elaborato implementazione e funzioni

Per la registrazione abbiamo deciso di importare un template predefinito da internet che abbiamo successivamente modificato in base alle esigenze.

Il template è formato da due sezioni :

REGISTRAZIONE

- Registrazione : quando un utente clicca sul button “ISCRIVITI” un’ animazione di css3 lo porterà all’ ambiente di creazione dell’account, dove abbiamo deciso di definire 3 campi input per la creazione del nuovo utente. Troviamo un input (di tipo text) “Nome” che prende in input il nome dell’utente , un input type password “Password” (l’ attributo type password ci permette di oscurare i caratteri in modo da rendere piu sicuro il mantenimento della password non essendo visibile) . Infine troviamo un input type email “Email” in cui l’utente immetterà la sua email che verrà verificata di default grazie all’attributo type dell’input .

  Per quanto riguarda il controllo dell’immissione di una corretta password, abbiamo deciso di utilizzare una regular expression (regEXP) descritta nell’attributo “pattern” dell’input quale ci permette di verificare quasi in modo automatico la correttezza della password secondo gli standard (deve contenere almeno 8 caratteri, almeno una lettera minuscola, almeno una lettera maiuscola e un numero) . RegExp = **"^(?=.*d)(?=.[a-z])(?=.*[A-Z])(?=.[a-zA-Z]).{8,}$"**

  le regEXP garantiscono un metodo efficace e flessibile per elaborare del testo. l’utilizzo di espressioni regolari consente di analizzare rapidamente grandi quantità di testo per:

  - Trovare modelli di caratteri specifici.
  - Convalidare il testo per assicurarsi che corrisponda a un modello predefinito, ad esempio un indirizzo di posta elettronica.
  - Estrarre, modificare, sostituire o eliminare sottostringhe di testo.

Tutti i campi input sono descritti dall’attributo “required” che ci permette di gestire con istantaneità e comodità l’utente in caso non digiti nulla negli input e cerchi comunque di creare l’account : in quel caso verrà segnalato tramite un’interfaccia standard del browser l’obbligo di completare i vari campi.

Quando l’utente ha completato tutti i campi della registrazione e clicca sul button “CREA ACCOUNT” (di tipo submit), viene richiamata una funzione javascript descritta tramite un addEventListener (che secondo noi era il metodo piu comodo per gestire gli eventi associati al rispettivo handler). La funzione formRegistration.addEventListener inizialmente prende gli utenti che sono gia presenti nel local storage per verificare che non ci sia già un utente registrato con i campi inseriti dall’ utente che sta provando a registrarsi ora (questo controllo lo effettuiamo tramite un filter sull’array degli utenti del local storage e controllando che non vi sia una email uguale a quella che ha inserito l’utente nuovo, nel caso sia già presente mostriamo un danger_alert che avvisa l’utente del corrispondente errore).

Se tutti i controlli sono andati a buon fine, viene creato l’oggetto json utente descritto da 3 attributi (nome, email, password) presi tramite un document.GetElementById(””).value di ogni singolo input. A questo punto l’utente può essere inserito nell’ array di utenti (”users”) presenti sulla piattaforma e possiamo settare l’array aggiornato nel local storage col nuovo utente appena creato. Infine, attraverso un window.location.href indirizziamo l’utente verso l’ ultimo step per la creazione dell’ account : il setting delle preferenze e dei generi musicali

LOGIN

- Login : quando l’utente desidera accedere, la schermata di default è quella del login mentre se si trova nella sezione registrazione può semplicemente cliccare sul button “ACCEDI” quale tramite un’ animazione porterà l’utente all’ambiente di accesso.

  Per il login abbiamo deciso di far accedere l’utente tramite due campi : l’Email e la Password. Troviamo perciò due input uno type=”email” e uno type=”password” nel quale l’utente metterà i suoi dati. Quando avrà inserito tutti i dati, l’utente dovrà cliccare sul pulsante sottostante “ACCEDI” che richiama una funzione javascript formLogin.addEventListener con il ruolo di controllare che tutti i dati relativi all’utente siano corretti (password ed email corrispondano

con quelle nel local storage dell'utente corrispondente), perciò estrae gli users dal local storage e si procede al controllo delle informazioni dell'utente che sta cercando di accedere alla piattaforma.

il controllo dell' email e della password viene effettuato tramite un metodo users.findIndex in cui si controlla se vi è una email che corrisponde con quella dell'utente che sta loggando e in seguito se anche la password corrisponde (se users.findIndex trova un'elemento incline alle condizioni definite ritorna l'indice dell'utente corrispondente, mentre se non trova nessuna corrispondenze ritorna -1). In caso affermativo l' utente ha inserito i dati giusti e viene quindi indirizzato alla pagina della home (home.html) seguendo l'action della form mentre nel caso i dati non siano corretti l'utente verrà avvisato da un danger_alert della situazione d'errrore e non potrà passare alla home.

Infine per quanto riguarda l'utente ci salviamo un session Id nel Session storage e una copia di questo dentro l'oggetto utente nella key "users" del local storage per gestire poi l'utenteLoggato, sapere sempre chi sta usando la piattaforma e gestire il controllo del session Id (se un utente cerca di passare direttamente alla home senza avere un session Id verrà indirizzato nuovamente sulla pagina del login, cosi evitando situazioni che potrebbero violare la piattaforma).

Dopo che l'utente ha effettuato il login, tramite una funzione javascript getToken() si controlla e si preleva il client_id per poter fare la fetch (dove sono descritti l'url, il metodo che in questo caso è una POST, gli headers che contengono le autorizzazioni per l'accesso all'api) sull'api di spotify per poter fare tutti i tipi di query. Se la richiesta è andata a buon fine, l'api fornirà un access token che salviamo nel local storage e che servirà per operare con le query dell'api (verrà inserito nelle autorizzazioni).

GENERI e PREFERENZE MUSICALI

Ambiente selezione generi/preferenze iniziali : da questo punto in poi ci troviamo sulla pagina Home.html e abbiamo deciso di operare su tutti gli elementi tramite il visually hidden : una tecnica che ci permette di mostrare e nascondere gli elementi in base alle esigenze della piattaforma (viene alternato lo style css display : "none"

e display : “block” dei vari elementi html; “none” per quando si vogliono nascondere gli elementi mentre block per quando si devono mostrare nella pagina). In questo caso dobbiamo mostrare il div riferito ai generi da selezionare quindi vengono nascosti di default tutti gli altri div della pagina e viene mantenuto il div dei generi tramite l’attributo display : “block”.

Nella schermata di selezione dei generi l’utente clicca sulle preferenze da lui scelte e le posiziona nel local storage dell’utente loggato tramite le seguenti funzioni :

- funzione getCategories( ) : tramite questa funzione verrà eseguita la query all’api per mostrare tutti i generi. Per prima cosa estraiamo l’access token dell’utente salvato nel local storage in precedenza e tramite questo inviamo una richiesta all’api per ricevere le informazioni sui generi (troviamo una fetch all’api su tutti i generi tramite l’url “https://api.spotify.com/v1/browse/categories”).
- funzione createCardGrid( ) : passandogli come parametro l’array di elementi della ricerca tramite il forEach( ) vengono create le Cards (i contenitori singoli che corrispondono ad ogni genere) usando il metodo InnerHTML in cui possiamo combinare del codice HTML in javascript per creare le card con i corrispondenti dati per ognuna. Queste Cards sono definite tutte come degli input di tipo “checkbox” poichè devono essere cliccabili per essere selezionate dall’utente e per essere in seguito salvate nell’account. Ogni card è un’ oggetto descritto da vari campi come il nome/titolo del genere, un’id, le icone tramite cui mostriamo le card e l’url corrispondente al genere.

  Le cards saranno posizionate nella pagina tramite uno “scheletro a griglia” gestito e creato dalla funzione creaGriglia( ) che attraverso due cicli for crea le righe e le colonne dove saranno inserite le Cards (il for piu esterno descrive che saranno 5 Cards per ogni riga e quello piu interno definisce il numero di righe) che saranno mostrate tramite la funzione showCategories( ) .
- funzione salvaGeneri( ) : questa funzione controlla tutte le Cards (checkbox) che sono state selezionate, le mette in un array “userCard” e le salva nel local storage inviando un avviso di successo nell’interfaccia utente quando l’utente clicca sul pulsante “SALVA”.

HOME

Dopo la selezione dei generi preferiti l’utente verrà indirizzato nella home vera e propria (viene nascosto il div dei generi e viene mostrato il div della home dell’utente) in cui può modificare tutte le informazioni relative al suo account tramite una sidebar posizionata sul lato sinistro della piattaforma formata di button che rimandano ad ogni div corrispondente.

SETTINGS PROFILO

Per quanto riguarda il Profilo, quando l’utente clicca sul bottone “Profile” gli verrà mostrato un div in cui ha la possibilità di cambiare il nome dell’ account, inserire o modificare la sua biografia e mostrare un’immagine del profilo (che è data di default dalla piattaforma) tramite dei campi input con a lato il relativo pulsante. Ogni button quando cliccato richiama un addEventListener che inserirà o aggiornerà le informazioni prelevate dagli input nel local storage.

SETTINGS ACCOUNT

l’ambiente “Account” permette all’utente di modificare le informazioni definite alla creazione dell’account perciò la email e la password. Come per la sezione Profilo, anche qui ci siamo avvalsi di utilizzare due campi della form input “Cambia email” e “cambia password “ contenuti in delle checkbox e un pulsante submit che richiama funzioni di aggiornamento/modifica del local storage dello user.

SETTINGS PREFERENZE

La sezione Preferenze da la possibiltà all’utente di modificare le preferenze e i generi da lui selezionati al momento della creazione dell’account. Mostriamo i generi presenti nel local storage dell’ utente loggato tramite una table e per ogni preferenza

l’utente ha la possibiltà di rimuoverla dal poprio account cliccando sul pulsante rosso definito con l’icona “X” (questo pulsante richiama una funzione che rimuoverà il genere corrispondente dal local storage). Inoltre si ha la possibilità di aggiungere preferenze cliccando sul relativo pulsante “Aggiungi preferenze” quale richiama una funzione che permette di rimostrare tutte le cards all’utente e di fargli selezionare i generi da aggiungere; in seguito verrà riaggiornato il local storage quando si clicca il pulsante salva)

SETTINGS PLAYLIST

Il prossimo button è quello delle playlist : è un button particolare perchè è inserito in un drag & drop poichè a sua volta troviamo 3 sotto-sezioni :

1. **Nuova playlist** : qui l’utente può creare una nuova playlist (definiamo la playlist come un object per poi essere descritta da tutti gli attributi). Troviamo perciò tutti i campi input per la creazione di una lista musicale (Nome playlist, Descrizione playlist, Tag playlist e una chechbox di tipo Switch importata da Bootstrap che dà la possibilità di rendere una playlist pubblica) ma soprattutto un input type=”text” ricerca canzoni dove l’utente aggiunge le canzoni alla playlist che sta creando. Questo campo di ricerca è associato ad un AddEventListener definito dal metodo ‘keyup’ per consentire di mostrare la ricerca in tempo reale ad ogni carattere inserito nell’input. Questa funzionalità combinata ad una fetch ci permette di utilizzare funzioni asincrone eseguite tramite un approccio sincrono. Una volta che ha completato tutti i campi, l’utente clicca sul pulsante Salva e verrà creato l’oggetto playlist che verrà aggiunto in un array quale sarà a sua volta aggiunto nel local storage nella sezione “Playlists” dello user corrispondente. Quando un utente esegue una ricerca verrà richiamata una funzione che crea l’url, esegue la fetch sulle track e tramite una funzione createTrackDetail( ) prendendo come parametri i risultati della fetch e il div mostra all’utente i risultati della ricerca in forma tabulare.
2. **Le tue playlist** : qui vengono mostrate le playlist dell’utente che vi è loggato. Vengono estratte le informazioni dal local storage dello user e tramite l’uso di una const “html” e le informazioni sull’oggetto playlist si forma una table (funzione fillTableMyPL( )) che descrive ogni playlist creata. Vi sono inoltre due button per la rimozione e modifica di ciascuna playlist : il bottone Delete definito

da una “X” rossa chiama una funzione che rimuove la playlist selezionata dallo storage mentre il bottone Modify (“X” blu) mostra un div dove è possibile rimuovere in modo selettivo le canzoni da quella playlist oppure aggiungere nuove canzoni tramite il campo Input text “**Cerca canzoni**”.

3. **Playlist pubbliche** : in questa parte vengono mostrate all’utente tutte le playlist pubbliche di ogni utente ed esso può decidere di importarle nel suo profilo. Piu nello specifico viene controllato nel local storage nella key ”publicPL” quali hanno l’attributo collaborative : “true” , ciò evidenzia che quella lista è pubblica e dovrà essere mostrata in questo div. Attraverso una determinata funzione che usa l’Inner.HTML si crea e si popola una tabella dove vengono mostrate tutte le playlist pubbliche con le relative informazioni e un pulsante “ADD” che richiama una funzione per importare la corrispondente playlist nell’account , un campo input di ricerca per ricercare le playlist tramite tag associativi e canzoni in esse contenute.

COMUNITÀ CONDIVISIONI

Proseguendo con la sidebar, troviamo la Comunità condivisioni : in quest’area l’utente trova alcuni campi di input per la creazione della propria comunità (nome, descrizione , utenti e playlist da inserire nella comunità). Quando ha completato tutti gli input e clicca su Salva comunità, viene richiamata una funzione che crea l’oggetto Comunità descritto da tutti gli attributi estratti dagli input, dalle checkbox e viene salvato nello storage degli utenti selezionati.

ALL COMMUNITY

Una volta Creata una community, l’utente può verificare nella sezione “All Community” tutte le community in cui lui è presente o che ha creato. In particolare troviamo una tabella con il nome di ogni comunità, l Owner della comunità (l’utente che l’ha creata) e i Members (i membri presenti in quella comunità). Come ultimo elemento vi è un pulsante “PL” che da la possibilità all’utente di aprire la comunità, visualizzare le playlist condivise dall’owner e decidere di importarle o meno nel proprio account (quando viene cliccato il pulsante “PL” viene mostrato un “Example modal” importato da bootstrap e tramite una funzione con un forEach vengono

mostrate tutte le playlist contenute nella comunità con la possibilità di essere importate cliccando sul button “ADD”).

CANCELLA UTENTE

L’ultima sezione è quella di “Cancella utente” : viene mostrato un div molto evidente poichè è un’azione che può risultare “pericolosa” quindi abbiamo deciso di mostrarlo in modo molto esplicito. Troviamo un pulsante “CANCELLA UTENTE” che se cliccato invoca una funzione in cui si estrae l’array di utenti dal local storage, si crea un nuovo array senza l’utente che ha deciso di cancellarsi e si setta il nuovo array aggiornato nel local storage (verrà rimosso il session_Id dal local storage e l’utente verrà indirizzato nuovamente alla schermata iniziale di registrazione).

LOGOUT

Un’aspetto molto importante che non si può tralasciare è la possibilità di cambiare utente : nella nav bar posizionata superiormente vi è l’opzione, in un dropDown menu, di fare Log Out. Quando l’utente clicca questo button viene richiamata una funzione che rimanda l’utente alla pagina di login (window.location.replace("index.html")), quindi dovrà eseguire di nuovo l’accesso e potrà di conseguenza cambiare anche account .

LOCAL STORAGE

Per quanto riguarda il Local Storage abbiamo deciso di dividerlo in 4 ambienti che abbiamo ritenuto come i piu importanti e a cui ogni utente deve avere “accesso” indirettamente per quanto riguarda il suo percorso sulla piattaforma :

- Users : qui troviamo un’array di oggetti utente (ogni utente è descritto a sua volta da vari campi quali email, password, generi preferiti, nome utente, biografia, playlist e infine una copia del session_Id)
- access_token : ci salviamo il token nel local storage per avere la possibilità di utilizzare l’api in ogni momento. è un campo molto importante poichè serve per autorizzare il client ad avere accesso alle informazioni dell’api

- PublicPL : in questa key salviamo tutte le pubbliche playlist di ogni utente descritte dagli attributi nome playlist, descrizione, tag e le canzoni contenute in esse (ci serve salvarle nel local storage poichè ogni utente che logga deve avere la possibilità di vedere le playlist pubbliche)
- Community : salviamo nel local storage l’insieme di ogni comunità e ciascuna sarà descritta da attributi come Owner e membri interni, per poi sapere a quali utenti mostrare e quali utenti possono operare con le corrispondenti community.

FINE