GESTIONE COMUNITÀ MUSICALI ONLINE – MATTEO MANCINI (906526)

Per quanto riguarda la realizzazione del progetto “Gestione comunità musicali online” la prima cosa che ho fatto è stata pensare alla struttura dell’applicativo.

Ho pensato di strutturarlo in 7 diverse pagine Html tutte contenenti lo stesso header e footer:

1. Index.html: home page che contiene tutti i bottoni per muoversi tra le pagine.
2. Register.html: pagina per registrarsi al sito web.
3. Login.html: pagina per loggare con l’utente desiderato (già registrato) all’interno del sito web.
4. Admin.html: pagina con le informazioni dell’utente loggato/registrato. La pagina contiene anche una sezione apposita per scegliere dei generi preferiti.
5. Songs.html: pagina predisposta per la ricerca della musica tramite l’API di Spotify. La pagina mostra anche delle canzoni suggerite nel caso le avessimo selezionate dalla pagina Admin.html.
6. Playlist.html: pagina in cui è possibile vedere le playlist pubbliche di tutti gli utenti che ne hanno creata una e le playlist private dell’utente loggato.
7. Details.html: pagina in cui è possibile vedere il dettaglio della playlist selezionata con tutte le canzoni all’interno e il dettaglio delle stesse.

HEADER

Ho creato un header classico uguale per tutte le pagine composto da un logo a sinistra, il menù con i link per muoversi tra le pagine al centro e a destra i link alla pagina per la creazione dell’account, per la pagina dell’admin e quella per eseguire il login all’interno del sito.

FOOTER

All’interno del footer ho posizionato il logo del sito.

DESCRIZIONE PAGINE

- Index.html
  La home page è molto semplice con 4 bottoni che rimandano alle parti principali del sito a sinistra. Per separarle ho messo tutto all’interno di due div con classe “col-lg-6” in modo da dividere la pagina in due verticalmente grazie al tool Bootstrap.

- Register.html
  Per la pagina di registrazione ho diviso la pagina in due parti, a destra ho creato il form per la registrazione con 3 campi, mail, password e conferma password e un bottone per sottomettere la richiesta con classe “register-action”. Questa classe viene ripresa in javascript “const registerAction = document.querySelector('.register-action');”. Al click del bottone, la sessione viene terminata e si crea un nuovo user prendendo i dati dai campi del form con “formAction”. A questo punto verifico che le password siano identiche e che nello storage non ci sia già un user con la stessa mail. Se questo non esiste allora inserisco i nuovi dati nello storage. Se i dati non sono corretti l’utente viene avvisato tramite delle scritte di errore.

- Login.html
  La struttura della pagina di login è simile a quella per la registrazione. A destra un form con un campo mail e password e un bottone per sottomettere la richiesta con classe “login-action”. Questa classe viene ripresa in javascript in maniera analoga alla registrazione. Anche in questo caso recupero i dati

degli input dal form e cerco se l'utente esiste cercando se la mail inserita coincide con quella nello storage tramite "store.getBy". Se trovo la mail verifico che anche la password sia uguale e in caso positivo attivo una sessione con quell'user mandandolo alla pagina di admin, altrimenti pulisco il form e mando un messaggio di errore.

- Admin.html
  La pagina è composta da due sezioni (separate sempre tramite colonne di Bootstrap). La colonna di sinistra contiene un'immagine di profilo con sotto i campi principali (nome, e-mail). Sotto questi campi troviamo i bottoni per creare una playlist e cancellare l'account.
  Il tasto di creazione della playlist, se cliccato, fa comparire un modal in cui inserire il nome della playlist e una descrizione. Inoltre, c'è la possibilità di flaggare una checkbox per indicare se deve essere pubblica o privata. Una volta creata, comparirà una box sotto i dati dell'utente, contenente il nome della playlist e il numero di canzoni al suo interno.
  I dati della playlist vengono inseriti all'interno dello storage sotto la voce "Playlist".
  Il tasto della cancellazione, se cliccato, fa comparire un modal con la richiesta esplicita di cancellazione dell'account e se la risposta è positiva, chiude la sessione ed elimina l'utente dallo storage, mandando l'utente alla pagina di login.
  Nella colonna di destra invece c'è una box dove poter inserire il nome dell'utente con un tasto, il quale, se premuto, aggiunge il nome nella sezione "fullname" dell'"users" all'interno dello storage, se il nome già esiste lo sostituisce. Sotto c'è uno spazio per scegliere il genere preferito. I generi li prendo facendo richiesta a Spotify tramite api prendendo il token con "getToken" e grazie a questo richiedendo i generi grazie a "getGenre". In seguito, recupero il valore della checkbox selezionata, recupero l'user e inserisco il genere preferito nella sezione genres dell'user all'interno dello storage. Questo mi servirà per dare delle canzoni consigliate all'interno della pagina "Songs.html".

- Songs.html
  In questa pagina ho inserito una barra di ricerca in cui inserire un titolo di una canzone che verrà ricercata all'interno di Spotify tramite il tasto "Cerca".
  Tramite la funzione "search" posso ricercare su Spotify l'elenco di canzoni con il nome inserito dall'utente e l'api mi restituirà i risultati che saranno visibili nella parte sottostante al box di ricerca.
  Per ogni canzone posso aggiungerla ai preferiti nella playlist desiderata (tramite un modal) oppure posso riprodurla tramite un link diretto a Spotify.

- Playlist.html
  In questa pagina compariranno, in alto, tutte le playlist pubbliche di tutti gli utenti registrati, mentre in basso tutte le playlist private del singolo utente. Tutte le playlist, con tutte le canzoni all'interno, verranno salvate sullo storage.
  A sinistra ho inserito una barra di ricerca per ricercare la playlist desiderata in base al nome o alla descrizione.
- Details.html
  In questa pagina viene mostrato il dettaglio della playlist.
  In un box viene mostrato il titolo della playlist e il nome dell'utente che l'ha creata mentre, nella parte inferiore, vengono mostrate le canzoni all'interno della playlist. Queste hanno la copertina (presa da Spotify) e tutti i dati annessi (Nome, Durata, Data di pubblicazione, Numero all'interno dell'album, Popolarità). Queste informazioni sono raccolte tramite lo storage.
  Ogni canzone ha anche la possibilità di essere eliminata dalla playlist tramite il bottone con la stella e può essere riprodotta su Spotify attraverso il link.

DESCRIZIONE SCRIPT.JS

Innanzitutto, ho creato tre classi con i relativi costruttori, una per l'User, una per le Playlist e una per lo Storage.

Per quanto riguarda lo Storage ho creato 6 funzioni (get, getBy, set, delete, drop, update) che serviranno per scrivere/eliminare i dati all'interno dello storage ma anche per chiudere la sessione in caso di cancellazione dell'account.

In seguito, ho creato un sistema di notifiche tramite una arrow function che restituisce il tipo della notifica, il messaggio che si porta in pancia e l'azione che ne consegue.
Le notifiche possono essere di 5 tipologie (primaria, successo, pericolo e attenzione), tutte gestite tramite un operatore ternario tutte dalla durata di 2s.

Creo delle pagine "private" (admin.html, playlist.html, songs.html, details.html) per le quali, quando un utente non è ancora registrato, se le visita finisce nella pagina di login. Quando sarà registrato le potrà visitare.

A questo punto creo una funzione per la gestione delle playlist. Questo avviene tramite un modal nel quale si può inserire il nome della playlist, una descrizione e flaggare la checkbox per renderla pubblica (e quindi farla vedere a tutti gli utenti che si iscriveranno) o tenerla privata. Successivamente ho creato due funzioni per popolare l'oggetto Playlist.
Avendo queste informazioni sulle Playlist, inserite nello storage, posso creare una funzione per inizializzare la pagina di dettaglio della singola playlist.

In seguito, ho cercato di raccogliere in un punto tutte le azioni dell'utente per poi andarle a sviluppare nel dettaglio (registerAction, loginAction, updateAction, deleteAction, searchAction, searchPlaylistAction, addPlaylistAction):

- registerAction: è una funzione che serve per captare la registrazione dell'utente. Per prima cosa viene cancellata la sessione corrente e creato un nuovo User. Poi setto i valori delle password (campo password e campo conferma password) e, se questi dati vengono recuperati, li controllo. Per prima cosa verifico che le password siano identiche e, se nello storage non trovo l'utente, lo inserisco insieme a tutti i suoi dati e lo spedisco alla pagina di admin, altrimenti esce una notifica di utente già presente. Se le password non coincidono esce una notifica di errore.
- loginAction: è una funzione che serve per captare il login dell'utente. Per prima cosa viene cancellata la sessione corrente. Se i dati sono stati recuperati cerco nello storage una corrispondenza per mail e se la trovo vedo se le password corrispondono e se va tutto bene, dirotto l'utente verso la pagina di admin. Altrimenti, pulisco il form e mando una notifica di errore.
- updateAction: è una funzione che serve per aggiornare il nome dell'utente nella pagina di admin. Come nelle funzioni precedenti, se recupero i dati con successo aggiorno lo storage con il nuovo "fullname". Se tutto avviene nel modo corretto esce una notifica di successo dell'operazione.
- deleteAction: è una funzione che serve per cancellare l'utente dallo storage. Creo un messaggio per il modal di conferma eliminazione dell'account, attendo il responso dell'utente e se è positivo (quindi schiaccia sul bottone di eliminazione), cancella la sessione corrente ed elimina l'user dallo storage. L'utente viene dirottato nella homepage.
- searchAction: è una funzione che serve per cercare le canzoni sulla pagina songs.html. Se i dati vengono raccolti correttamente aspetto che venga fatta la ricerca tramite api su Spotify della canzone ricercata. Elenco tutte le canzoni all'interno di un container con tutti i dati provenienti da Spotify (copertina dell'album, nome dell'album, nome dell'artista, nome della canzone ecc…). Creo anche un bottone per ascoltare la canzone direttamente su spotify e un bottone che farà comparire un modal per inserire la canzone in una delle playlist che ho creato (pubbliche o private).
- searchPlaylistAction: è una funzione che serve per ricercare tra le playlist pubbliche create dagli utenti.

- addPlaylistAction: è una funzione che serve per creare una playlist usando la funzione keepUpdated che serve per tenere aggiornate con la sessione corrente le playlist, i generi musicali preferiti e serve per caricare le playlist private e pubbliche sulla pagina playlist.html.

Poi ho messo a punto tutte le funzioni per pescare i dati (tramite api) da spotify e farli vedere a schermo (getToken, getGenre, getFound, search):

- getToken: è una funzione per recuperare il token che deve essere incluso in ogni richiesta per essere autorizzati a prelevare i dati da Spotify.
- getGenre: è una funzione che prende il token di autorizzazione e fa una richiesta per i generi.
- getFound: è una funzione che prende il token e un input (la canzone richiesta). Questo input lo metto all'interno dell'url che mi restituirà la canzone cercata.
- Search: è una funzione che prende un input (titolo della canzone) e lo mette all'interno della funzione getFound per trovare tutte le tracce associate al nome cercato.

Ho creato una funzione per gestire il contenitore dei generi della pagina admin.html. questo posso farlo tramite le api e la funzione getGenre vista prima. Per ogni genere creo una checkbox e se questa viene flaggata, recupero il valore del genere, recupero l'user, verifico se il genere musicale è presente tra le scelte, se è già presente allora lo elimino, se non è presente lo aggiungo. Aggiorno lo storage e la sessione e mostro una notifica di successo.

Recuperati i generi preferiti, nella pagina songs.html, mostro delle canzoni suggerite in base ai generi selezionati.

CSS

Ho creato un foglio di stile css in cui ho apportato tutte le modifiche e le migliorie grafiche a tutte le componenti delle varie pagine.
Ho anche creato vari stili in base alla larghezza dello schermo per rendere il sito responsive.

TOOL

- Ho utilizzato il template SolMusic come base per l'impaginazione per poi andare a modificare ogni singola funzione e pagina per adattarle alle necessità delle richieste del progetto.
- Ho utilizzato il plugin Owl Carousel per l'organizzazione della home page.
- Ho utilizzato Slicknav, un plugin per rendere il menù responsive e trasformarlo in hamburger.
- Ho utilizzato bootstrap per la gestione delle colonne all'interno della pagina.
- Ho utilizzato jQuery per il plugin Slicknav.