Relazione progetto SNM

Tecnologie e linguaggi per il web

Anita Papetti

In questo progetto ho sviluppato l'applicazione web "Social Network for Music" (SNM) in JavaScript. Ho deciso di utilizzare come lingua del progetto l'inglese, in quanto comprensibile e usata a livello internazionale, specialmente in ambito informatico.

Installazione

Requisiti: un account Spotify for developers, Node.js, un browser (es. Google Chrome), connessione internet

Si devono inserire nel file .env il client_id e il client_secret corrispondenti al proprio account Spotify for developers. Si può accedere all'account o crearne uno dalla pagina https://developer.spotify.com/ . Inoltre, si deve inserire nel file .env anche il jwt_secret, con cui verrà fatta la codifica del token di autorizzazione di SNM.

L'applicazione può essere utilizzata dopo aver installato i pacchetti necessari:

- Aprire un terminale, recarsi nella cartella del progetto ed eseguire il comando "npm install"

Utilizzo

L'applicazione funziona in locale e può essere utilizzata in due passaggi:

- Aprire un terminale, recarsi nella cartella del progetto ed eseguire il comando "npm start"
- Aprire un browser e recarsi all'url http://localhost:4000/

Struttura del sito web

La parte del sito visitabile senza autenticazione comprende tre pagine:

1) http://localhost:4000/ : indirizzo base, in cui si ha la possibilità di scegliere tra login e signup
2) http://localhost:4000/login : pagina di login (autenticazione utente)
3) http://localhost:4000/signup : pagina di signup (registrazione nuovo utente e autenticazione)

Tutte le altre pagine sono visitabili solo previa autenticazione (login o signup).

A seguito di login effettuato con successo, viene caricata la home page:

- http://localhost:4000/home : homepage dell'utente. Ho scelto di lasciarla vuota. Nell'ottica di un'applicazione reale, ci inserirei delle sezioni "Esplora" , "Consigliati per te" o "Visitati di recente".

A seguito di signup effettuato con successo, all'utente viene chiesto di inserire i generi musicali preferiti e successivamente gli artisti preferiti, dopo di che viene caricata la home page.

- http://localhost:4000/choose-genres/signup : pagina per la scelta dei generi musicali preferiti (versione signup). L'utente appena registrato seleziona i generi musicali preferiti dall'elenco e li salva. In alternativa, può saltare questo passaggio cliccando sul link "Maybe later".

- http://localhost:4000/choose-artists/signup : pagina per la scelta degli artisti musicali preferiti (versione signup). L'utente appena registrato può cercare degli artisti, aggiungerli alla lista dei suoi artisti musicali preferiti e salvare. In alternativa, può saltare questo passaggio cliccando sul link "Maybe later".

Nella home page è presente una barra di navigazione da cui si possono ottenere le varie funzionalità dell'applicazione. La barra di navigazione è composta da: logo SNM (riporta alla home page), "Search" dropdown menu "Playlists" con opzioni "Create new playlist" e "Your playlists", dropdown menu "Preferences" con opzioni "Edit favourite genres" e "Edit favourite artists", "Profile", "Account", "Logout".

Il link "Search" della barra di navigazione porta alla pagina di ricerca:

- http://localhost:4000/search : pagina di ricerca. L'utente può fare ricerche tramite una barra di ricerca. È possibile selezionare il tipo di ricerca, che può essere "Playlist", "Song", "Artist" o "User profile". I risultati vengono mostrati sotto la barra di ricerca, in modo appropriato al tipo.
  - Ricerca di "Playlist": vengono mostrate le anteprime delle playlist. Ogni anteprima contiene titolo, numero di likes, descrizione e il pulsante "View playlist", che porta alla pagina di quella playlist. Si possono scorrere le pagine dei risultati con i pulsanti prev/next.
  - Ricerca di "Song": vengono mostrate le canzoni risultato della ricerca. Le informazioni visualizzate sono titolo, artisti, album, data di pubblicazione e durata. Si possono scorrere le pagine dei risultati con i pulsanti prev/next.
  - Ricerca di "Artist": vengono mostrati gli artisti risultato della ricerca. Le informazioni visualizzate sono la foto dell'artista, il nome e i generi musicali. Si possono scorrere le pagine dei risultati con i pulsanti prev/next.
  - Ricerca di "User profile": viene mostrato l'utente con l'username cercato, con il pulsante "View profile".

Il pulsante "View profile" del risultato di ricerca di "User profile" porta alla pagina:

- http://localhost:4000/profile/:username : pagina del profilo dell'utente con username ":username" (componente URI codificato). Mostra il profilo di quell'utente.
  Nella prima sezione mostra i generi musicali preferiti, nella seconda mostra gli artisti musicali preferiti e nella terza mostra le anteprime delle playlist pubbliche che ha creato. Ogni playlist può essere visualizzata con il pulsante "View playlist".

L'opzione "Create new playlist" del dropdown menu "Playlists" della barra di navigazione porta alla pagina:

- http://localhost:4000/create-playlist : pagina per la creazione di una nuova playlist. Viene chiesto all'utente di inserire titolo, descrizione, tags e visibilità (pubblica o privata) della nuova playlist. Inseriti tutti i campi, l'utente può salvare o abbandonare la creazione. Se ha scelto di salvare, viene indirizzato alla pagina della playlist appena creata.
- http://localhost:4000/playlist/:title : pagina della playlist con titolo ":title" (componente URI codificato). Mostra le informazioni della playlist: titolo, descrizione, tags, visibilità, username dell'autore, numero di likes, pulsante per mettere o togliere il like, durata totale, elenco delle canzoni. Per ogni canzone è visualizzato #, titolo, artisti, album, data di pubblicazione e durata.
  - Se l'utente è l'autore della playlist, vengono visualizzati anche il pulsante per modificare le informazioni della playlist (titolo, descrizione, tags, visibilità), il pulsante per cancellare la playlist e la barra di ricerca per cercare nuove canzoni. I risultati della ricerca hanno un pulsante per aggiungerle alla playlist. Le canzoni della playlist hanno un pulsante per rimuoverle.

- Se l'utente non è l'autore della playlist, viene mostrata la playlist solo se pubblica.

Il pulsante "Modify info" nella visualizzazione della playlist porta alla pagina:

- http://localhost:4000/playlist/:title/modify : pagina per la modifica delle informazioni della playlist con titolo ":title" (componente URI codificato). L'utente può modificare titolo, descrizione, tags e visibilità. Le informazioni modificate vengono salvate solo se l'utente è l'autore della playlist.

L'opzione "Your playlists" del dropdown menu "Playlists" della barra di navigazione porta alla pagina:

- http://localhost:4000/library : pagina con la libreria di playlist dell'utente.
  Nella sezione "Your playlists" mostra le playlist create dall'utente. Se non ce n'è, invita l'utente a crearne una.
  Nella sezione "The playlists you like" mostra le playlist a cui l'utente ha messo like. Se non ce n'è, invita l'utente a fare una ricerca.

L'opzione "Edit your favourite genres" del dropdown menu "Preferences" della barra di navigazione porta alla pagina:

- http://localhost:4000/choose-genres : pagina per la scelta dei generi musicali preferiti.
  L'utente visualizza l'elenco dei generi musicali, in cui sono già selezionati i suoi generi musicali preferiti attuali. Può modificare e salvare i generi musicali preferiti, o annullare.

L'opzione "Edit your favourite artists" del dropdown menu "Preferences" della barra di navigazione porta alla pagina:

- http://localhost:4000/choose-artists : pagina per la scelta degli artisti musicali preferiti.
  L'utente visualizza una barra di ricerca e l'elenco dei suoi artisti preferiti attuali. Può cercare nuovi artisti e aggiungerli alla lista dei suoi artisti musicali preferiti, oppure rimuovere degli artisti dalla lista dei suoi artisti musicali preferiti. Può salvare le nuove preferenze o annullare.

Il link "Profile" della barra di navigazione porta alla pagina:

- http://localhost:4000/profile : pagina del profilo dell'utente. Mostra il profilo pubblico dell'utente.
  Nella prima sezione mostra i generi musicali preferiti, nella seconda mostra gli artisti musicali preferiti e nella terza mostra le anteprime delle playlist pubbliche che ha creato. Ogni playlist può essere visualizzata con il pulsante "View playlist".

Il link "Account" della barra di navigazione porta alla pagina:

- http://localhost:4000/account : pagina dell'account dell'utente. Vengono riepilogate le informazioni dell'account (username ed email). L'utente può modificare username e/o email e/o inserire una nuova password (ripetendola due volte per conferma) e salvare le modifiche. Prima di effettuare la modifica, vengono riepilogate le modifiche e viene chiesta conferma all'utente.
  L'utente può anche cancellare l'account con il pulsante "Delete account". Anche in questo caso, viene chiesta conferma all'utente prima di eseguire l'operazione.

Il link "Logout" della barra di navigazione esegue il logout e riporta alla pagina di login.

Struttura del codice sorgente

La cartella del progetto è strutturata così:

- **docs/**
  - PWM_project_22_23.pdf -> requisiti del progetto
  - Relazione_SNM_Papetti.pdf -> questa relazione
  - Spotify_API.pdf -> indicazioni per l'uso delle API Spotify
- **node_modules/** -> files e cartelle creati da Node.js con il comando nmp install
- **src/**
  - **controllers/**
    - authenticationController.js
    - dbController.js
    - pagesController.js
    - spotifyController.js
  - **css/**
    - default.css
  - **html/**
    - account.html
    - choose-artists.html
    - choose-genres.html
    - create-playlist.html
    - home.html
    - index.html
    - library.html
    - login.html
    - modify-playlist.html
    - playlist.html
    - profile.html
    - search-page.html
    - signup.html
  - **images/**
    - blank-profile-picture.png
  - **middlewares/**
    - authorization.js
    - routes.js
  - **models/**
    - Playlist.js
    - User.js
  - **utils/**
    - account.js
    - artists.js
    - genres.js
    - library.js
    - login.js
    - menu.js
    - pages.js
    - playlist.js
    - profile.js

    - search.js
    - signup.js
- .env -> file di environment dell'applicazione. Contiene, ad esempio, parametri per il server, URI del database, segreti per la codifica/decodifica dei token
- index.js -> file da cui parte l'esecuzione dell'applicazione
- package.json -> file creato da Node.js per l'applicazione
- package-lock.json -> file creato da Node.js per l'applicazione

Ho diviso i files JavaScript in varie cartelle a seconda del loro ruolo.

Nella cartella utils/ ho inserito gli script che gestiscono il frontend, ovvero la scrittura dinamica di pagine html e la prima verifica, preparazione e inoltro delle richieste alle API dell'applicazione.

Nella cartella middlewares/ ho inserito

- il file routes.js con le routes di tutte le richieste alle API dell'applicazione
- il file authorization.js che, quando necessario, verifica l'autorizzazione e poi inoltra la richiesta al controller corrispondente.

Nella cartella controllers/ ho inserito i files con le funzioni che eseguono le richieste dell'applicazione, divise per tipo: richieste di autenticazione (authenticationController.js), richieste al database (dbController.js), richieste di pagine html (pagesController.js) e richieste alle API di Spotify (spotifyController.js).

Nella cartella models/ ho inserito le definizioni dei modelli di MongoDB per le collezioni usate dall'applicazione, ovvero gli utenti e le playlist.

Il modello User per gli utenti ha campi: id, username, email, password, createdAt, genres, artists, playlists, liked_playlists. Il campo username è un indice testuale. Ho scelto di salvare gli artisti preferiti come un array di Spotify_ID (tipo String) e di salvare le playlist create e preferite come array di _id di documenti Playlist.

Il modello Playlist per le playlist ha campi: id, title, descripted_by, tags, author, is_public, songs, likes. I campi title e tags sono indici testuali. Ho scelto di salvare le canzoni come array di Spotify_ID (tipo String).

Implementazione delle operazioni richieste

Gli url indicati nelle sezioni seguenti sono la parte variabile dell'url; è sottinteso che siano sempre preceduti dall'url di base http://localhost:4000 .

Registrazione (signup) e login al sito

L'utente inserisce email e password (login) o username, email, password e conferma password (signup) e clicca sul pulsante di login/signup.

Faccio una prima verifica dell'input nel frontend, verificando che l'email sia della forma any@any.any (eventualmente anche con altri punti) e che la password abbia fra 8 e 30 caratteri. Per il signup verifico anche che username abbia fra 3 e 15 caratteri e che le due password inserite siano uguali. Se c'è qualche campo non valido, lo mostro all'utente senza far partire la richiesta.

Se l'input passa la verifica, invio la richiesta di login/signup con metodo POST, allegando le informazioni dell'utente. La richiesta viene inoltrata all'authenticationController (url: /login, funzione: login) (url: /signup, funzione: signup). Qui, nel caso di errori imprevisti, restituisco status 500.

Per il signup:

Mi connetto al database. Cerco nel database se esiste già un utente con quell'email: se sì, restituisco status 409 e mostro all'utente che l'email non è valida. Faccio la stessa cosa per l'username (email e username sono unique).

Creo il nuovo utente nel database, salvando un hash della password.

Creo un JSON Web Token per l'autenticazione all'interno dell'applicazione e lo restituisco all'utente in un cookie http only.

Faccio richiesta del token per l'autenticazione nelle API Spotify e lo restituisco all'utente in un altro cookie http only.

Restituisco status 201. Nel frontend, metto nel session storage due array vuoti per i generi musicali preferiti e per gli artisti preferiti (uso il sessionStorage come cache). Infine indirizzo l'utente nella pagina per la scelta dei generi musicali preferiti.

Per il login:

Mi connetto al database e cerco l'utente nel database tramite l'email. Se non trovo un utente da autenticare restituisco status 401 e mostro all'utente che l'email non è valida.

Faccio l'hash della password e verifico che corrisponda a quello salvato nel database. Se non corrisponde restituisco status 401 e mostro all'utente che la password non è valida.

Creo un JSON Web Token per l'autenticazione all'interno dell'applicazione e lo restituisco all'utente in un cookie http only.

Faccio richiesta del token per l'autenticazione nelle API Spotify e lo restituisco all'utente in un altro cookie http only.

Restituisco status 200 e indirizzo l'utente nella home page.

Gestione token

Il token dell'applicazione è valido per 6 ore, dopo di che l'utente è reindirizzato al login.

Il token per le API Spotify è valido per un'ora: viene rinnovato in modo "lazy" nello spotifyController, quando, prima di fare richieste alle API Spotify, viene verificato.

Autorizzazione

L'autorizzazione, ovvero il controllo della validità del token e l'estrazione dell'id utente, viene eseguita nel middleware authorization.js per tutte le richieste alle API, escluse quelle delle pagine html pubbliche (base, login e signup) e le richieste con metodo POST di login e signup.

Qualunque funzione nei controllers, se fa uso dell'id dell'utente, usa l'id estratto dal token.

Session storage

Uso il session storage per salvare informazioni ottenute in risposta ad alcune richieste API, per limitare il numero di esecuzioni della stessa richiesta, la cui risposta sarà ancora la stessa.

Logout

L'utente può eseguire il logout cliccando sul link "Logout" nella barra di navigazione.

Quando l'utente clicca "Logout", svuoto il session storage e faccio la richiesta di logout, con metodo GET, gestita da authenticationController (url: /logout, funzione: logout).

Chiudo la connessione con il database, restituisco status 200 e comunico di eliminare i cookies con i token. Infine indirizzo l'utente alla pagina di login.

Gestione dei dati utente: account

La pagina per la visualizzazione delle informazioni dell'account permette di visualizzare username ed email, di modificarli e di inserire una nuova password, confermandola. Inoltre, ha un pulsante per eliminare l'account.

La pagina viene compilata dinamicamente con le informazioni attuali (username ed email), ottenute dal session storage o altrimenti tramite una richiesta con metodo GET gestita dal dbController (url: /get-account, funzione: get_account_info). Le informazioni ottenute dal database vengono salvate nel session storage.

Modifica account

Quando l'utente clicca il pulsante per salvare le modifiche, verifico che l'input sia valido (secondo gli stessi criteri della verifica pre-signup, con l'opzione di non specificare nessuna nuova password). Se ci sono campi non validi, li mostro all'utente senza eseguire la richiesta.

Se sono tutti validi, appare un confirm dialog in cui si riassumono quali informazioni sono state modificate e i loro nuovi valori (password non in chiaro), e si chiede conferma di volerle salvare.

Se confermata, la richiesta con metodo PUT è gestita dal dbController (url: /account, funzione: update_account). Qui controllo che email o username non siano già registrati per un altro utente; in quel caso restituisco status 409 e mostro i campi non validi all'utente.

Fatta la verifica, aggiorno l'utente con le informazioni modificate e restituisco status 200.

Infine, salvo le nuove informazioni dell'account nel session storage e indirizzo l'utente alla home page.

Eliminazione account

Quando l'utente clicca il pulsante per eliminare l'account, appare un confirm dialog in cui si chiede conferma.

Se confermata, viene fatta la richiesta con metodo DELETE, gestita dal dbController (url: /account, funzione: delete_account). Qui elimino tutte le playlists create dall'utente e infine elimino l'utente. Ho deciso di non eliminare i likes messi dall'utente durante l'utilizzo dell'applicazione.

Se non trovo l'utente restituisco status 404. Se l'eliminazione va a buon fine restituisco status 200, elimino i cookies, e nel frontend svuoto il session storage e indirizzo l'utente all'indirizzo base. Altrimenti restituisco status 500.

Nota: richieste al database

Prima di tutte le richieste al database, nel dbController, viene verificata ed eventualmente ripristinata la connessione al database.

Inoltre, anche dove non ribadito espilictamente, nel caso non si trovi l'utente che ha fatto la richiesta, viene restituito status 404.

In quasi tutte le risposte, viene restituito anche un messaggio di successo/errore.

Gestione dei dati utente: preferenze musicali

L'utente può visualizzare e modificare i propri generi e artisti musicali preferiti.

Generi musicali preferiti

La pagina choose-genres.html viene popolata dinamicamente con la lista di tutti i generi musicali di Spotify, mostrando già selezionati gli attuali generi preferiti dell'utente.

I generi musicali di Spotify vengono ottenuti dal session storage o altrimenti tramite richiesta con metodo GET gestita da spotifyController (url: /get-genres, funzione: get_spotify_genres). La richiesta fa una richiesta con metodo GET alle API Spotify. I generi così ottenuti vengono salvati nel session storage.

Gli attuali generi musicali preferiti vengono ottenuti dal session storage o altrimenti tramite richiesta con metodo GET gestita da dbController (url: /get-user-genres, funzione: get_user_genres). La richiesta ottiene l'array dei generi preferiti dall'utente, eventualmente vuoto. I generi preferiti ottenuti vengono salvati nel session storage.

Quando l'utente clicca il pulsante per salvare i nuovi generi preferiti, prima controllo se ci sono state modifiche: in caso contrario, indirizzo l'utente alla home page oppure, se l'utente ha appena eseguito il signup, lo indirizzo alla pagina per la scelta degli artisti preferiti.

Se ci sono state modifiche, faccio la richiesta con metodo PUT per sovrascrivere i generi musicali preferiti dell'utente (url: /genres, funzione: update_genres). Se non trovo l'utente restituisco status 404. Se aggiorno i generi dell'utente con successo restituisco status 200.

Aggiornati con successo i generi musicali, salvo i generi aggiornati nel session storage. Se l'utente ha appena eseguito il signup, lo indirizzo alla pagina per la scelta degli artisti preferiti, altrimenti lo indirizzo alla home page.

Artisti musicali preferiti

La pagina choose-artists.html contiene una barra di ricerca, spazio per mostrare eventuali risultati, e la lista degli attuali artisti preferiti dell'utente, che viene popolata dinamicamente.

Ho scelto di salvare nel database solo l'array degli Spotify_ID degli artisti preferiti dell'utente. Quindi, per popolare la lista degli attuali artisti preferiti, per prima cosa ottengo i loro Spotify_ID, dal session storage o altrimenti tramite richiesta con metodo GET gestita da dbController (url: /artists, funzione: get_artists). Gli artisti preferiti ottenuti vengono salvati nel session storage.

Controllo che ci sia almeno uno Spotify_ID da cercare; quindi, faccio richiesta con metodo GET per ottenere i dati degli artisti, gestita da spotifyController (url: /artists/:artists_ids, funzione: get_many_artists). Se la richiesta fallisce restituisco status 500.

Se la richiesta ha successo, restituisco status 200 e solo i dati utili all'applicazione (Spotify_ID, nome, url dell'immagine più piccola, generi musicali). Infine popolo la lista.

L'utente può usare la barra di ricerca. La ricerca fa scattare una richiesta con metodo GET gestita da spotifyController (url: /search/:query/:type/:limit/:offset con type=artist, limit=6, offset=0, funzione: search). Per praticità ho deciso di mostrare solo i primi 6 artisti risultato della ricerca.

La richiesta è inoltrata alle API Spotify. Nel frontend estraggo nome e url dell'immagine più piccola, e mostro un elenco di artisti, ognuno con un pulsante "Add". Se dovesse mancare l'url dell'immagine, uso l'immagine blank-profile-picture.png nella cartella images/.

Se l'utente clicca su "Add", l'artista viene spostato, all'interno della pagina html, nella lista degli artisti preferiti. Ogni artista della lista dei preferiti ha un pulsante "Remove", che lo elimina dalla pagina html.

Una volta che l'utente è soddisfatto della sua lista degli artisti preferiti, clicca il pulsante "Save".

Raccolgo gli Spotify_ID degli artisti preferiti dalla lista e li salvo facendo richiesta con metodo PUT, gestita da dbController (url: /artists, funzione: update_artists). Se non trovo l'utente restituisco status 404, altrimenti sovrascrivo gli artisti preferiti e restituisco status 200.

Infine, salvo gli Spotify_ID dei nuovi artisti preferiti nel session storage e indirizzo l'utente alla home page.

Playlist: creazione

L'utente può creare playlist dall'opzione "Create new playlist" del link "Playlists" sulla barra di navigazione. Se non ha ancora creato nessuna playlist, può crearne una anche dal link "Create a playlist" nella pagina con la libreria delle sue playlist.

La pagina create-playlist.html chiede in input titolo, descrizione, tags e visibilità (default: public).

Quando l'utente clicca il pulsante "Save", verifico che tutti i campi siano stati compilati; in caso contrario, mostro all'utente i campi non validi senza far partire richieste.

Se i campi sono validi, faccio una richiesta con metodo POST, gestita da dbController (url: /create-playlist, funzione: create_playlist).

Qui controllo che il titolo non corrisponda a una playlist già esistente (il titolo è unique); in quel caso restituisco status 409 e mostro all'utente che il titolo non è valido.

Quindi, creo la nuova playlist e inserisco il suo id nelle playlist dell'utente; restituisco status 201.

In questo caso, nel frontend elimino dal session storage l'elenco di playlist dell'utente (che, se presente, non è più aggiornato) e indirizzo l'utente sulla pagina di visualizzazione della playlist appena creata.

Playlist: visualizzazione

La visualizzazione di una playlist varia a seconda che sia privata o pubblica, e nel caso sia pubblica varia a seconda che l'utente ne sia l'autore o meno. Una playlist privata può essere visualizzata solo dal suo autore.

La pagina di visualizzazione ha url /playlist/title , dove "title" è il titolo della playlist codificato come URI component. La pagina viene popolata dinamicamente.

Ottengo il titolo della playlist da richiedere dall'url della pagina, quindi faccio richiesta con metodo GET, gestita da dbController (url: /playlist-info/:title, funzione: get_playlist).

Se non trovo la playlist o l'utente restituisco status 404. Se la playlist non appartiene all'utente ed è una playlist privata, restituisco status 401.

Altrimenti, restituisco status 200 e le informazioni della playlist (titolo, descrizione, tags, visibilità, username dell'autore, lista di Spotify_ID delle canzoni, numero di likes, se appartiene all'utente, se l'utente aveva già messo like).

Nel frontend, inizio a popolare la pagina (titolo, descrizione, tags, visibilità, autore); se la playlist è pubblica, mostro anche il numero di likes e il pulsante "like" o "liked" opportuno.

Se c'è almeno una canzone, faccio richiesta per ottenere le informazioni delle canzoni, con metodo GET, gestita da spotifyController (url: /songs/:song_ids, funzione: get_many_songs).

Qui faccio richiesta per ottenere tutte le informazioni delle canzoni alle API Spotify, con metodo GET. Restituisco status 200 e le informazioni utili all'applicazione (per ogni canzone, Spotify_ID, titolo, stringa con l'elenco degli artisti, titolo dell'album, data di pubblicazione, durata in ms).

Per ogni canzone, scrivo le informazioni ottenute (la durata in formato mins:secs). Ogni canzone ha un pulsante "Remove".

Se l'utente non è il proprietario della playlist, tolgo tutti i pulsanti "Remove".

Se l'utente è il proprietario della playlist, aggiungo i pulsanti per modificare e cancellare la playlist e aggiungo una barra di ricerca per cercare e aggiungere nuove canzoni.

Nota: richieste GET

Anche dove non esplicitato, tutti gli eventuali parametri di tutte le richieste con metodo GET vengono inviati codificando il parametro con encodeURIComponent.

Playlist: modifica

Aggiungere una canzone

Se l'utente è l'autore della playlist, può fare la ricerca di canzoni da aggiungere nella barra di ricerca. La ricerca fa scattare una richiesta con metodo GET gestita da spotifyController (url: /search/:query/:type/:limit/:offset con type=track, limit=10, offset=0, funzione: search). Per praticità ho deciso di mostrare solo i primi 10 risultati della ricerca.

La richiesta è inoltrata alle API Spotify. Nel frontend, per ogni canzone estraggo le informazioni (titolo, artisti, album, data di pubblicazione, durata in ms) e la mostro all'utente (durata in formato mins:secs), con un pulsante "Add". Mostro all'utente la durata totale della playlist.

Ho deciso di non mostrare i generi delle canzoni: non sono presenti nella risposta delle API Spotify; ci sarebbero i generi associati agli autori, ma non sono per forza associati alla canzone specifica, inoltre, sarebbero poco pratici da mostrare e poco interessanti per l'utente.

Quando l'utente clicca il pulsante "Add" di una canzone, questa viene spostata nella pagina html dai risultati della ricerca all'elenco delle canzoni della playlist, e il pulsante "Add" viene sostituito dal pulsante "Remove". Faccio richiesta con metodo PUT per aggiungere la canzone alla playlist nel database, gestita da dbController (url: /playlist/add-song, funzione: add_song_to_playlist).

Se non trovo la playlist, restituisco status 404. Ricontrollo che l'utente sia l'autore della playlist, e in caso contrario restituisco status 401.

Aggiungo lo Spotify_ID della canzone all'elenco di canzoni della playlist e restituisco status 200. Se la canzone era già presente, non la aggiungo e restituisco status 204.

Se ho aggiunto correttamente la nuova canzone, aggiorno la durata totale della playlist.

Rimuovere una canzone

Quando l'utente clicca il pulsante "Remove" di una canzone, faccio richiesta con metodo PUT per rimuovere la canzone dal database, gestita da dbController (url: /playlist/remove-song, funzione: remove_song_from_playlist).

Se non trovo la playlist, restituisco status 404. Ricontrollo che l'utente sia l'autore della playlist, e in caso contrario restituisco status 401.

Rimuovo lo Spotify_ID della canzone all'elenco di canzoni della playlist e restituisco status 200. Se la canzone non era presente, non la rimuovo e restituisco status 204.

Se ho rimosso correttamente la canzone, aggiorno la durata totale della playlist.

Mettere o togliere il like

Ho considerato la possibilità di mettere like e contare i likes solo per le playlist pubbliche. È possibile mettere like anche alle proprie playlist.

Il pulsante "like" può passare dallo stato non premuto "like" allo stato premuto "liked", e viceversa. A seconda del cambiamento che avviene con il click dell'utente, imposto un'azione, che può essere di mettere like o rimuovere il like.

Faccio richiesta per aggiornare playlist e utente con il nuovo like oppure per rimuoverlo, con metodo POST, inviando nel corpo della richiesta azione e titolo della playlist. La richiesta è gestita da dbController (url: /like, funzione: like_dislike).

Qui, se non trovo playlist o utente restituisco 404. Se l'azione non è una delle due attese, restituisco status 400.

Se l'azione è di mettere il like: se l'utente aveva già messo like, restituisco status 400; altrimenti aumento di uno il numero di likes della playlist e la aggiungo alle playlist preferite dell'utente. Restituisco status 200.

Se l'azione è rimuovere il like: se non risulta che l'utente avesse messo like a quella playlist, restituisco status 400; altrimenti diminuisco di uno il numero di likes della playlist e la rimuovo dalle playlist preferite dell'utente. Restituisco status 200.

Modificare le informazioni della playlist

Se l'utente è l'autore della playlist, può cliccare il pulsante "Modify info" e accedere alla pagina di modifica delle informazioni della playlist, che viene popolata dinamicamente.

Le informazioni della playlist vengono ottenute facendo richiesta al database, con metodo GET, in modo analogo a quanto fatto per la pagina di visualizzazione della playlist.

Quando l'utente clicca il pulsante "Save changes", verifico che non ci siano campi vuoti: in quel caso, mostro i campi non validi senza fare richieste.

Faccio la richiesta per aggiornare le informazioni della playlist, con metodo PUT, inviando nel corpo il vecchio titolo e tutte le nuove informazioni. La richiesta è gestita da dbController (url: /playlist/update-info, funzione: update_playlist).

Cerco la playlist da aggiornare: se non la trovo restituisco status 404. Trovata la playlist, ricontrollo che l'utente sia l'autore; in caso contrario, restituisco status 401.

Sovrascrivo le informazioni della playlist e restituisco status 200. Nel frontend, elimino le playlist dell'utente dal session storage (non più aggiornate) e indirizzo l'utente alla visualizzazione della playlist (con il nuovo titolo).

Playlist: eliminazione

Se l'utente è l'autore della playlist, può eliminare la playlist cliccando sul pulsante "Delete playlist" nella pagina di visualizzazione della playlist.

Quando l'utente clicca "Delete playlist", appare un confirm dialog in cui si chiede conferma.

In caso di conferma, faccio richiesta di eliminazione con metodo DELETE, gestita da dbController (url: /playlist, funzione: delete_playlist).

Se non trovo playlist o utente, restituisco status 400. Trovata la playlist, ricontrollo che l'utente sia l'autore; in caso contrario, restituisco status 401.

Trovati playlist e utente, rimuovo la playlist dalle playlist dell'utente e poi elimino la playlist; restituisco status 200.

Non elimino la playlist dalle liste delle playlist preferite di tutti gli utenti che avevano messo like, perché dovrei far scorrere tutti gli utenti del database, popolando le playlist preferite di ogni utente. Ho adottato un approccio "lazy": quando vengono richieste le playlist preferite dell'utente, se una playlist della lista non esiste più, viene rimossa dalla lista in quel momento.

Eliminata con successo la playlist, elimino le playlist dell'utente dal session storage (non più aggiornate) e lo indirizzo alla sua libreria di playlist.

Playlist: libreria

Nella libreria l'utente può visualizzare tutte le playlist che ha creato e tutte le playlist a cui ha messo like (playlist preferite). La pagina html viene popolata dinamicamente.

Se l'utente non ha creato nessuna playlist, la sezione delle sue playlist glielo scrive e mostra un pulsante con cui creare una nuova playlist.

Se l'utente non ha messo like a nessuna playlist, la sezione delle sue playlist preferite glielo scrive e mostra un pulsante con cui indirizzarlo alla pagina di ricerca.

Per popolare la sezione delle playlist dell'utente, ottengo tutte le playlist e le loro informazioni per l'anteprima (titolo, descrizione, numero di likes, visibilità) dal session storage o altrimenti tramite richiesta con metodo GET gestita da dbController (url: /playlists, funzione: get_your_playlists).

Ottenute le playlist dell'utente dal database, le salvo nel session storage.

Per popolare la sezione delle playlist preferite, ottengo tutte le playlist e le loro informazioni per l'anteprima (titolo, descrizione, numero di likes, visibilità) dal session storage o altrimenti tramite richiesta con metodo GET gestita da dbController (url: /liked-playlists, funzione: get_liked_playlists).

Trovato l'utente nel database, raccolgo le informazioni di tutte le playlist preferite. Se trovo playlist che sono state eliminate, ovvero non riesco a trovare i dati associati a quell'id (sono undefined), le rimuovo dalle playlist preferite dell'utente. Restituisco status 200.

Ottenute le playlist preferite dal database, le salvo nel session storage.

Le playlist sono mostrate in anteprima (titolo, descrizione, numero di likes, visibilità, pulsante "View playlist") a gruppi di 6 per ciascuna sezione (playlist dell'utente e playlist preferite). L'utente può

navigare tra le playlist di ogni sezione con pulsanti prev e next, che permettono di scorrere le playlist come se fossero un array circolare infinito.

Ricerca

La pagina di ricerca mostra una barra di ricerca con un menù di selezione del tipo di ricerca (Playlist, Song, Artist, User profile).

Quando l'utente clicca "Search", in base al tipo di ricerca faccio una ricerca su Spotify (Song, Artist) o una ricerca nel database (Playlist, User profile).

Tutti i risultati (ad eccezione del tipo User profile) sono mostrati in numero ridotto, con la possibilità di scorrere i risultati con pulsanti prev/next.

Ricerca di canzoni e artisti

La ricerca viene eseguita dalle API Spotify, quindi posso cercare le canzoni per parole chiave (titolo canzone, nome artista, titolo album...) e gli artisti per nome (secondariamente, anche per genere).

Faccio richiesta di ricerca con metodo GET, gestita da spotifyController (url: /search/:query/:type/:limit/:offset, funzione: search).

Qui faccio richiesta di ricerca alle API Spotify, con metodo GET; se la ricerca va a buon fine restituisco status 200 e tutti i risultati.

Tipo "Song": mostro 12 canzoni per pagina di risultati. Per ogni canzone mostro titolo, artisti, album, data di pubblicazione e durata. Come detto per le canzoni nella visualizzazione delle playlist, non mostro i generi.

Tipo "Artist": mostro 6 artisti per pagina di risultati. Per ogni artista mostro immagine, nome e generi musicali.

Ricerca di playlist

Faccio richiesta di ricerca con metodo GET, gestita da dbController (url: /playlist/search/:query/:limit/:offset con limit=6, funzione: search_playlists).

Qui faccio la ricerca tra le playlist pubbliche del database, sia per titolo sia per tags, che sono indici testuali, limitando il numero di risultati a "limit" e usando "offset" come primo indice dei risultati da restituire. La ricerca è case insensitive, e trova risultati corrispondenti ad ognuna delle parole complete della query. Restituisco status 200 e l'array delle playlist trovate con tutte le loro informazioni (può essere vuoto).

Non ho implementato la ricerca di playlist tramite il titolo delle canzoni che contiene. La playlist salva le canzoni come un array di Spotify_ID: per poter fare le ricerca per canzone, avrei dovuto far scorrere tutte le playlist, popolare l'array di canzoni di ognuna e cercare la query fra i titoli. Oppure avrei dovuto salvare, in ogni playlist, non solo gli Spotify_ID ma anche i titoli di tutte le canzoni. Non mi sembravano soluzioni fattibili, in termini di tempo e di memoria, specialmente al crescere del numero di playlist e canzoni. Forse avrei potuto salvare nel database un dizionario, usando come chiavi le parole singole e come items gli _id di tutte le playlist che contengono la parola chiave nel titolo.

Mostro 6 anteprime di playlist (titolo, descrizione, numero di likes, visibilità, pulsante "View playlist") per pagina di risultati.

Ricerca di profili utente

Faccio richiesta di ricerca con metodo GET, gestita da dbController (url: /profile/search/:username, funzione: search_profile).

Qui cerco tra gli utenti per username, che è indice testuale. La ricerca è case insensitive. L'username deve corrispondere all'username completo. Posso cercare più username separati da spazio e ricevere in risposta più utenti. Restituisco status 200 e l'array degli username degli utenti trovati.

Per ogni utente trovato, mostro l'username e un pulsante "View profile".

Visualizzazione profilo utente

L'utente può visualizzare il profilo pubblico degli utenti cercati cliccando il pulsante "View profile", oppure può visualizzare il proprio profilo dal link "Profile" della barra di navigazione.

Ho deciso di mostrare le playlist preferite solo all'utente nella sua libreria, e non nel profilo pubblico.

Il profilo dell'utente viene popolato dinamicamente e mostra tre sezioni: generi musicali preferiti, artisti preferiti e playlist pubbliche create.

Per popolare il profilo, faccio una richiesta al database con metodo GET, gestita da dbController (url: /profile-info/:username o /profile-info, funzione: get_profile).

Se ricevo un parametro username, lo uso per cercare l'utente; altrimenti cerco l'utente tramite l'id estratto dal token in fase di autorizzazione.

Se non trovo l'utente, restituisco status 404. Altrimenti, restituisco status 200 e restituisco i generi musicali preferiti, gli artisti preferiti e le playlist pubbliche di cui quell'utente è autore, con tutte le informazioni necessarie alle anteprime (titolo, descrizione, numero di likes, visibilità).

Mostro i generi preferiti, gli artisti preferiti (immagine, nome, generi musicali) e le anteprime delle playlist pubbliche (titolo, descrizione, numero di likes, visibilità, pulsante "View playlist").

TO DO

- generare la documentazione automatica degli end-point dell'applicazione con uno swagger
- usare il cacheStorage vero e proprio per fare la cache delle richieste alle API, invece di usare il sessionStorage come cache di dati
- nel caso di input non valido, oltre a mostrare il campo interessato, scrivere all'utente il motivo per cui non era valido
- creare una pagina "code 404 – Not found" da mostrare all'utente quando riceve dalle API una risposta con status 400-499
- permettere ricerca, login e signup con la pressione del tasto Enter
- migliorare la home page
- rendere più carino e intuitivo il pulsante "like" delle playlist