su true, quando questi states sono su true vengono renderizzate delle modali ovvero ModalCreatePlaylist e ModalModifyUser. Ad esse vengono passati i setter degli state per impostare il valore a false. Ogni modale nel progetto è stata gestita in questo modo.

Alla modale ModalCreatePlaylist viene inoltre passata una funzione per poter aggiungere una nuova playlist all'inizio dell'array playlistResults.

MODAL CREATE PLAYLIST

Modal Create Playlist è il componente modale che viene renderizzato quando si clicca sul pulsante Crea nella personalArea; questa modale contiene una serie di box da compilare che permettono di definire un titolo e una descrizione, inoltre permette di aggiungere un'immagine ad una playlist e scegliere lo stato di visibilità. Se si cambia idea, il pulsante chiudi, chiude la modale senza nessun effetto ulteriore.

Siccome non esiste un campo apposito per definire le categorie di una playlist i tag possono essere inseriti direttamente nella descrizione di essa.



Implementazione

La modale ha diversi states con dei valori default; null per image, "" per title e description, e infine false per isPublic.

Ogni select, input, text area etc.. inseriti nella modale, al verificarsi di un cambiamento del value impostano il valore corrente nello state corrispondente.

L'input dell'immagine tuttavia verifica prima che l'immagine sia quadrata, se non lo è restituisce un warning e non imposta l'immagine.

La chiamata vera e propria per creare una nuova playlist viene fatta solo quando si clicca sul pulsante salva oppure sul pulsante salva e aggiungi brani.

Entrambi i pulsanti eseguono la stessa funzione, tale funzione se il titolo della playlist non è vuoto invia titolo, descrizione e visibilità della nuova playlist con una chiamata a Spotify; la risposta di Spotify sarà la nuova playlist creata.

A questo punto viene eseguita un'altra funzione che verifica che il campo immagine non sia null, se lo è si può proseguire, altrimenti significa che l'utente voleva impostare anche un'immagine, in questo caso allora invio una chiamata a spotify con l'immagine convertita in base64 e l'id della nuova playlist; se torna una risposta positiva proseguo poiché l'immagine è stata ricevuta correttamente.

Il primo pulsante "Salva" esegue quello che abbiamo descritto, e arrivati a questo punto esegue la funzione addPlaylist che gli è stata passata dalla personalArea per aggiungere la playlist appena creata a playlistResult, dopodichè chiude la modale.

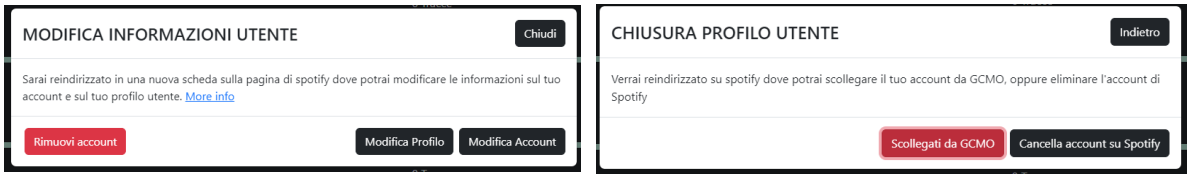
Il pulsante "Salva e Aggiungi brani" invece chiama la stessa funzione, tuttavia inserisce un parametro, se dopo le operazioni descritte il parametro viene trovato significa che è stato cliccato "Salva e Aggiungi brani" e si vuole quindi continuare con l'inserimento di brani. Invece di eseguire addPlaylist e chiudere la modale si aggiunge quindi la playlist nel local storage sotto la voce createdPlaylist, dopodichè si viene reindirizzati alla navigationPage dove sarà possibile fare ricerche e aggiungere canzoni alla playlist creata. La playlist viene salvata nel local storage per poterla recuperare direttamente dalla navigationPage e visualizzarla in un footer come playlist selezionata.

Una precisazione deve essere fatta per quanto riguarda l'immagine salvata in una playlist: se l'immagine è appena stata inviata, probabilmente richiedendo subito la get della playlist non verrà restituita poiché spotify non l'ha ancora caricata nel server; per questo motivo quando si crea una nuova playlist viene creato un URL blob dell'immagine da aggiungere alla playlist in locale in modo da usare la copia locale dell'immagine e visualizzarla subito. Quando però si inserisce la playlist nel local storage per passare alla navigation page l'URL blob scade; per questo motivo al posto dell'immagine in questo caso viene scritto un valore speciale ovvero "ASK", quando la playlist verrà estratta dal local storage nel componente FooterElement potrà avere cosi immagine = null oppure immagine = ASK, se immagine = ASK significa che dovrebbe esserci ma non c'è e quindi vengono inviate richieste ricorsive alle API finchè non viene restituita.

MODAL MODIFY USER

Questa modale è molto semplice, presenta un testo centrale con diversi pulsanti, un pulsante per chiuderla nell'header, e 3 pulsanti nel footer.

La modale viene renderizzata in 2 versioni; quando viene aperta vengono renderizzati 2 pulsanti che rimandano alle modifiche di account e profilo utente su spotify, un terzo pulsante permette di cambiare la versione renderizzata (il pulsante Rimuovi account), quando si clicca su di esso il pulsante rimuovi account sparisce, il pulsante chiudi viene sostituito con il pulsante indietro, e gli altri 2 pulsanti vengono sostituiti con quelli che rimandano ai link di chiusura dell'account o di revoca degli scopes.



PLAYLIST CARD PERSONAL AREA

Questo componente è modellato su una card orizzontale in cui vengono mostrate le informazioni primarie di una playlist quali titolo, creatore, immagine, numero di tracce, e visibilità.

L'intera card è un Button che è possibile cliccare per aprire la modale che mostra le informazioni dettagliate della playlist.

Inoltre ci sono altri buttons sulla card; se la card è una playlist salvata ci sarà solo il pulsante che ce lo indica con la stellina blu; se si clicca sul pulsante la playlist viene rimossa.

Se la playlist appartiene all'utente invece può essere sia eliminata (pulsante con cestino), sia modificata (pulsante con matita). Inoltre troviamo un pulsante di modifica della visibilità in modo rapido, il pulsante è un lucchetto rosso chiuso se la playlist è privata e un lucchetto verde aperto se è pubblica, cliccando su di esso la playlist viene switchata.

Implementazione

Innanzitutto viene eseguita una funzione useEffect che controlla il tipo della playlist ogni volta che questa cambia, ovvero se la playlist passata come prop al componente è una playlist pubblica, privata o salvata. Se l'utente creatore è diverso dall'utente corrente è sicuramente salvata, altrimenti si guarda l'attributo public per capire se è pubblica o privata. Una volta capito questo viene impostato il valore corrispondente al tipo di playlist nello state type.

Ogni volta che la playlist cambia viene controllata anche l'immagine; se c'è qualcosa nel campo immagine della playlist lo state image viene impostato con quel valore, altrimenti se l'immagine è null lo state viene impostato con un'immagine di default del progetto.

Il componente presenta poi una funzione per switchare il tipo di playlist da public a private, questa viene eseguita quando si clicca sul pulsante del lucchetto; per switchare viene controllato lo state type attuale e inviata la richiesta alle API per modificare la visibilità, se la richiesta va a buon fine viene eseguita la funzione passata dal padre del componente (quindi dalla personalArea) per modificare la visibilità di una playlist nell'elenco playlistResults, infine viene invertito lo state type.

Infine c'è una funzione che viene eseguita ogni volta che cambia lo state della PlaylistViewModal. Quando quest'ultimo cambia in false e lo state updatePlaylist è true allora viene eseguita una chiamata che richiede a spotify di nuovo la stessa playlist. Questo avviene poiché updatePlaylist è uno state che viene impostato a true nella modale se vengono rimosse delle tracce dalla playlist, se update è true e lo state della modale diventa false significa che la modale è stata chiusa e siccome aclune tracce sono state rimosse potrebbe essere cambiata l'immagine di copertina e anche il numero di tracce interno, per questo il codice richiede di nuovo la playlist a spotify, dopodichè usa la funzione passata dalla personalArea per sostituire il nuovo risultato a quello precedente relativo alla playlist in questione nell'array playlistResult.

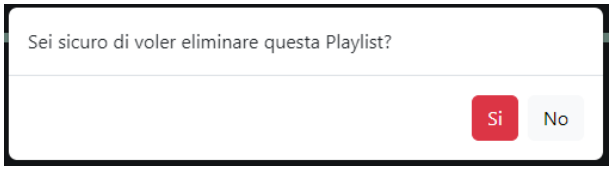
Alla modale di visualizzazione PlaylistViewModal viene quindi passata una funzione per modificare lo state updatePlaylist.

Alla modale di modifica ModalModifyPlaylist viene passata invece direttamente la funzione di modifica di una playlist proveniente a sua volta dalla personalArea.

Alla modale di eliminazione ModalDeletePlaylist viene invece passata la funzione proveniente da personalArea per rimuovere una playlist da playlistResult.

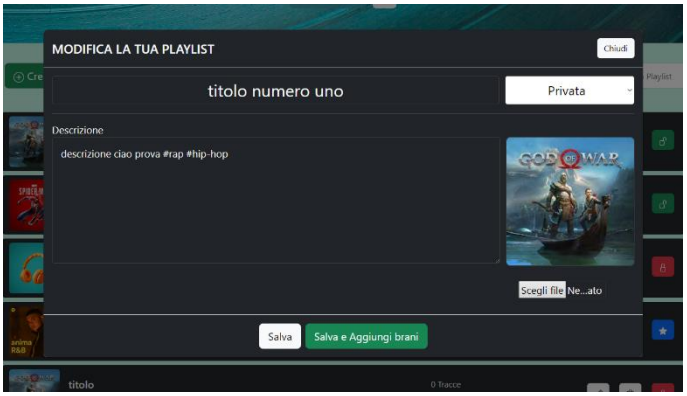
MODAL DELETE PLAYLIST

Questa modale viene aperta quando si clicca sulla stellina di una playlist salvata, oppure sul cestino di un'altra playlist, la modale ha un pulsante che avvia una funzione, la funzione fa una chiamata a spotify per fare unfollow della playlist, se va a buon fine la playlist viene rimossa da playlistResult con la funzione setRemovedPlaylist che gli è stata passata come prop, dopodichè la modale viene chiusa. L'altro pulsante invece chiude la modale senza fare nessuna azione.



MODAL MODIFY PLAYLIST

Questa modale è molto simile alla ModalCreatePlaylist presente nella personalArea; il rendering è lo stesso e le funzioni sono molto simili; la differenza sta nel fatto che all'apertura della modale le informazioni della playlist relativa alla modale devono essere visualizzate nella modale stessa, mentre la modale per la creazione inizialmente all'apertura appare vuota.



Ciò significa che gli states di questa modale vengono inizialmente impostati sui valori della playlist passata come prop.

In particolare l'immagine viene gestita con uno state chiamato image, se nella playlist c'è un attributo image è sicuramente un url quindi settiamo un oggetto image.url uguale a quello della playlist. Se l'immagine nella playlist non è presente image.url sarà null.

Se viene inserito un file immagine nuovo nell'input allora viene aggiunto il file direttamente all'oggetto image.immagine e image.url viene cambiato con il blob dell'immagine aggiunta.

Il resto è molto simile alla ModalCreatePlaylist ma è necessario precisare che:

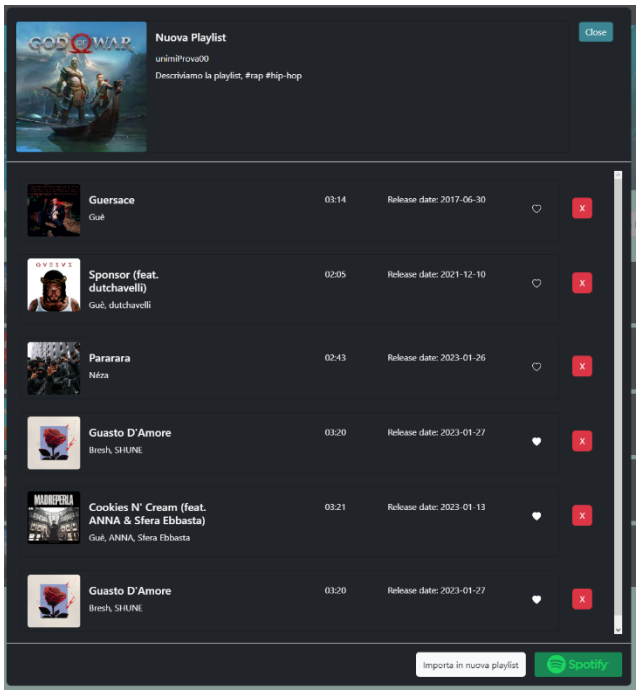
a seguito della richiesta di modifica a Spotify e della funzione di aggiunta immagine, inizialmente, viene sempre impostato un oggetto playlist nuovo con attributo image uguale a image.url. Image sarà quindi null se la playlist non aveva immagine e non è stata aggiunta, l'URL sarà uguale a iniziale se c'era un'immagine ma non è stata cambiata, infine sarà un nuovo URL blob se un'immagine nuova è stata aggiunta.

Questo va bene finchè si preme sul pulsante salva, tuttavia se è stato impostato un URL blob, se si passa alla navigationPage non va bene, quindi, in questo caso se l'URL dell'immagine nella nuova playlist è diverso da null e diverso dall'URL iniziale viene impostato il valore speciale ASK che permetterà di richiedere in futuro l'immagine.

Inoltre in questo caso viene impostato l'URL iniziale in un nuovo attributo della playlist chiamato oldImage che permetterà di confrontare le risposte di spotify per capire quando l'immagine restituita è quella vecchia e quando è quella nuova.

PLAYLIST VIEW MODAL

Questa modale permette di visualizzare le informazioni principali relative alla playlist passata come input. La modale permette di visualizzare il titolo, la descrizione, il nome dell'autore (che può a sua volta essere cliccato per aprire la UserViewModal), l'URL di spotify, l'immagine di copertina e l'elenco delle tracce.



Inoltre sono presenti, un pulsante per chiudere la modale e un pulsante per importare (copiare) la playlist in una nuova playlist.

Ogni traccia sarà cliccabile e salvabile, e aprirà un'altra modale. Se la playlist appartiene all'utente inoltre ogni traccia può essere rimossa dalla playlist.

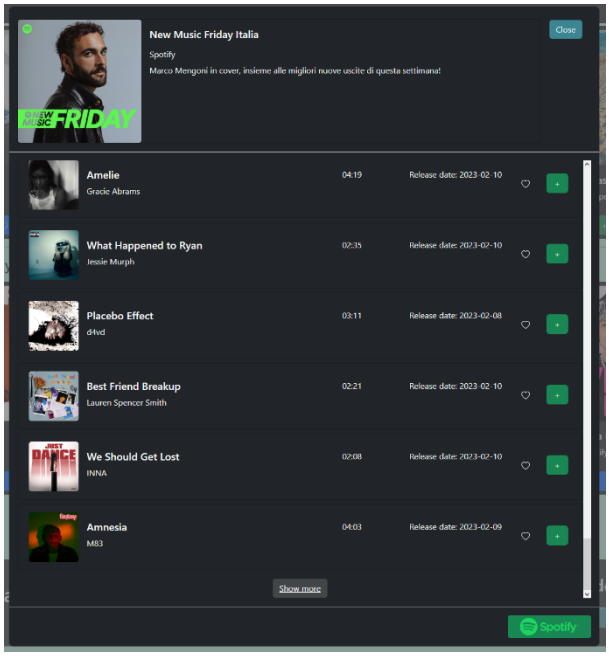
Il limite delle API è che ogni chiamata per recuperare le tracce di una playlist restituisce massimo 50 playlist, ma; per fare l'import ho bisogno dell'elenco intero di playlist. Questo viene gestito renderizzando il pulsante di import solo quando tutte le canzoni sono state richieste; allo stesso modo quando tutte le canzoni sono state richieste il pulsante show more infondo alla lista di tracce non viene più renderizzato.

Implementazione

La playlist con le sue informazioni viene passata come prop in input al componente e le sue informazioni vengono quindi renderizzate.

La prima volta e ogni volta che cambia lo state accessToken viene rieseguita una funzione getAllTracks; questa funzione fa una chiamata a spotify per recuperare le prime 50 tracce della playlist con offset (state) uguale a zero. Le tracce vengono estratte e inserite nell'array (state) tracks; se non ci sono altre tracce da richiedere il campo next della risposta sarà null, quando è null l'offset viene riportato a zero, se non è null l'offset viene invece aggiornato (+=50). Ogni traccia dell'array viene passata attraverso una .map al rendering in un componente TrackCardHorizontal.

Infondo all'elenco di tracce compare un pulsante show more solamente quando offset è diverso da zero, infatti dopo la chiamata alle prime 50 tracce se next esiste l'offset viene aggiornato e quindi è diverso da zero e quindi viene renderizzato il pulsante. Se next non esiste offset rimane zero, non ci sono altre tracce da chiedere e quindi show more non viene renderizzato. Inoltre quando si arriva alle ultime 50 playlist next è null e offset viene riportato a zero e non viene più renderizzato show more.



Il pulsante show more ciò che fa è eseguire di nuovo la funzione getAllTracks recuperando cosi le successive 50 tracce.

Remove track è una funzione che viene eseguita al click del pulsante per eliminare una traccia. Il pulsante viene renderizzato per ogni traccia solo se la playlist appartiene all'utente, ovvero se il creatore è l'utente attuale.

Alla funzione viene passato l'oggetto di una specifica traccia, quindi si occupa di inviare una chiamata a spotify per rimuoverla dalla playlist, se la chiamata va a buon fine viene rimossa la traccia anche dall'array tracks in locale impostando l'oggetto all'index corrispondente a null, se una traccia è null non viene renderizzata. Inoltre viene usata la funzione passata come prop dal padre per cambiare lo stato di updatePlaylist a true, questo permetterà una volta chiusa la modale di aggiornare i valori che potrebbero essere cambiati nella playlist con la rimozione di alcune tracce.

Il pulsante Importa in una nuova playlist esegue la funzione importaPlaylist(), questa funzione si occupa di fare una chiamata per creare una playlist con nome e descrizione uguali a quelli della playlist di cui è aperta la modale. Dopodiché se va a buon fine la chiamata esegue un ciclo for di chiamate per aggiungere tutte le tracce contenute nell'array tracks alla nuova playlist. Ogni chiamata può aggiungere 100 canzoni, quindi quando le canzoni sono meno di 100 viene fatta una sola chiamata, quando sono più di 100 di tracks viene fatto lo slice in più parti, ogni chiamata invierà 100 canzoni tranne l'ultima che potrà inviarne anche meno di 100.

Alla fine la funzione fa una get della nuova playlist creata;

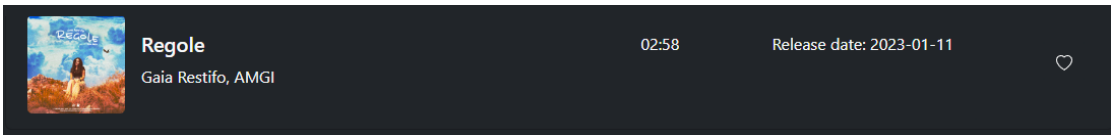
Se la modale è aperta nella personalArea, gli sarà stata passata la funzione prop che permette di aggiungere la playlist in cima alla lista di playlist nella lista locale playlistResult, quindi viene eseguita.

Se invece non c'è la prop, la modale significa che è aperta nella navigationPage, la nuova playlist viene allora aggiunta in cima all'elenco di playlists presenti nel local storage sotto la voce playlist_list di cui parleremo in seguito.

Se qualcosa nella funzione non è andato a buon fine, la playlist viene rimossa da spotify con una chiamata unfollow alle API.

TRACK CARD HORIZONTAL

La card è molto semplice e simile a quella di una playlist, presenta le informazioni più importanti relative ad una traccia come ad esempio titolo, artisti, immagine, release date, e durata. È possibile cliccare sulla card per aprire una modale relativa alla traccia, inoltre è possibile cliccare sul cuore per salvare la traccia.



Implementazione

Ogni volta che la traccia in input di una card cambia allora viene eseguito uno useEffect, l'effect controlla con una chiamata a spotify se la traccia è seguita oppure no dall'utente corrente. Il risultato è booleano e viene inserito con setType in type.

Premendo sul cuore viene eseguita la funzione switchFollow, se type è true viene eseguita una chiamata di unfollow di quella traccia, dopodichè type viene impostato a false. Al contrario se è false viene eseguita la chiamata follow della traccia e type viene impostato a true.

Quando type è true viene renderizzato il cuore pieno, altrimenti vuoto.

Ogni card può avere anche un pulsante che permette di aggiungerla ad una specifica playlist, la playlist selezionata in cui aggiungere le tracce è quella presente nel local storage sotto la voce di createdPlaylist, quindi questo può avvenire solamente se nel local storage esiste createdPlaylist, tuttavia ogni volta che entriamo nella personal area createdPlaylist viene rimosso; quindi questo pulsante esiste solo se il componente si trova nella navigationPage e createdPlaylist è presente nello storage.

Il pulsante + esegue una funzione che fa una chiamata a spotify comunicando di aggiungere la traccia all'id della playlist presente nel local storage sotto il nome di createdPlaylist. Quando la risposta è positiva il pulsante per aggiungere quella specifica traccia viene rimosso dal rendering. Il rendering del pulsante viene controllato tramite uno state booleano che quando la canzone viene aggiunta viene impostato su false, inoltre la canzone viene aggiunta all'elenco createdPlaylistTracks nel localStorage.

Viene poi eseguito uno useEffect ogni volta che showFooter cambia, showFooter è un valore booleano che permette di mostrare l'elemento FooterElement nella navigationPage con la createdPlaylist del local storage selezionata; il footer serve per segnalare qual è la playlist corrente a cui fare le aggiunte delle tracce.

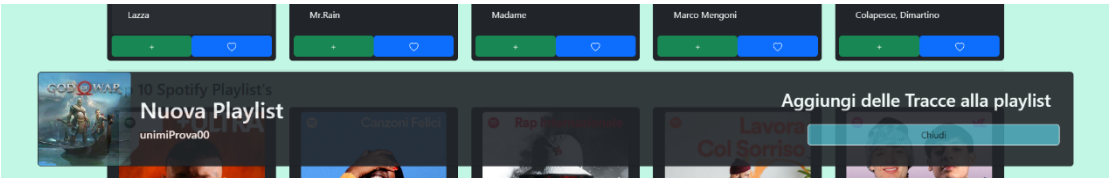
Quando showFooter cambia, se nel local storage è presente la lista createdPlaylistTracks ovvero la lista di tracce della playlist selezionata, allora si controlla che la traccia corrente non sia presente nella lista, se lo è il booleano viene impostato a false e il

pulsante di aggiunta non viene renderizzato. Se invece non è presente il booleano per renderizzare il pulsante viene impostato come showFooter, in questo modo sarà showFooter a gestire quando renderizzare i pulsanti e quando no. Se il footer è attivo allora vengono renderizzati altrimenti no. (Il footer è attivo solamente se nel local storage esiste createdPlaylist).

FOOTER ELEMENT

FooterElement è un componente semplice inserito nella navigationPage. Il componente serve solamente per visualizzare le informazioni di createdPlaylist quando essa è presente nel localStorage. Serve quindi per visualizzare le informazioni della playlist selezionata per l'aggiunta di brani.

Presenta un unico pulsante chiudi che rimuove le informazioni relative alla playlist dal localStorage quando viene premuto.



Implementazione

Interessante è come il componente gestisce l'immagine della playlist.

Siccome la createdPlaylist nel local storage potrebbe essere appena stata creata nella personalArea e quindi non avere ancora un'immagine disponibile, la funzione controlla cosa c'è nell'attributo image della created.

Se all'interno si trova un URL viene restituita l'immagine dell'URL, se si trova null viene restiruita l'immagine di default. Se si trova invece il valore speciale "ASK", viene eseguita una funzione ricorsiva che esegue una serie di chiamate a spotify per recuperare l'immagine della playlist. Quando il valore è ASK nella playlist sarà presente anche un attributo oldImage; se esiste oldImage e la risposta di spotify è diversa da oldImage e non è nulla allora le chiamate ricorsive si interrompono e l'immagine viene renderizzata, altrimenti continuano finchè non viene soddisfatta la condizione.

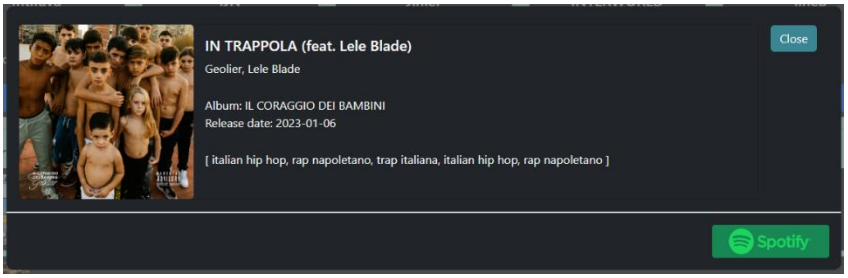
Questo accade perché se spotify non ha ancora aggiornato l'immagine nei server alla chiamata della playlist restituirebbe quella vecchia oppure null. In questo modo invece noi sappiamo che ASK significa che un'immagine è stata sicuramente appena cambiata quindi finchè viene restituito null o la vecchia immagine facciamo chiamate.

Se le chiamate ricorsive generano un'errore viene gestito e la funzione viene riavviata in modo ricorsivo.

TRACK VIEW MODAL

TrackViewModal è una modale molto semplice a cui vengono passate le informazioni relative ad una traccia e vengono quindi renderizzate.

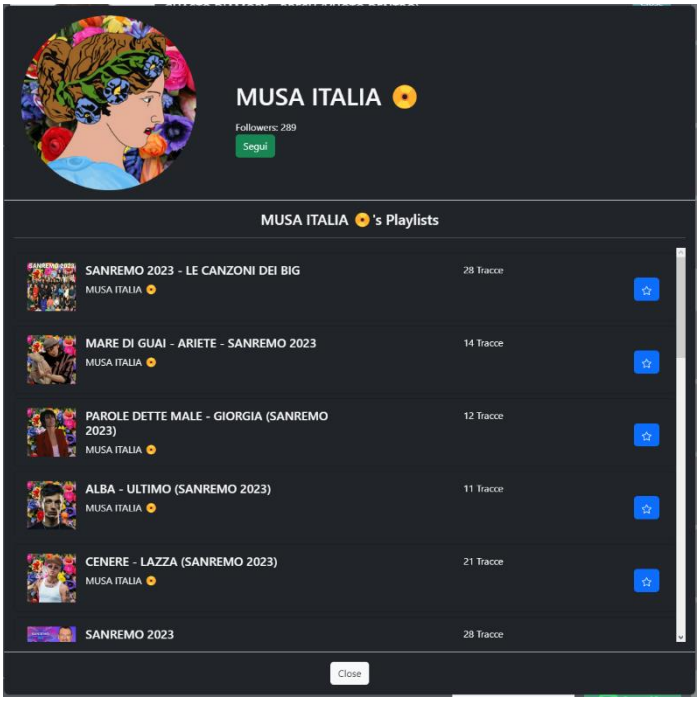
Si può visualizzare il titolo, gli artisti, i generi musicali (recuperati dai generi musicali di un artista), l'album di appartenenza, la release date e un button con il link a spotify.



Il componente effettua delle chiamate per ottenere i generi musicali di tutti gli artisti della traccia, dopodichè accoda tutti i generi in una stringa unica.

USER MODAL VIEW

Questa modale può essere aperta cliccando sul nome dell'autore di una playlist all'interno della PlaylistViewModal.



All'interno della modale troviamo l'elenco delle playlists pubbliche, ogni playlist viene renderizzata grazie al componente PlaylistCardNavigationPage dell'utente e le informazioni relative all'utente come nome, fotografia e followers.

Infine vediamo un pulsante Segui per poter seguire l'utente; il pulsante diviene blu e cambia la scritta se si segue già l'utente.

Ogni playlist è cliccabile e salvabile e apre un'ulteriore modale.

Implementazione

Alla modale viene passato l'id dell'utente, grazie all'id può essere subito eseguita una chiamata alle API per recuperarne le informazioni tramite uno useEffect per poi renderizzarle.

Nello stesso useEffect viene eseguita una chiamata per recuperare le top 50 playlist pubbliche più popolari dell'utente (se ce ne sono). Le playlist vengono salvate in un array e vengono renderizzate una ad una con il componente PlaylistCardNavigationPage che è diverso da PlaylistCardPersonalArea come vedremo in seguito.

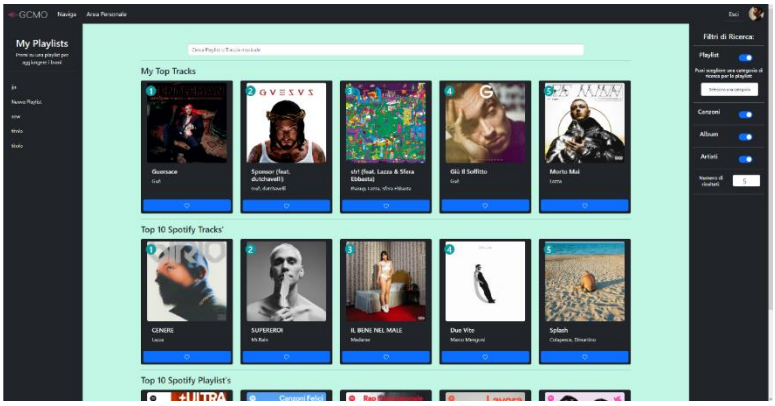
Una volta ottenute le informazioni sull'utente, ogni volta che esse cambiano viene eseguito uno useEffect per effettuare una chiamata a spotify e sapere se l'utente corrente segue l'utente della modale. In base alla risposta viene impostato un attributo followed a true o false nell'oggetto dell'utente, e, in base ad esso viene renderizzato il pulsante segui oppure segui già.

Il pulsante segui esegue la funzione che effettua la chiamata follow, l'altro pulsante fa la chiamata opposta, eseguite le chiamate il valore followed viene invertito.

NAVIGATION PAGE

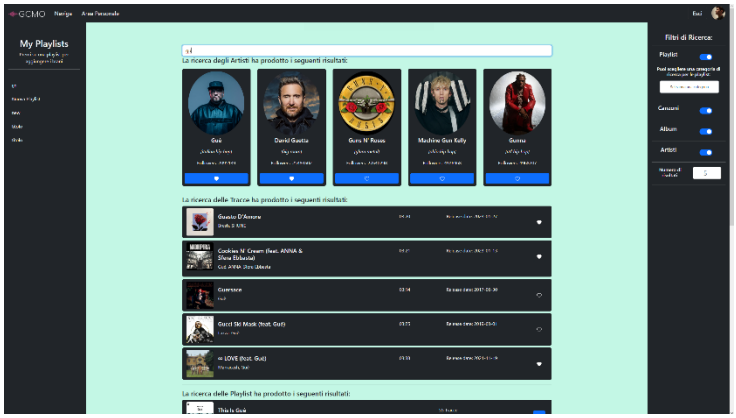
La navigationPage è dedicata alla ricerca; al suo interno è possibile visualizzare l'elenco delle playlist personali create dall'utente loggato, ogni playlist è selezionabile e apre il footer che indica che la playlist è stata selezionata. Quando una playlist è selezionata è possibile visualizzare il pulsante di aggiunta tracce alla playlist su ogni trackCard, comprese quelle interne alle playlist e comprese le tracce interne ad un album.

Il corpo della pagina è composto da 3 caroselli, ognuno dei quali contiene una top 10, il primo contiene la top 10 delle canzoni più ascoltate dall'utente corrente ed è presente solo se l'utente corrente ha ascoltato delle canzoni prima d'ora. Il secondo è una top 10 delle tracce più famose del momento, recuperate dalla playlist top 50 global di spotify oppure top 50 Italia se l'utente è italiano. L'ultimo carosello mostra 10 delle playlist consigliate su spotify. Ogni canzone e playlist è cliccabile e salvabile e apre la modale relativa.



In alto al centro della pagina è presente una barra di ricerca in cui è possibile ricercare canzoni, playlist, album e artisti. A destra della pagina è presente una serie di filtri che permettono di filtrare la ricerca e cambiare il numero di risultati. Inoltre consentono di ricercare playlist per categoria/tag; categoria, genere e tag sono fondamentalmente la stessa cosa per spotify.

Quando si esegue una ricerca i risultati sono sempre card orizzontali, tranne gli artisti che vengono inseriti in un carosello. I risultati di ricerca sostituiscono sempre i caroselli delle top 10.



Il problema più rilevante della pagina è proprio la ricerca, essa poteva essere implementata in diversi modi; tuttavia la ricerca offerta dalle API è una ricerca Fuzzy, questo tipo di ricerca non è una ricerca esatta; cerca sempre di approssimare il risultato migliore che potrebbe fare match con la stringa di ricerca inserita. Per implementare filtri di ricerca più precisi come ad esempio la possibilità di ricercare oggetti scrivendo una categoria o un tag o altro sono stati fatti diversi esperimenti di raffinazione (come ad esempio specificare il market da cui attingere oppure specificare che la parola di ricerca si tratta di una categoria etc..), tuttavia sebbene i tentativi funzionassero restituivano quasi sempre risultati uguali o molto simili a quelli di una semplice chiamata di ricerca alle API. Per questo motivo la ricerca Fuzzy di Spotify API è sembrata la soluzione migliore e più facile da implementare.

Tuttavia, la possibilità di ricercare tramite tag/categorie era richiesta nella consegna e sicuramente interessante da implementare, quindi oltre ad aver predisposto dei filtri sugli oggetti restituiti da una ricerca (album, tracce, playlist, artisti) ho inserito la possibilità di ricercare delle playlists (senza specificare una searchWord nella barra di ricerca) tramite categorie preimpostate da spotify tramite una chiamata più specifica ovvero getPlaylistsForCategory.

Implementazione

Appena aperta la pagina uno useEffect imposta il valore di showFooter a true se nel local storage c'è createdPlaylist, altrimenti lo imposta a false.

Nel codice è poi presente uno useEffect che esegue la funzione di ricerca; questo useEffect viene eseguito ogni volta che cambia la stringa scritta nella form ovvero searchWord, ogni volta che cambia showFooter (per ri renderizzare i pulsanti nel caso si selezioni una playlist), ogni volta che cambia filterArr ovvero l'array che tiene traccia di quali oggetti devono essere ricercati (playlist, canzoni, album, artisti) e ogni volta che cambia searchLimit ovvero il numero di risultati che si vogliono ottenere per ogni categoria di oggetti in ogni ricerca (ex. 5 canzoni, 5 album...etc).

La funzione di ricerca viene chiamata dallo useEffect solamente se la searchWord esiste e non è vuota.

Quando chiamata la funzione di ricerca esegue da 0 a 4 chiamate alle API. filterArr è l'array che si occupa di stabilire quali chiamate devono essere fatte; se la posizione 0 di filterArr è true viene fatta la chiamata inviando la searchWord per cercare le playlists con un limit uguale a searchLimit. 1 per canzoni, 2 album, 3 artisti..

Le risposte vengono inserite negli array corrispondenti (searchPlaylists, searchAlbums etc…); uno per ogni categoria di oggetti. Quando lo useEffect non può eseguire la funzione di ricerca ovvero quando si svuota searchWord imposta tutti questi array a null, se gli array sono null vengono rimossi dal rendering.

Quando gli array non sono null vengono ciclati e viene estratto ogni oggetto.

Ogni oggetto viene passato cosi ai componenti corrispondenti che lo potranno renderizzare. Le tracce vengono passate ognuna a TrackCardHorizontal, gli artisti a Artist, gli album a Album, le playlists a PlaylistCardNavigationPage.

La ricerca di playlists tramite categoria viene fatta tramite uno useEffect che viene eseguito quando cambiano searchLimit o optionCategory. La ricerca viene rieseguita quindi quando cambia searchLimit (il numero di risultati desiderati) e quando cambia optionCategory (ovvero la categoria di playlist selezionata da ricercare). La chiamata in questo useEffect invia l'id della categoria selezionata e si chiama getPlaylistsForCategory e restituisce una serie di playlist di quella categoria che vengono poi renderizzate come PlaylistCardNavigationPage.

Ogni volta che il token cambia (viene refreshato) o che la searchWord cambia viene eseguito un altro useEffect; se la searchWord è vuota lo useEffect esegue le funzioni necessarie per recuperare le top 10.

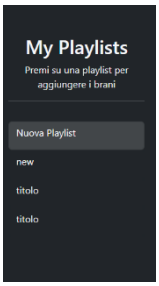
La funzione getTop fa 2 chiamate, una per recuperare le top 10 playlists consigliate da spotify (getFeaturedPlaylists), l'altra chiamata invece invia una richiesta per ottenere 10 tracce dalla playlist top 50 global o top 50 Italia in base alla nazionalità dell'utente.

Entrambe le chiamate hanno limit 10; i risultati verranno inseriti nei rispettivi array ed uno ad uno saranno renderizzati nei componenti corrispondenti ovvero PlaylistCardVertical e TrackCardVertical. Ogni gruppo di 5 oggetti viene inserito in una pagina di un carosello. Ogni carosello è composto da 2 pagine (limit 10).

L'altra funzione che viene eseguita è getTopTracksFunction che fa una chiamata per richiedere le 10 tracce più ascoltate da un utente (getMyTopTracks). I risultati vengono inseriti in un array, poi in 2 pagine di un carosello e poi ogni traccia in un TrackCardVertical.

PLAYLIST LIST

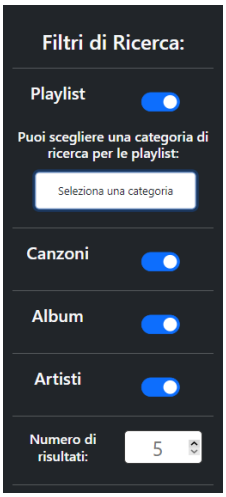
Questo componente recupera dal localStorage la lista di playlists e la inserisce in uno state, se nel localStorage è presente anche createdPlaylist e non è già presente nella lista viene estratta e aggiunta in cima allo state lista, il componente renderizza poi tutte le liste come pulsanti, il pulsante di ogni playlist inserisce la playlist nel local storage sotto la voce createdPlaylist, in questo modo compàrirà il footer e i pulsanti per aggiungere le canzoni.



Ogni volta che si seleziona una playlist inoltre viene fatta anche una chiamata per recuperarne le tracce e aggiungerle a createdPlaylistTracks nel local storage.

FILTRI DI RICERCA

Questo componente è responsabile per la gestione dei filtri di ricerca utilizzati nella navigationPage. Riceve input tramite setter per gli stati che rappresentano i filtri selezionati nella navigationPage, permettendogli di modificare i filtri attualmente attivi. I filtri riguardanti gli oggetti di ricerca sono raccolti in un array booleano, dove ogni posizione rappresenta una categoria di oggetti come playlist, artisti, etc. Se la posizione corrispondente a una categoria viene impostata a "true", gli oggetti di quella categoria verranno inclusi nella ricerca, altrimenti verranno ignorati. Internamente al componente l'array dei filtri viene gestito con degli switch che al cambiamento invertono il booleano alla posizione corrispondente nell'array dei filtri.



Inoltre, ci sono i filtri di categoria per le playlist. Quando uno di questi filtri viene attivato, la barra di ricerca viene disabilitata nella navigationPage, e all'interno del componente, tutti gli switch vengono disabilitati ad eccezione di quello per le playlist. Quando la categoria selezionata torna quella nulla vengono riattivati gli switch.

Le categorie delle playlist sono ottenute tramite una richiesta alle API durante il primo rendering del componente e vengono inserite in una select attraverso un ciclo. La select è in grado di modificare il valore della categoria selezionata tramite un setter passato dalla navigationPage.

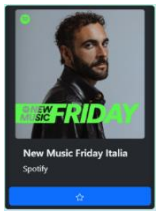
Se la posizione o nell'array dei filtri è false e quindi la ricerca delle playlist non è attiva la select delle categorie per le playlist non viene renderizzata.

Infine, c'è un input per inserire il limite dei risultati di ricerca. Anche questo, in caso di modifica, utilizza un setter passato dal componente padre per modificare il limite. Il valore predefinito è impostato a 5 e può raggiungere un massimo di 50.

TRACK CARD VERTICAL

Il funzionamento di questo componente è uguale a TrackCardHorizontal, la differenza è solamente una differenza di design.

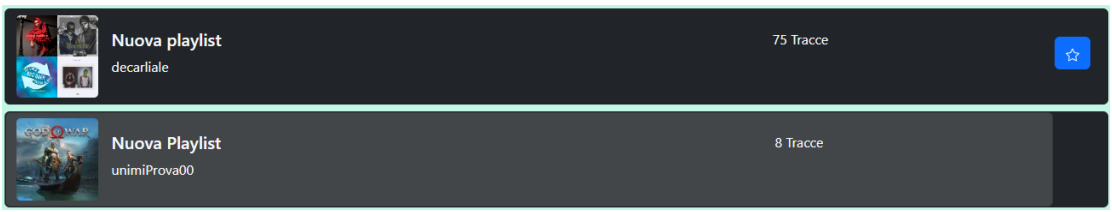
PLAYLIST CARD VERTICAL



Il funzionamento di questo componente è uguale a PlaylistCardNavigationPage, la differenza è solamente una differenza di design.

PLAYLIST CARD NAVIGATION PAGE

Questo componente è molto simile a PlaylistCardPersonalArea, tuttavia presenta alcune importanti differenze. Nonostante abbia lo stesso design, viene utilizzato solo dove non è necessario effettuare modifiche alle playlist. Infatti, le modifiche possono essere effettuate solo nella sezione personalArea. Inoltre, ogni playlist passata a questo componente è sempre il risultato di una ricerca pubblica e di conseguenza, ogni playlist sarà pubblica. Pertanto, l'unica distinzione calcolata è tra le playlist create dall'utente e quelle create da altri utenti. Se una playlist non è stata creata dall'utente corrente, verrà visualizzato un simbolo di stella piena o vuota. Se invece l'utente corrente è il creatore della playlist, non verrà visualizzato alcun pulsante aggiuntivo oltre a quello che apre la modale. Non ci sono pulsanti per la modifica o l'eliminazione della playlist. Per modificare o eliminare una playlist di cui si è il creatore, bisogna sempre accedere alla sezione personalArea.



Implementazione

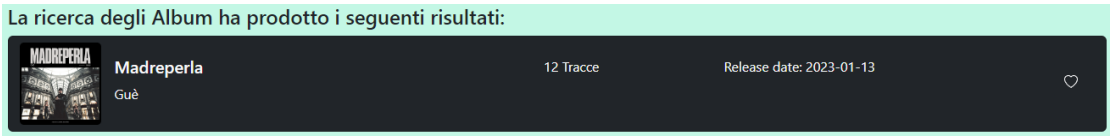
Nel componente ci sono fondamentalmente 2 funzioni, una viene eseguita subito e verifica se l'utente creatore e l'utente corrente sono lo stesso utente. Se non è così viene verificato con una chiamata alle API se l'utente corrente segue o no la playlist; infine viene classificata la playlist con uno state che può essere: MINE, FOLLOWED, NOTFOLLOWED.

Se lo state è mine non viene renderizzato nessun pulsante aggiuntivo; se lo state è followed viene renderizzato il pulsante con la stella piena, altrimenti vuota.

La seconda funzione viene invece eseguita se si clicca sul pulsante renderizzato, la funzione verifica se lo state è followed o notfollowed, nel primo caso invia una chiamata di unfollow a spotify, nell'altro invia la chiamata opposta, dopodichè cambia lo state con il suo opposto.

ALBUM

Il componente Album è molto simile a TrackCardHorizontal, tuttavia gli viene passato un'album invece che una traccia, il componente permette di visualizzare le informazioni principali di un album quali: Titolo, immagine di copertina, release date, numero di tracce e artista.



L'album può essere salvato tra le preferenze con il simbolo del cuore proprio come avviene per le Tracce. Cliccando invece sulla card viene aperta la modale AlbumViewModal.

Implementazione

All'interno del componente ci sono sostanzialmente 2 funzioni simili a quelle presenti nei componenti TrackCardVertical e TrackCardHorizontal;

Ogni volta che l'album cambia viene sostanzialmente rieseguito uno useEffect che verifica con una chiamata alle API se l'album è uno degli album salvati tra le preferenze dell'utente. Nello state type viene cosi impostato lo stesso valore booleano della risposta alla chiamata.

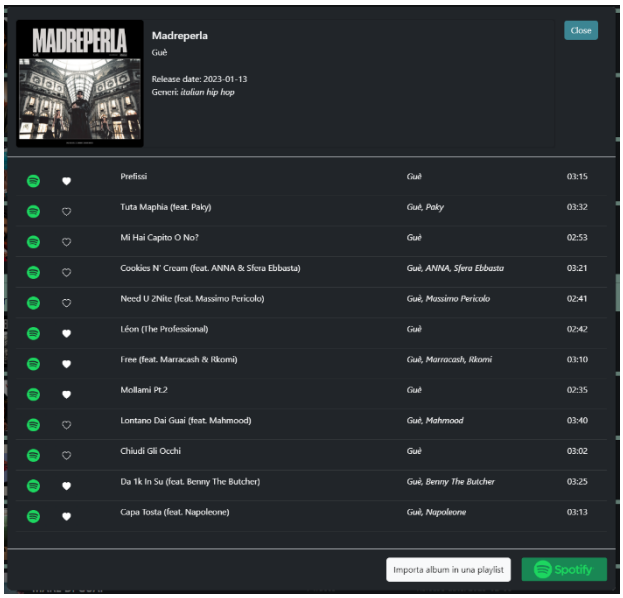
Il pulsante con il cuore quando cliccato permette di eseguire la seconda funzione che verifica se il type è true o false; se è true invia una chiamata removeFromMySavedAlbums, e alla risposta imposta type a false, se è false invece invia una chiamata addToMySavedAlbums per seguire l'album e imposta type a true.

Quando type è true il pulsante rappresenta un cuore pieno, quando è false un cuore vuoto.

Un'ulteriore useState tiene traccia di quando la modale dovrà essere renderizzata, alla modale verrà passata una funzione per cambiare lo state e rimuoverla dal rendering.

ALBUM VIEW MODAL

Il componente AlbumViewModal serve per visualizzare il contenuto di un album in dettaglio, si può attraverso di esso visualizzare informazioni come: Titolo, artista, genere, release date, immagine di copertina e l'elenco completo delle tracce contenute. Ogni traccia ha un pulsante per salvarla tra le preferenze e un link a spotify e presenta informazioni quali gli artisti che hanno partecipato alla traccia, il titolo e la durata.



Le tracce in questo caso non sono renderizzate tramite un componente specifico come TrackCardHorizontal poiché ognuna delle tracce contenute in un album avrà la stessa copertina e le stesse informazioni principali.

Siccome le tracce non sono renderizzate tramite il componente TrackCardHorizontal la verifica e le funzioni per aggiungere ogni traccia ad una playlist vengono incluse direttamente nel componente AlbumViewModal.

Infine la modale come per le playlist presenta la possibilità di importare l'album in una nuova playlist tramite uno specifico pulsante.

Implementazione

La funzione per recuperare le tracce dell'album è molto simile a quella implementata nella PlaylistViewModal, tuttavia ci sono delle differenze. Innanzitutto la chiamata alle API differisce in quanto in questo caso viene utilizzata getAlbumTracks.

In secondo luogo la funzione getAllTracks è stata disposta come una funzione iterativa poiché è difficile e molto raro che gli album possano presentare più di 50 o 100 tracce al loro interno, per questo motivo verranno eseguite al massimo 1 o 2 iterazioni e non c'è quindi il rischio di inviare troppe richieste alle API con conseguente error status 429. La differenza sta quindi nel fatto che in una playlist abbiamo il pulsante show more e ogni volta che viene cliccato vengono richieste nuove tracce, in un album invece le tracce vengono richieste iterativamente tutte subito all'apertura della modale analogamente al recupero delle playlist nella personalArea.

Per farlo viene tenuta traccia di un offset e ad ogni nuova richiesta si accodano i risultati a quelli precedenti; alla fine ogni elemento dell'array risultante viene renderizzato. Il ciclo si interrompe quando la risposta presenta il campo next nullo. Ad ogni iterazione l'offset viene incrementato di 50; quando next è nullo viene riportato a 0. La funzione viene rieseguita ogni volta che cambia lo state accessToken (viene refreshato).

Infine all'interno del ciclo per ogni traccia viene effettuata una chiamata che verifica se la traccia è seguita o no dall'utente e il valore booleano restituito viene inserito nella traccia come attributo.

Il pulsante a forma di cuore per fare follow e unfollow viene quindi renderizzato in questo caso sul valore booleano dell'attributo followed della traccia.

Cliccando sul pulsante viene eseguita la funzione che verifica l'attributo followed e fa una chiamata di follow o unfollow analogamente ai componenti precedenti; ovviamente in questo caso quando la chiamata va a buon fine viene cambiato direttamente l'attributo followed dell'oggetto per cambiare la grafica del pulsante.

Le tracce in AlbumViewModal vengono renderizzate ciclando sull'array tracks e inserendole in una tabella.

Il componente AlbumViewModal riceve infine la prop "ShowFooter" e può utilizzarla insieme all'array "Tracks" per eseguire uno useEffect, al fine di controllare la presenza dell'elemento "createdPlaylist" nel local storage. Se presente, il componente estrae le tracce della playlist dal local storage e le inserisce nello state "traccePlaylist". Successivamente, il componente itera sull'array "Tracks" utilizzando il metodo ".map()" per creare un array di valori booleani. Ogni elemento booleano corrisponde all'operazione AND tra il valore booleano di "ShowFooter" e la negazione del valore booleano restituito dal metodo ".some()" quando viene eseguito sull'array "traccePlaylist" per verificare se l'id della traccia corrente in ".map()" è uguale a qualsiasi id presente in "traccePlaylist".

Infine, l'array di booleani viene salvato nello state "addBtn", che viene utilizzato per renderizzare il pulsante di aggiunta di ogni traccia solo se "ShowFooter" è true e la traccia non è contenuta in "createdPlaylistTracks", e quindi non è già presente nella playlist.

Al click del pulsante di aggiunta viene eseguita la funzione addTrack che prende in ingresso una traccia e un indice; la funzione fa una chiamata per aggiungere la traccia alla createdPlaylist estratta dal local storage.

Se va a buon fine la chiamata la traccia viene aggiunta a traccePlaylist e a createdPlaylistTracks nel local storage e il valore booleano corrispondente all'indice passato in ingresso viene cambiato in false nell'array addBtn rimuovendo cosi il pulsante dal rendering.

Il pulsante "importa in nuova playlist" esegue una funzione del tutto analoga a quella presente in PlaylistViewModal.

ARTIST

Il componente Artist è una card verticale che permette di visualizzare l'immagine dell'artista, il nome, i generi musicali, i followers e un pulsante per salvare o rimuovere l'artista tra le preferenze dell'utente.

Implementazione

Come per le tracce vengono riprese analogamente 2 funzioni, un'effect che verifica il tipo dell'artista con una chiamata che controlla se l'utente lo segue oppure no, e una funzione che viene eseguita al click del pulsante a forma di cuore per fare il follow o l'unfollow dell'artista.

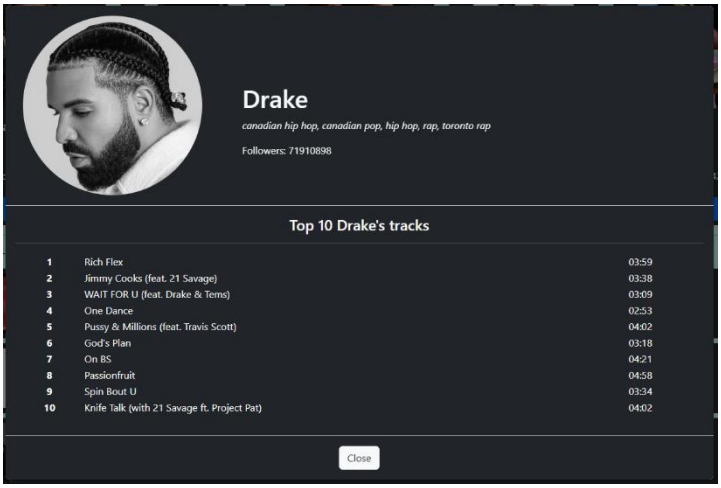
Inoltre al click sulla card lo state della modale cambia e viene renderizzato ArtistViewModal.

ARTIST VIEW MODAL

La modale per visualizzare gli artisti è in sé molto semplice, vengono semplicemente renderizzate informazioni sull'artista quali l'immagine dell'artista, il nome, i generi musicali, e i followers.

Inoltre viene renderizzata una tabella di 10 titoli di tracce dell'artista ordinate per popolarità, e relativa durata in minuti.

Le tracce vengono recuperate con una sola funzione eseguita tramite useEffect ad ogni cambio dello state accessToken; la chiamata è getArtistTopTracks e prende come opzione il paese corrente dell'utente.



NAVIGATION BAR

Il componente NavigationBar sfrutta il componente react-bootstrap NavBar e viene renderizzata tramite una condizione.

Viene verificato se nel local storage è presente l'accessToken, se è cosi significa che l'utente è loggato, quindi viene recuperata l'immagine dell'utente e inserita nella navbar, inoltre viene renderizzato il pulsante Esci che al click esegue la funzione Logout(), essa rimuove tutto ciò che c'è nel local storage tranne il valore "visited".

Se l'accessToken non è presente l'immagine visualizzata per l'utente nella nav è invece un'immagine di default e il pulsante Esci viene sostituito invece con il componente ButtonLogin.

In entrambe le versioni della nav ci saranno 2 button che rimandano agli altri percorsi del sito e il logo GCMO "cliccabile" che riporta alla indexPage.

BUTTON LOGIN

Il componente ButtonLogin viene renderizzato semplicemente con lo stile di un pulsante bootstrap, tuttavia si tratta in realtà di un link a cui viene passata una prop relativa al testo.

Al click del pulsante l'utente viene linkato all'AUTH_URL, ovvero una costante di testo contenente un link che è stato costruito per indirizzare l'utente all'autenticazione con spotify.

Implementazione

L'URL di autenticazione è composto dalle seguenti costanti:

- BASE_URL: rappresenta l'URL di base per l'autenticazione su Spotify.
- RESPONSE_TYPE: indica il tipo di risposta che si vuole ricevere da Spotify dopo aver effettuato la richiesta di autenticazione. In questo caso, il valore è 'code', il che significa che si richiede un codice di autorizzazione da parte di Spotify.
- REDIRECT_URI: indica l'URI a cui verrà reindirizzato l'utente dopo che avrà dato il consenso all'accesso alle sue informazioni personali. In questo caso, l'URI è http://localhost:3000/.
- SPOTIFY_SCOPE: rappresenta le autorizzazioni che si richiedono per accedere alle informazioni dell'utente. In questo caso, vengono richieste una serie di autorizzazioni per la riproduzione di musica, la lettura delle email dell'utente, l'accesso alla sua libreria musicale, la gestione della riproduzione e delle playlist, ecc.

Quindi, l'URL di autenticazione completo è ottenuto concatenando tutte queste costanti in una stringa.

Quando un utente è autenticato viene rimandato quindi alla indexPage e ButtonLogin viene ri-renderizzato.

Se nel local storage è presente l'access token l'utente viene reindirizzato da ButtonLogin alla personalArea o alla setPreferencesPage (in base alla presenza di visited nel local storage.)

Se l'accessToken non è presente invece ButtonLogin prova ad eseguire la funzione useAuth() passandogli il code estratto dall'url della pagina corrente (se c'è), se non c'è significa che ButtonLogin non è ancora stato premuto.

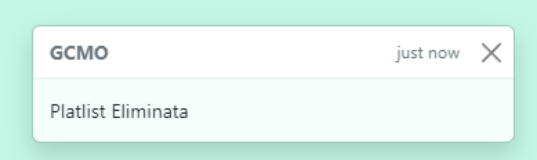
useAuth è la funzione che permette di richiedere al backend lo scambio del code per l'accessToken alle API.

TOAST NOTIFY

Toast notify è un componente che è stato creato per notificare successi ed errori comuni nell'utilizzo del sito web. Il componente prende in ingresso delle props tra cui uno stato show, una funzione onClose e una stringa di testo.

Quando show cambia viene eseguito un'effect; l'effect imposta un timeout di 3 secondi dopodichè esegue la funzione onClose chiudendo il toast.

All'interno del componente troviamo il rendering del componente Toast di react-boostrap.



ToastNotify ha la particolarità di essere inserito direttamente nel componente App, il suo css ha una z-index molto elevata e una posizione assoluta. Questo è stato fatto per garantire che in ogni pagina il toast venga renderizzato al di sopra di qualsiasi modale aperta e sempre nella stessa posizione.

È proprio con ToastNotify che è nata quindi la necessità di utilizzare l'hook useContext, poiché essendo definito nel componente App aveva però bisogno di poter essere modificato (show) all'interno di ogni altra pagina o componente.

APP

Il componente "App" rappresenta il componente principale di un'applicazione React. Questo componente contiene tutti gli altri componenti e definisce la struttura di base dell'applicazione.

Nel componente App sono stati definiti innanzitutto due state: showToast, e toastText, poi è stato definito il contesto dei Toasts tramite lo createContext; al provider di ToastContext è stata passata come value una funzione che prende in input 2 valori e sfrutta i setter di showToast e toastText. Ogni componente interno al contesto è

quindi in grado di modificare la visibilità e il testo di un toast quando ne ha bisogno semplicemente importando ToastContext e richiamandone la funzione tramite useContext.

Nell'app, inoltre, è presente un componente Router importato da react-router-dom. All'interno di esso si trova il componente Routes, che è una collezione di componenti Route. Ogni Route associa un path a un componente pagina, in modo tale che il Router possa gestire la navigazione tra le diverse pagine dell'applicazione in base all'URL richiesto dall'utente renderizzando solo la pagina corrispondente al path.

Gestione degli errori

Gli errori risultanti dalle chiamate alle API di qualsiasi componente vengono sempre gestiti tramite una funzione comune a tutto il progetto. La funzione si chiama ErrorStatusCheck e prende in input l'oggetto restituito dalle chiamate in caso si verifichi un errore e una funzione setter dello state accessToken del componente (se presente).

La funzione viene sempre chiamata all'interno dei blocchi catch e dopo la sua esecuzione, se necessario viene settato un toast che segnali l'errore all'utente.

Prima di tutto ErrorStatusCheck verifica che l'oggetto in ingresso sia effettivamente un oggetto di errore controllando che ci sia un body e un body.error.

Se ci sono allora la funzione può proseguire verificando lo status dell'errore con una serie di condizioni; se lo status è:

- 401: L'access token non è più valido, viene lanciata la funzione refreshToken() per ottenere un nuovo token di accesso.
- 403: L'utente non ha i permessi per accedere al servizio richiesto, viene inserito un alert e lanciata la funzione refreshToken() per ottenere un nuovo token di accesso.
- 429: Sono state inviate troppe richieste a Spotify, viene inserito un alert e impostato un timeout con tempo uguale all'header retry_after dell'errore, in modo da impedire ulteriori richieste per il periodo di tempo specificato nell'header.
- 500/502/503: Si è verificato un errore sul server API di Spotify, viene gestito con un alert uguale per tutti e 3 i casi.
- 400: La richiesta non è valida, viene stampato solo a console.log, poiché questo non dovrebbe mai verificarsi lato utente se l'applicazione è stata progettata correttamente.
- 404: La pagina richiesta non è disponibile, viene stampato solo a console.log, per gli stessi motivi dell'errore 400.

In molti componenti e pagine del sito viene utilizzato un valore di stato chiamato accessToken, che viene impostato con il valore presente nel localStorage. Questo valore viene utilizzato solamente per rilevare il cambiamento del token e non per visualizzare il token stesso. Quando il valore di accessToken è presente, il setter dello stato viene passato alla funzione ErrorStatusCheck.

Se la funzione ErrorStatusCheck rileva un errore 401, passerà il setter dello stato come parametro alla funzione refreshToken. La funzione refreshToken cambierà il valore di accessToken nel localStorage e nell'oggetto spotifyApi e utilizzerà il setter per aggiornare lo stato di accessToken. Il cambiamento dello stato di accessToken viene rilevato da una funzione useEffect nel componente e riavvia le funzioni principali necessarie per la visualizzazione della pagina.

Ad esempio, nella pagina personalArea, il cambio di stato avvia la funzione per ottenere le informazioni dell'utente e l'elenco delle playlist. Questa strategia è stata implementata per mascherare l'errore ed evitare che l'utente debba aggiornare manualmente la pagina per visualizzarne i contenuti a seguito del cambiamento del token.

Funzioni e costanti

SPOTIFY API

Nel progetto è presente un file costanti.js, il file ha al suo interno l'export di un oggetto chiamato spotifyAPI con al suo interno il Client_ID dell'applicazione e inizializzato con un metodo costruttore fornito da spotify-web-api-node. L'oggetto viene condiviso in tutto il progetto e importato in ogni componente e funzione che ha bisogno di fare delle chiamate alle API di spotify.

Tale oggetto viene Sfruttato per astrarre le chiamate alle API, per farlo però ha bisogno di un'attributo accessToken, l'attributo viene impostato la prima volta nella funzione useAuth e viene cambiato all'interno della funzione refreshToken.

USEAUTH

La funzione useAuth viene chiamata dal componente ButtonLogin una volta recuperato il code. Il code viene passato a useAuth che si occuperà di inviare una richiesta http con Axios al backend per scambiare il code con i token. La risposta conterrà l'accessToken, il refreshToken e l'expires_In, tutti e 3 vengono impostati nel localStorage. L'accessToken viene inoltre inserito nell'oggetto SpotifyApi.

REFRESH TOKEN

La funzione refreshToken viene sempre avviata dalla funzione ErrorStatusCheck a seguito di un errore di una richiesta che aveva come status 401. RefreshToken invia una richiesta http con axios al backend con il refreshToken, la risposta conterrà il nuovo accessToken e il nuovo expires_in che vengono aggiornati nel localStorage. L'accessToken inoltre viene aggiornato nell'oggetto spotifyApi, e se presente viene inserito nel setter passato come parametro alla funzione refreshToken.