TuneMate - Social Network for Music

Tiziano Radicchi
12172A

July 13, 2023

Abstract

Si propone l'analisi e lo sviluppo di *TuneMate*, applicativo web per la gestione di un social network volto alla creazione e modifica di playlist musicali e alla loro condivisione con gli altri utenti della piattaforma.

Una demo dell'applicativo è raggiungibile qui.

Contents

1 Introduzione

TuneMate è un applicativo che permette ai suoi utenti di registrarsi via interfaccia web, inserendo informazioni come estremi personali, nonché informazioni relative ai propri interessi musicali. Successivamente, gli utenti hanno la possibilità di ricercare attraverso la piattaforma brani e canzoni, filtrando attraverso diversi parametri come titolo, autore e genere, nonché periodo di pubblicazione. Successivamente, si ha la possibilità di creare playlist personali, riorganizzarle e associare tag all'intera playlist. La playlist può essere poi resa pubblica in modo tale che altri utenti possano ricercarla attraverso una sezione apposita. Gli utenti hanno la possibilità di seguire diverse playlist pubbliche e ritrovarne facilmente riferimenti dalla loro pagina personale.

2 Analisi dei requisiti

Si procede ora con l'analisi dei requisiti da cui deriverà, nella sezione successiva, la progettazione dell'applicativo.

Le tre macro-aree funzionali richieste sono:

1. Gestione degli **utenti**
2. Gestione delle **playlist**
3. Gestione delle **condivisioni**

Le entità identificate e a cui si associano funzionalità sono:

- Brano
- Playlist
- Utente
- Condivisione

Per soddisfare pienamente le funzionalità richieste, si identificano inoltre le seguenti tipologie di utenze che opereranno sulla piattaforma:

- Utente *base*: a cui sono associate tutte le core functionalities citate fino ad ora.
- Utente *amministratore*: Può intervenire direttamente sul contenuto della piattaforma, eliminando utenti e playlist. Non essendo necessario al fine di implementare le funzionalità esplicitamente richieste, influenzerà il dataset utilizzato per memorizzare le informazioni degli utenti, ma non verrà implementato nella versione presentata.

Il concetto di utente *anonimo* corrisponde all'utenza che non ha effettuato autenticazione sulla piattaforma. In questo caso, non si prevedono funzionalità di alcun tipo, se non visualizzare alcune informazioni generiche sulle funzionalità della piattaforma e la realizzazione di login o registrazione.

Segue una tabella contenente un resoconto delle funzionalità identificate, successivamente approfondite singolarmente.

Core Functionalities	Analisi / Report	Funzionalità di supporto
Utente base: - Inserimento - Lettura dati - Modifica dati - Cancellazione	Utente base: - Generi preferiti	- Login e logout
Playlist: - Inserimento - Ricerca playlist - Modifica dati - Cancellazione	Playlist: - Tag più utilizzati - Brani più frequenti	
Brano: - Ricerca brano - Aggiunta brano a playlist		
Condivisione: - Inserimento - Ricerca condivisione - Cancellazione		

Le *Core Funcionalities* sono relative alle utenze base. Con *Analisi e report* si intendono alcune funzionalità ipotizzate e che potrebbero risultare utili di fronte ad un'utenza amministratore, per moderare al meglio il contenuto della piattaforma ottenendo informazioni di natura statistica sul contenuto. Le funzionalità trattate d'ora in avanti saranno implicitamente relative alle utenze base, essendo le uniche effettivamente implementate all'interno dell'applicativo.

3 Analisi funzionalità

Segue ora un'analisi approfondita delle funzionalità precedentemente identificate.

3.1 Utente

Un utente può registrarsi inserendo le seguenti informazioni:

- Nome, cognome, data di nascita
- Email, password e username
- Generi musicali e artisti preferiti

Tutti i campi sono obbligatori. è necessario specificare almeno un genere ed un artista preferito.
L'utente può poi modificare i propri dati tramite apposito form, nonché richiedere la cancellazione dell'account. Per la cancellazione dell'account si prevede l'eliminazione di tutti i dati relativi all'utente, dunque anche di eventuali playlist da lui create. Si noti come una scelta implementativa diversa che dovesse prevedere il mantenimento delle playlist create sulla piattaforma prevederrebbe comunque l'alterazione del record a DB della playlist, ad esempio con un'anonimizzazione dell'utente eliminato.
Per quanto riguarda la funzionalità *Lettura dati*, si prevede la possibilità da parte dell'utente di avere accesso alle proprie informazioni personali inserite in fase di registrazione anche in un secondo momento. Non si prevede quindi un "accesso" alle informazioni sotto forma di ricerca di altri utenti, funzionalità non prevista nell'applicativo. L'unico riferimento ottenibile da parte dell'utente loggato nei confronti di altri utenti della piattaforma consiste nel loro username nel caso di eventuali playlist rese pubbliche.

Infine, tra le funzionalità a supporto dell'applicativo vi è il login e il logout da parte dell'utente.

3.2 Playlist

Una nuova playlist può essere creata, specificando:

- Titolo
- Descrizione
- Tag (nel formato $\#tag$)

Le playlist sono associate agli utenti che le creano. Conseguentemente, i singoli utenti potranno consultare le proprie playlist da una sezione apposita.
Una volta creata una playlist, l'utente può visualizzare da una apposita sezione le informazioni relative come nome, descrizione, tag applicati, nonché numero di brani presenti e durata complessiva. Si prevede, oltre ai requisiti espressamente richiesti, anche un'informazione relativa al numero di follower di una playlist nel momento in cui questa dovesse essere pubblica.
L'utente che ne è proprietario può poi modificare il titolo della playlist, la descrizione, i tag associati, nonché modificare l'ordine dei brani presenti al loro interno o rimuoverli. Si fornisce poi la possibilità di eliminare la playlist. L'interfaccia offre la posssibilità di rendere la playlist pubblica o privata. Alla creazione le playlist sono private, dunque visibili solo all'utente che l'ha creata.
Per quanto riguarda la ricerca delle playlist ed in particolare per quelle rese pubbliche, l'utente autore avrà la possibilità di ritrovare riferimento a queste nella propria sezione personale. Inoltre, si fornisce un'interfaccia di ricerca per le playlist pubbliche, con la possibilità di filtrare eventualmente la ricerca per nome della playlist, tag associati e brani contenuti.

3.3 Brano

Per quanto riguarda i brani, questi possono essere ricercati attraverso una barra di ricerca secondo i seguenti criteri (combinabili tra loro):

- Titolo
- Autore
- Album
- Genere
- Periodo di pubblicazione

Si richiede di specificare almeno uno dei filtri sopra riportati.
Le informazioni relative ai brani esistenti non verranno ottenute dalla base di dati ipotizzata per l'applicativo (utilizzata invece per gestione utenze e playlist) bensì utilizzando le API REST offerte da Spotify.
Una volta ottenuta una lista di brani, la visualizzazione dei brani è realizzata attraverso una schermata che riporta l'elenco dei brani e le relative informazioni. Da qui, è possibile selezionarne uno e scegliere successicamente la playlist a cui aggiungerlo.
Si è quindi optato per evitare di proporre un'intera pagina di dettaglio dedicata al singolo brano, dal momento in cui le informazioni che l'utente desidera conoscere sono già riportate durante l'elencazione dei risultati.

3.4 Condivisione

TuneMate prevede poi la possibilità da parte degli utenti, a fronte di una playlist pubblica di cui non sono autori, di seguire la playlist. Quando l'utente diviene follower di una playlist pubblica questa viene riportata sul suo profilo, per facilitarne l'accesso. Il seguire una playlist ne permette un più rapido accesso, ma non lascia comunque spazio alla possibilità, da parte dell'utente, di alterare la struttura della playlist seguita. **Non** si prevede la possibilità per l'autore di seguire le sue stesse playlist pubbliche.
Si prevede inoltre una funzionalità aggiuntiva: di fronte ad una playlist pubblica o privata di cui l'utente è autore, questo ha la possilità di cederne la proprietà ad un altro utente di cui si conosce lo username. Questa diventerà di proprietà del destinatario, che ne erediterà la struttura, le informazioni e la visibilità. La cessione influenza anche lo status dell'utente destinatario circa il suo seguire o meno la playlist ricevuta, rimuovendo automaticamente il follow al fine di preservare le statistiche di following della playlist, qualvolta questa dovesse essere pubblica.
La cessione di proprietà di una playlist privata porterà alla perdità d'accesso a questa da parte dell'autore originale.

3.5 Use case

Segue ora uno schema *Use Case* per rappresentare in maniera grafica le funzionalità previste.

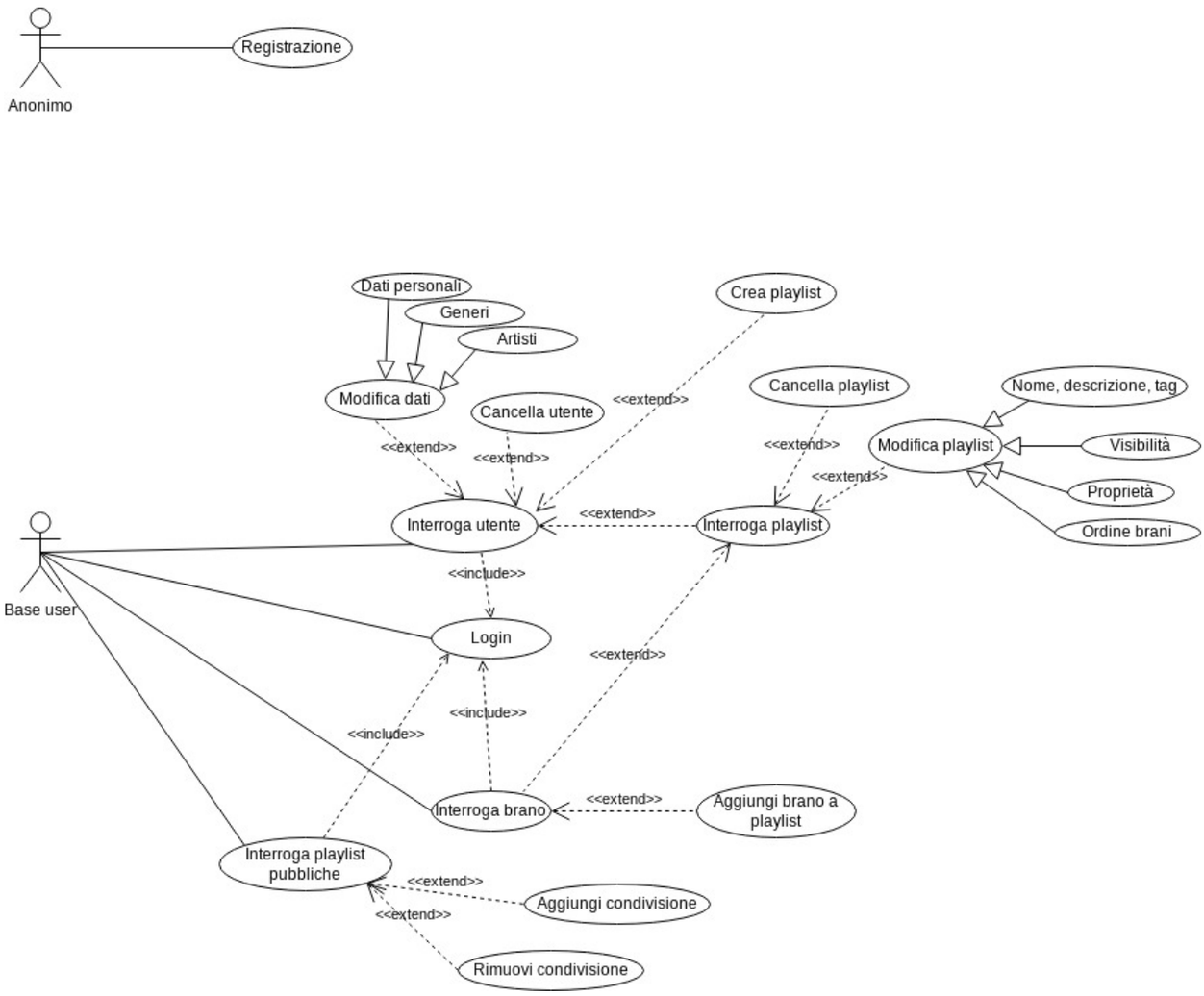

Figure 1: UML Use Case

I segmenti continui collegano l'autore con la funzionalità a lui associata; una funzionalità può prevedere che un'altra funzionalità sia stata precedentemente sfruttata, ad esempio nel caso della login (rami *include*); I rami *extend* invece rappresentano funzionalità raggiungibili dalle precedenti da loro puntate. Le frecce continue rappresentano invece delle generalizzazioni, ovvero delle precisazioni sui dati su cui la funzionalità opera.

Si chiarisce inoltre come, per semplicità rappresentativa, alcuni collegamenti di tipo *extend* siano stati omessi, preservando i principali. Alla luce della struttura frontend tra poco analizzata infatti vi è la possibilità, ad esempio, di interrogare playlist pubbliche anche a partire dall'interrogazione dell'utente.

4 Progettazione

4.1 Struttura

Alla luce dei requisiti dell'applicativo, si immagina di strutturarlo nella seguente modalità:

L'applicazione sarà suddisiva in una parte di frontend ed una di backend. In particolare, i file statici, comprendenti codice HTML, CSS e JS, comporranno il frontend. Il codice JS frontend è stato suddiviso in file in base alla pagina servita, per una maggiore leggibilità e manutenibilità. Vi sono poi file .js come *utils.js*, *genres.js* che includono funzioni riutilizzate in diverse pagine e sono dunque inclusi in diverse pagine HTML.
Il codice JS che comporrà il frontend si occuperà non solo di eventuali aspetti grafici, ma anche dell'interazione con le API REST di Spotify, **inviando** tuttavia in primo luogo **una richiesta** al backend.
Importante considerare che tutti i controlli di autenticazione posti frontend hanno esclusivamente una natura grafica e non sono in ogni modo necessari per definire effettivamente l'autorizzazione da parte di un utente ad interagire con una determinata risorsa. Questo controllo è infatti stato posto backend e non è in alcun modo manipolabile dall'utente da frontend.

Il backend sarà invece costituito da codice JS attraverso l'uso di Node.js e da una base di dati supportata da MongoDB. Il codice andrà a supporto delle funzionalità che prevedono interazione con la base di dati. Fungerà inoltre da middleware per le richieste provenienti dal JS frontend e destinate alle API di Spotify: così facendo le credenziali di autenticazione richieste dalle API di Spotify saranno collocate **lato server**, rendendole **inaccessibili** agli utenti della piattaforma per ragioni di sicurezza.

4.1.1 Base di dati

Segue un'analisi delle informazioni che verranno memorizzate all'interno della base di dati prevista:

Utente	Tipo	Flag
Id	ObjectId	PK
Nome	String	
Cognome	String	
Data di nascita	String	
Email	String	UQ
Username	String	UQ
Password	String	
Generi	Array	
Artisti	Array	
Data iscrizione	String	
Ruolo	String	
Playlist seguite	Array	

Table 1: Con **PK** si intende PRIMARY KEY, con **UQ** UNIQUE e con **FK** FOREIGN KEY.

In questo caso si è optato per utilizzare l'*Id* automaticamente inserito all'interno di un documento da parte di MongoDB come PK del record. Le informazioni personali dell'utente sono memorizzate sotto forma di stringhe. Email e username sono impostate come UNIQUE, in modo tale da impedire a due utenti di possedere il medesimo username. Si è deciso di impostare come UNIQUE anche la email prevedendo la possibilità, da parte dell'utente, di autenticarsi sulla piattaforma utilizzando sia email che username. Successivamente, generi e artisti preferiti sono memorizzati come array di stringhe. Come si vedrà meglio successivamente, queste sono ricevute in linea teorica come array di stringhe direttamente dal backend. Si adottano le dovute accortezze e controlli al fine di verificare che la richiesta non contenga dati manipolati che possano alterare l'integrità del documento. Si prevedono poi alcune informazioni aggiuntive come data iscrizione e ruolo, chiaramente non specificati dall'utente ma impostate in automatico da parte del backend. Infine, la funzionalità richiesta delle condivisioni è stata soddisfatta associando al documento dell'utente un array di ObjectId relativi a delle playlist seguite (strettamente pubbliche).

Playlist	Tipo	Flag
Id	ObjectId	PK
Nome	String	
Descrizione	String	
Tag	Array	
Autore	ObjectId	FK
Brani	Array	
isPublic	Boolean	

Anche nel caso delle playlist queste sono identificate dal relativo *Id*. Le informazioni generali sono memorizzate come stringhe. Medesimo discorso fatto per generi e artisti vale per i tag della playlist: il backend si assicura che l'input ricevuto sia sano, ed eventualmente corregge eventuali errori. Ad esempio, se dovesse essere presente un tag con molteplici #, l'applicativo si occuperà di rimediare lato backend. Si è deciso poi di realizzare l'associazione tra autore e playlist creata attraverso l'*Id* dell'autore migrato nella playlist come FK. La playlist contiene un array di stringhe per tenere traccia dei brani contenuti al loro interno. Le stringhe corrispondono agli *id* dei brani ottenuti dall'API di Spotify. L'interazione tra TuneMate ed API di Spotify sarà largamente approfondita di seguito. Infine, il set di dati si conclude con un booleano volto a tenere traccia della visibilità della playlist. Ho considerato inoltre la possibilità di rendere il nome delle playlist un campo univoco. Tuttavia, ho preferito lasciare agli utenti la possibilità di creare playlist omonime, in linea con il funzionamento di altre vere piattaforme musicali.

Come suggerito dalle funzionalità descritte precedentemente, si può notare come il set di dati non includa campi per

il numero di follower, la durata della playlist, il numero delle canzoni. Si è prediletto infatti il calcolo on demand di queste informazioni pur di non memorizzare a DB valori calcolabili a partire dalle informazioni in esso già contenute.

4.1.2 Pagine web

Segue ora un'analisi della struttura frontend scelta per la realizzazione di TuneMate. L'interfaccia rispecchia le relazioni precedentemente esplicitate all'interno dell'UML use case, al fine di realizzare tutte le funzionalità richieste.

Wireframe TuneMate si presenta con una schermata iniziale di benvenuto. La navbar viene generata dinamicamente in base all'utente visitatore: quando questo è loggato sulla piattaforma, ha accesso ad una navbar differentemente strutturata (e riportata nei successivi schemi).
Con riferimento alle regole grafiche adottate nei seguenti schemi: il testo contenuto in riquadri rappresenta pulsanti che possono sia fungere da pulsante che scatena un evento gestito attraverso JS frontend, sia da link. Il testo sottolineato invece rappresenta un link.

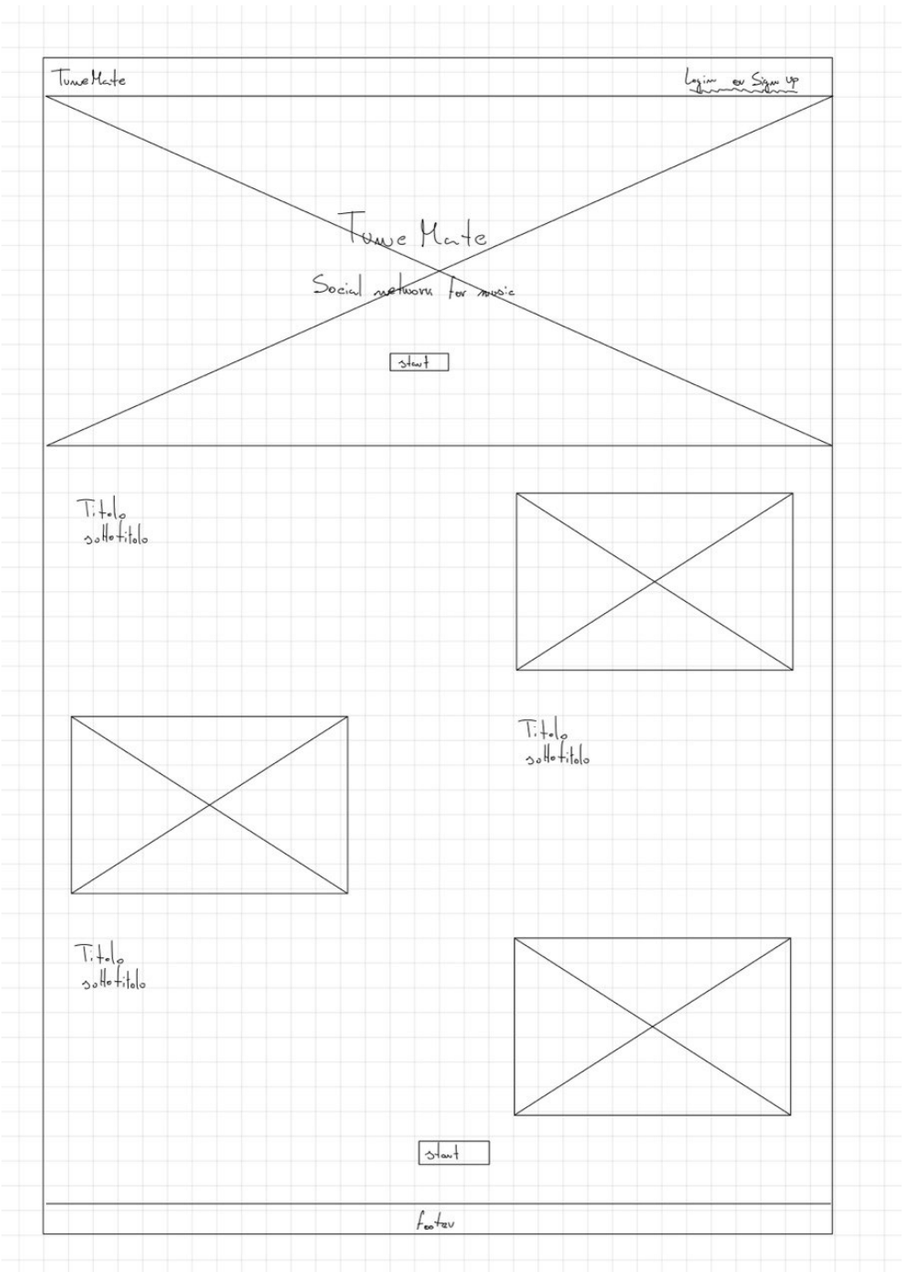

Figure 2: Pagina di benvenuto

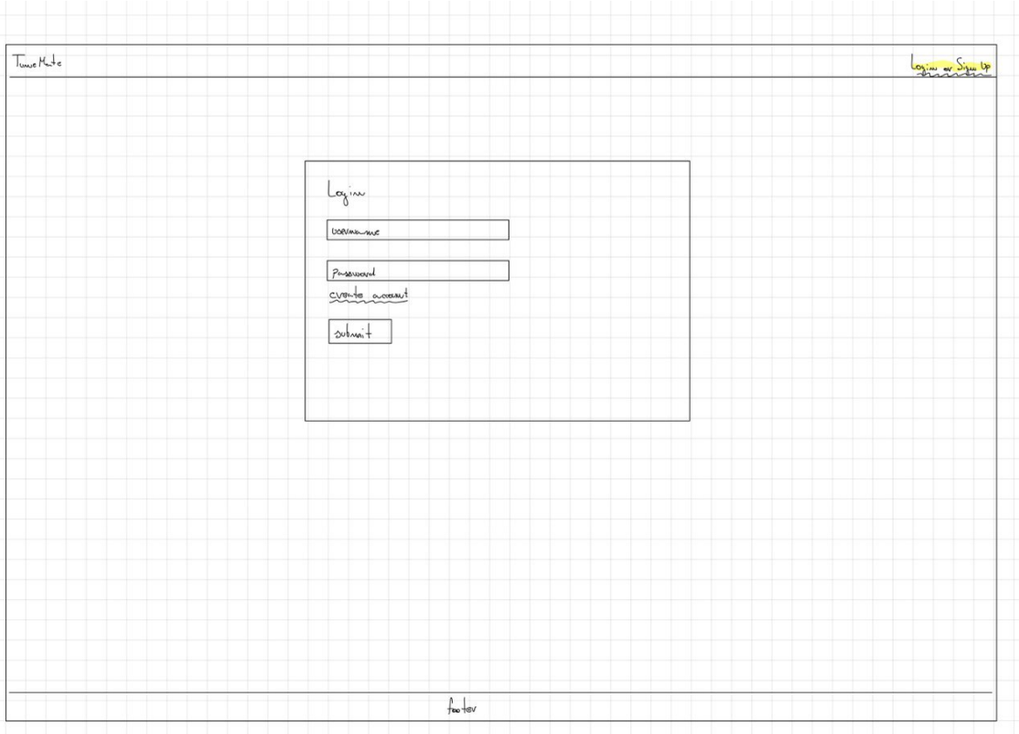

Figure 3: Schermata di login: attraverso la seguente schermata l'utente può effettuare un login. A fronte di credenziali errate o utente non esistente, l'utente sarà notificato con messaggi adeguati.

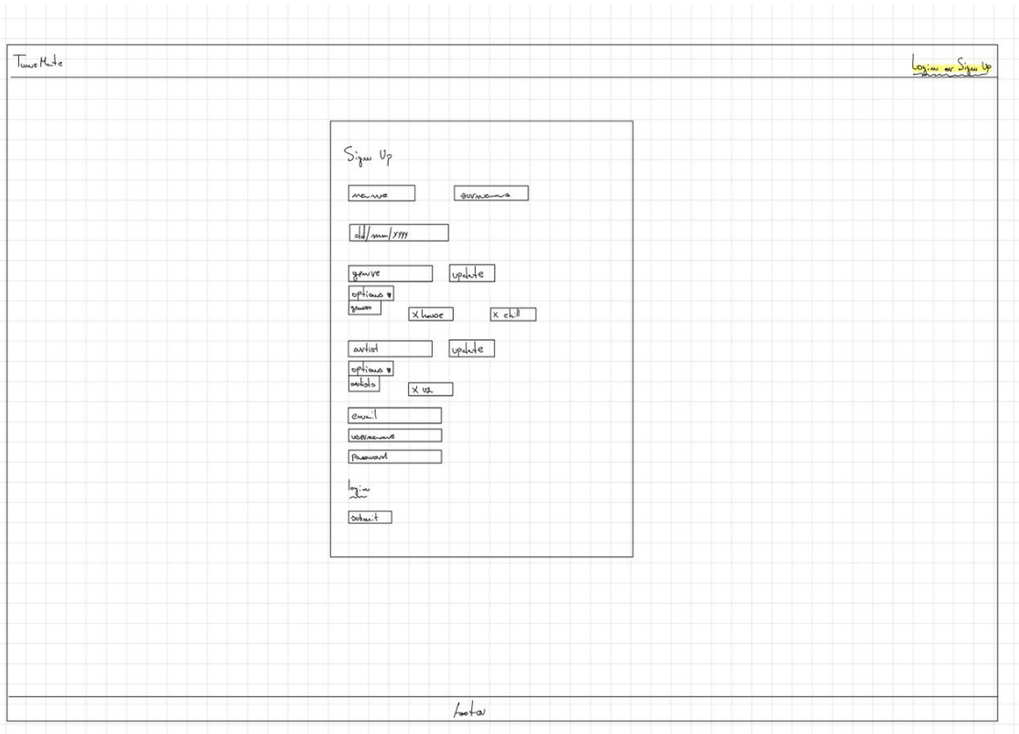

Figure 4: Schermata di registrazione: include un form contenente, tra i campi presenti, dei campi per l'inserimento dei generi e artisti preferiti. L'utente inserisce il nome, clicca il pulsante per aggiornare l'elenco dei risultati e seleziona il più adatto. Questo sarà riportato più in basso e da lì rimuovibile. Tra i requisiti una password che rispetti i criteri di sicurezza, nonché almeno un genere e un artista preferito. Eventuali campi mancanti o malformati saranno adeguatamente notificati.

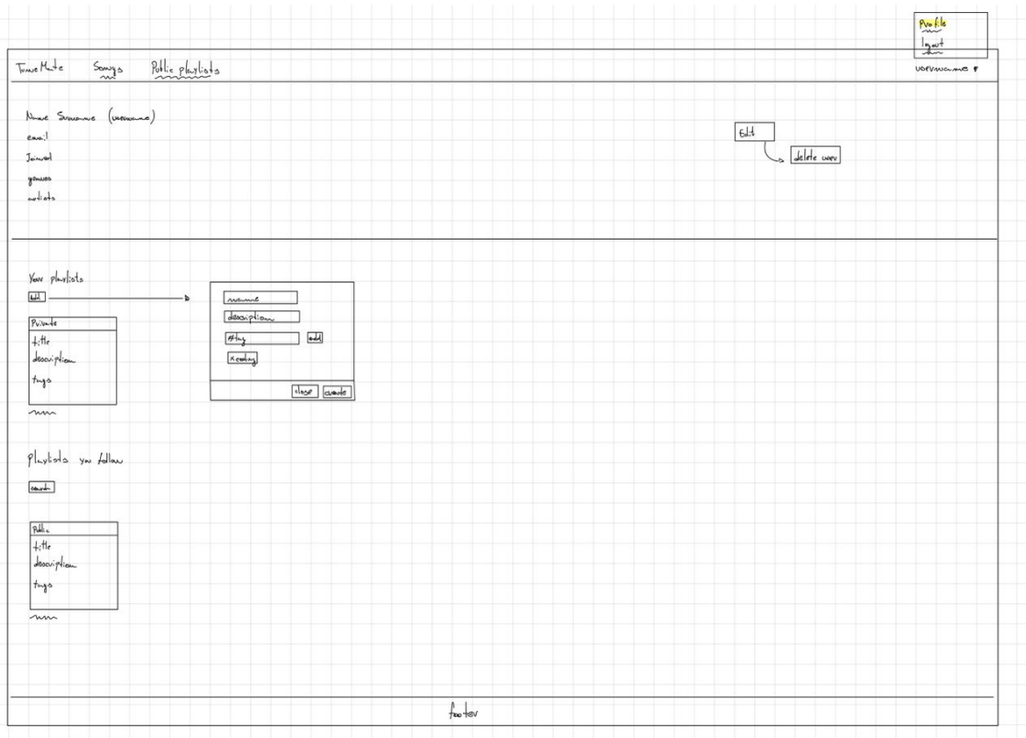

Figure 5: Profilo personale: è possibile consultare e modificare i propri dati personali, nonché cancellare l'account. Modifica ed eliminazione avvengono attraverso un modal che viene sollecitato alla pressione del rispettivo pulsante e rispecchiano il design del form di registrazione, con la caratteristica di essere presentato precompilato con i dati già presenti. Contiene inoltre i riferimenti alle proprie playlist di cui si è autori, nonché le playlist pubbliche seguite. Ogni card rappresenta un link che porta ad una pagina di dettaglio per la playlist visitata. Dalla pagina profilo è inoltre possibile creare una nuova playlist personale, oppure navigare alla pagina di ricerca di playlist pubbliche.

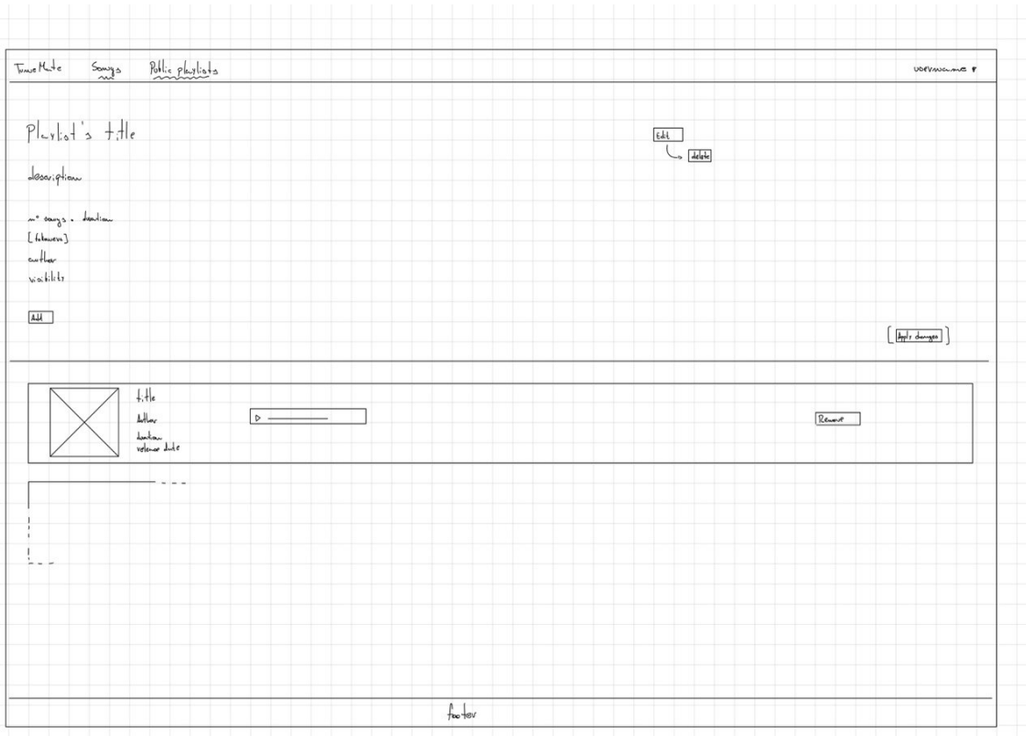

Figure 6: Pagina playlist: pagina relativa ad una specifica playlist. I componenti presenti possono variare in base alla natura della playlist: questa può essere privata e visualizzata dal suo autore, pubblica e visualizzata dal suo autore oppure pubblica e visualizzata da un altro utente. Utenti terzi non possiedono la possibilità di modificare la playlist, ma possiedono un pulsante per diventare follower di queste o per smettere di seguire. Gli elementi tra quadre sono opzionali: ciò significa che possono comparire in particolari scenari. Ad esempio, "*Apply changes*" permette di modificare l'ordine dei brani nella playlist, dopo che questa è stata modificata attraverso un'azione di drag&drop. Tra le informazioni relative alla playlist e riportate nella pagina ci sono informazioni provenienti da DB e informazioni calcolate al momento, come ad esempio la durata complessiva in minuti della playlist. Di seguito, un elenco dei brani inseriti all'interno della playlist. Tra le informazioni riportate, le medesime presenti nella pagina di ricerca brani (segue) tranne per l'elenco dei generi, omessi per non appesantire l'interfaccia e considerati omissibili dal momento in cui il brano è stato già inserito all'interno della playlist, dunque se ne conosce il genere. L'interfaccia offre poi la possibilità all'autore di riordinare i brani, eliminare i brani o aggiungerne di nuovi. L'ultima funzionalità si realizza attraverso la pagina di ricerca brani, raggiungibile dal link *add* in alto a sinistra.

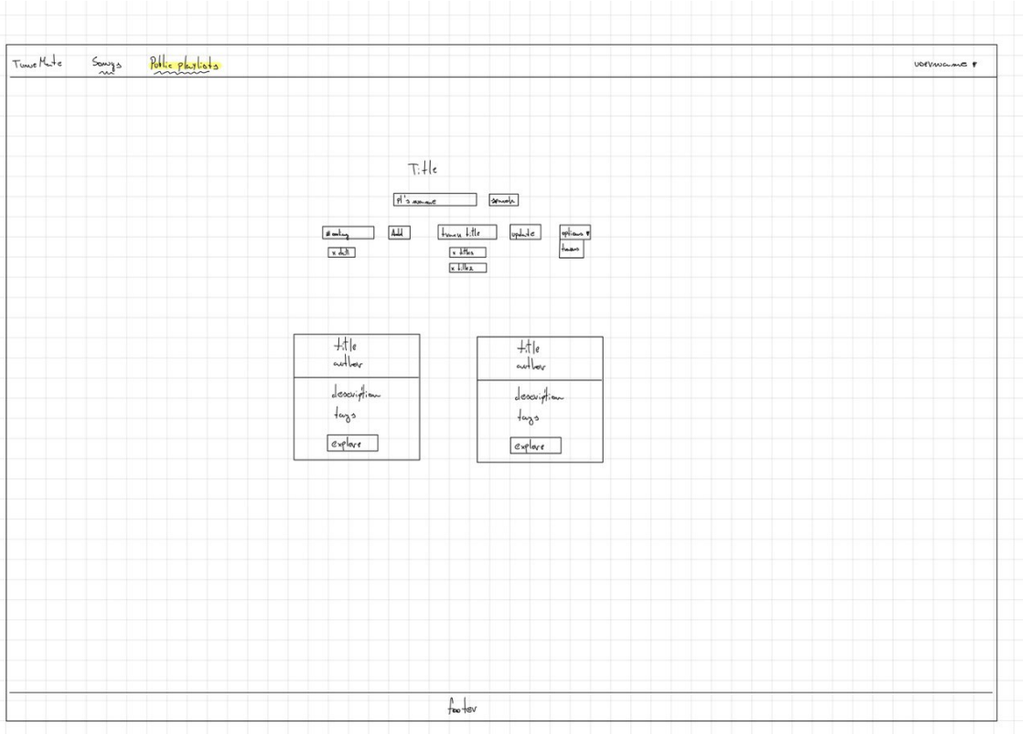

Figure 7: Pagina playlist pubbliche: Ricerca applicando filtri come nome, tag e canzoni incluse. Da qui si può raggiungere la pagina di dettaglio per una specifica playlist e da lì seguirla o meno. Visualizzando l'elenco delle playlist pubbliche, quelle già seguite vengono mostrate con un'interfaccia differente per suggerire la diversa relazione con l'utente loggato.

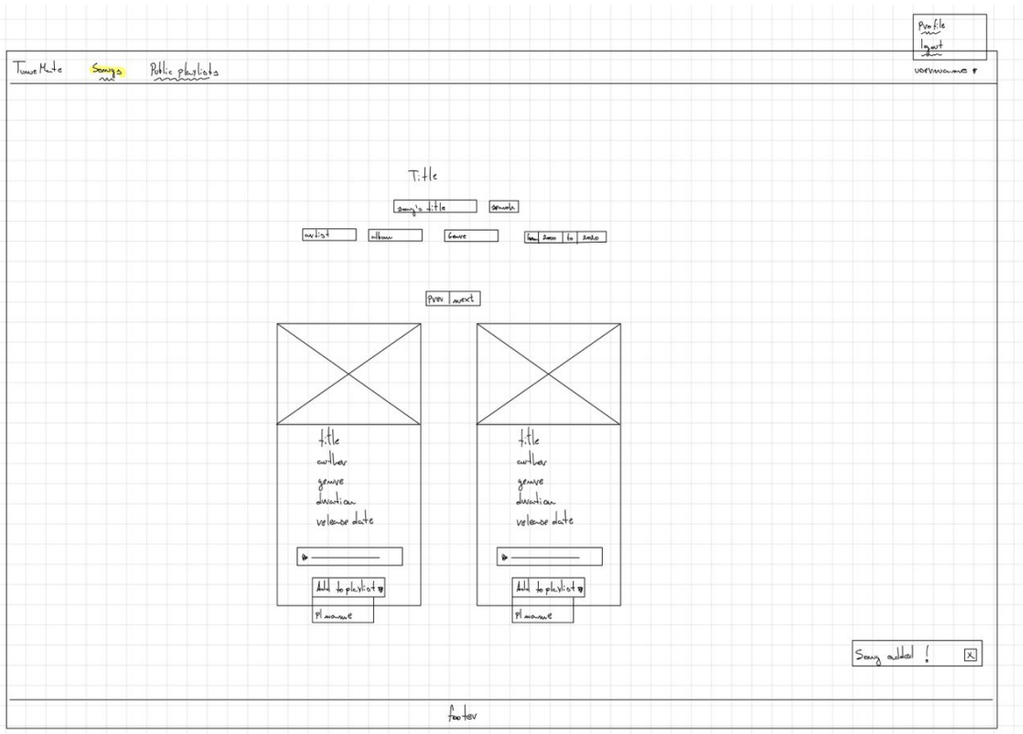

Figure 8: Pagina ricerca brani: tramite questa sezione è possibile ricercare brani attraverso richieste a Spotify e filtrando per nome, artista, album, genere e intervallo di pubblicazione. Da qui è possibile navigare tra i risultati e aggiungere il brano alle playlist di cui si è autori.

Si offre poi uno swagger al fine di documentare in maniera dettagliata la struttura dell'API di TuneMate e le possibili risposte.

4.2 Tecnologie utilizzate

Si ipotizza l'uso delle seguenti tecnologie:

- Lato **frontend**: HTML5, CSS3 (con framework Boostrap 5.3) e JavaScript
  - Si segnala l'utilizzo della libreria CSS e JS esterna AOS (Animate On Scroll library) per la homepage.
  - La feature di drag&drop utilizzata per aggiornare l'ordine dei brani è stata realizzata manualmente e senza l'uso di codice esterno.
  - Per il font utilizzato si ha riferimento esterno a Google Fonts.
- Lato **backend**: **Node.js** con utilizzo di framework come *Express* per l'implementazione delle funzionalità previste; base di dati non relazionale con **MongoDB** per la memorizzazione dei dati utente e playlist. Per questo progetto la base di dati è supportata dall'uso della piattaforma Atlas di MongoDB.

Per quanto riguarda alcune funzionalità come login e sessioni, si ipotizza l'utilizzo di JWT token. Così facendo è possibile gestire con facilità e relativa sicurezza le sessioni utente attraverso cookie e senza stabilire sessioni server-side.

5 Sviluppo

Si propongono ora una serie di considerazioni su alcune scelte degne di nota relativamente alle funzionalità sviluppate. Per alcune di queste sono stati riportati i metodi con cui è possibile operare su determinate risorse, ritenendo necessario chiarire alcune scelte.

5.1 Risorse

Il web server realizzato con *Express* offre risorse statiche e dinamiche. Infatti, il server restituisce automaticamente ogni risorsa statica (HTML, CSS e JS frontend) che dovesse essere richiesta dall'utente. Queste sono collocate in un'apposita directory *public*, che include tutte le risorse pubblicamente accessibili. Per semplicità di realizzazione, le pagine riservate di cui si è realizzata la struttura HTML sono anche loro collocate in questa pagina. Sarà poi compito del frontend assicurarsi dello stato dell'utente attraverso una call a backend ed eventualmente effettuare un redirect alla homepage, essendo pagine che richiedono login. Seppur non si tratti della soluzione più elegante, ciò semplifica l'erogazione delle risorse backend a costo di restituire all'utente non autenticato un template HTML privo di informazioni per cui sarebbe richiesta l'autenticazione.

Le risorse offerte dal web server possono utilizzare due diversi middleware in base alle operazioni svolte per erogare la risorsa in questione.
Un primo middleware è la funzione *auth*, che si assicura che il JWT token ottenuto tramite cookie nell'header della richiesta sia valido. Si è deciso che il token contenga all'interno del suo payload l'id utente (proveniente da MongoDB) e il suo username. Queste informazioni saranno utilizzate in diverse risorse come riferimento dell'utente loggato, per comprendere quali operazioni siano legittime e quali no.
Successivamente, per tutte le risorse che interagiscono con il DBMS di MongoDB si è optato per utilizzare un pacchetto npm che si occupa di sanificare il contenuto della richiesta e mitigare eventuali NoSQL injection. Per approfondire, si veda la sezione **Pacchetti utilizzati**.

Inoltre, è stata fatta la scelta di offrire attraverso Express le risorse delle API di Spotify. Contrariamente a quanto previsto infatti, le *fetch* alle API di Spotify non sono state poste frontend, bensì vengono realizzate da delle apposite risorse di TuneMate, a cui il frontend richiede l'accesso attraverso l'autenticazione già prevista per la piattaforma in sé (JWT token). Questo ha permesso di ottenere innumerevoli vantaggi in fatto di utilizzo delle credenziali di accesso a Spotify e di sicurezza. Gli utenti non entreranno mai a contatto con le effettive credenziali utilizzate per ottenere i token di Spotify e l'accesso a queste risorse sarà regolamentato dall'API di TuneMate.

5.2 Ricerca brani

Per realizzare la ricerca dei brani si fa uso della risorsa **/search** di TuneMate. Dal momento in cui la pagina di ricerca brani richiede anche l'informazione "genere" relativa al brano, a fronte delle informazioni erogate dalle API di Spotify si è optato per la realizzazione di un'ulteriore risorsa, **/artist/:id**, che permettesse di ottenere le informazioni relative ad un artista il cui ID viene passato nel path dell'URL. Questo perché il set di dati di Spotify non prevede una diretta associazione tra brano e genere, ma bensì tra artista e genere. Con ciò si è deciso di riportare nella pagina, in relazione al singolo brano, i generi del primo autore del brano.

5.3 Gestione utenze

Rilevante la gestione delle utenze per quanto riguarda la modifica di username o password, oppure la sua cancellazione. In tutti i casi citati, si è optati per imporre un logout all'utente e la realizzazione di un nuovo login (nel caso della modifica). In tutti i casi lo username dell'utente viene inserito all'interno di una blacklist gestita lato backend. Questa scelta è stata introdotta a fronte dell'utilizzo dei token JWT e dello scenario in cui un utente malevolo tenesse, ad esempio, traccia del proprio JWT token prima di effettuare una modifica di username o cancellazione dell'account. Così facendo sarebbe possibile per lui per al massimo un'ora operare sulla piattaforma a nome di un utente non più esistente. La blacklist viene poi svuotata ogni ora ed il timer fatto ripartire ad ogni operazione che prevede l'inserimento di un elemento all'interno della blacklist. Cruciale risulta optare per memorizzare lo username e non piuttosto il token in sé: in questo scenario infatti un attaccante potrebbe effettuare un primo login, memorizzarsi il token ottenuto per poi rieffettuare un login e sfruttare il secondo token ottenuto per realizzare modifiche utente o cancellazione. In questo modo l'utilizzo del primo token per richieste successive andrebbe a bypassare la blacklist imposta.

Altre operazioni possono essere svolte relativamente ad utenti attraverso la risorsa **/user**, in particolare:

POST: permette di creare una nuova utenza. Riceve nel body le informazioni necessarie, ne verifica la coerenza, le sanifica e procede ad inserire un nuovo record all'interno del DB. Restituisce un errore al client qualora un utente con medesimo username o email dovesse già esistere. Per quanto riguarda generi e artisti, per evitare di appesantire eccessivamente l'applicativo si presuppone che i riferimenti siano relativi a generi e artisti esistenti, senza effettuare un'ulteriore verifica della loro correttezza. Si considera di fatto un input a sua volta precedentemente restituito da backend. Ad esempio, il form che permette di ricercare il nome di un artista per inserirlo tra gli artisti preferiti o il form che permette la ricerca del titolo di un brano per ricercarlo all'interno delle playlist pubbliche, si basa su call a backend per ottenere le possibili opzioni che l'utente potrà successivamente selezionare.

GET: permette di ottenere tutte le informazioni (ad esclusione della password) relative all'utente loggato. Può eventualmente accettare il parametro username per restituire username e id a partire da una sottostringa dello username. Questa funzionalità è stata inserita per realizzare la cessione di proprietà, in cui l'utente inserisce un username e può riceverne l'autocompletamento.

PUT: permette di aggiornare informazioni relative all'utente. In particolare, permette di effettuare una delle seguenti informazioni (a richiesta):

- Modifica informazioni utente (estremi, credenziali...)
- Aggiungere following di una playlist pubblica
- Rimuovere following

Gli ultimi due punti prevedono l'alterazione dell'array di ID memorizzati all'interno del documento dell'utente.
Il backend si occupa di valutare gli input ricevuti e aggiornare solamente quelli considerati validi. I campi eventualmente malformati verranno ignorati e il record non subirà modifiche.

Operazioni sulla login Sono previste poi due operazioni distinte relativamente alla login, attraverso **/login**

GET: viene utilizzato esclusivamente per verificare lo stato dell'utente collegato alla piattaforma. La medesima funzionalità potrebbe essere realizzata con GET /user: tuttavia è risultata necessaria una risorsa la cui consultazione avesse una latenza minima e che trasportasse con sé le informazioni indispensabili, in questo caso lo username per generare dinamicamente la navbar.

POST: per la realizzazione del processo di login. Ricevendo username/email e password, in caso di corrispondenza viene firmato un JWT token e impostato come cookie. Si è inclusa la flag *httpOnly : true*, per mitigare (ma non escludere totalmente) attacchi che potrebbero causare il furto di cookie accedendo a questi tramite *document.cookie*.

5.4 Gestione playlist

La sezione è suddivisa in linea con le risorse relative a playlist:

/playlist operazioni sulle playlist

GET: il backend forgia una risposta attraverso più query al fine di restituire al frontend un unico oggetto contenente informazioni sulle playlist personali e quelle pubbliche seguite dall'utente. Il motivo dietro all'unione delle due macro informazioni all'interno della risorsa nasce dalle seguenti riflessioni:

- Rispetto alla progettazione dell'applicativo, l'utente è principalmente caratterizzato dalle playlist personali e dalle playlist che segue. Inoltre, per necessità implementative, l'inserimento di queste due informazioni sotto la medesima risorsa rendeva particolarmente efficiente il processo di reperimento delle informazioni necessarie.
- Alla luce della struttura del frontend, definire due diverse risorse per ottenere le due informazioni avrebbe previsto due diverse call da frontend, il che avrebbe aumentato la latenza e peggiorato l'esperienza d'uso.
- Dal momento in cui durante la progettazione della base di dati si è optato per definire associazioni migrando gli *_id* dei documenti, è necessario realizzare delle join tra i risultati di molteplici query. Il backend a seguito della GET su /playlist si occupa di ottenere tutte le informazioni necessarie e unirle in un unico oggetto da restituire, in modo tale che il frontend si possa occupare di realizzare una sola richiesta e riempire la pagina con le informazioni ottenute.

/playlist/:id operazioni su una playlist

GET: permette di ottenere informazioni relative ad una specifica playlist. Al fine di restituire un unico oggetto contenente tutte le informazioni necessarie:

- In ogni caso viene eseguita una query a *getseveraltracks* dall'API di spotify per ottenere informazioni sui brani contenuti nella playlist. L'oggetto restituito viene sostituito nell'oggetto finale restituito dalla risorsa.
- Se pubblica, viene aggiunta una specifica sul numero di follower
- Se pubblica e non si è autori, viene aggiunta una specifica sullo stato seguito/non seguito da parte dell'utente loggato.

- In ogni caso, per il medesimo discorso fatto per GET /playlist, anche in questo caso il backend si occupa di forgiare un oggetto finale ed unico da restituire. Ad esempio, per evitare di memorizzare all'interno del DB le effettive informazioni di brani provienienti da Spotify, la playlist è caratterizzata da un array di ID di brani contenuti. Alla richiesta in questione, il backend concatena tutti gli id presenti ed effettua una fetch all'API di Spotify *getseveraltracks* al fine di ottenere informazioni per ogni singola traccia. Successivamente, questo nuovo oggetto viene sostituito all'array di id nell'oggetto da restituire al frontend, in modo tale che questo possa direttamente costruire la pagina senza effettuare ulteriori call a backend.

PUT: permette di aggiornare informazioni relative alla specifica playlist. In particolare, possono essere svolte 4 diverse operazioni (una alla volta) in base ai campi inseriti all'interno del body. In particolare:

- Modifica informazioni playlist (nome, descrizione, tag...)
- Aggiunta brano
- Rimozione brano: come per l'aggiunta, anche per la rimozione si effettua un check sull'esistenza del brano
- Modifica ordine brani: in particolare, questa funzionalità è stata realizzata ricevendo nel body un array di ID, che verranno sostituiti ai già presenti all'interno del record della playlist. Prima di applicare la modifica, gli ID passati vengono utilizzati per contattare l'API di Spotify e verificare che si tratti di brani validi. In caso contrario, la modifica non viene realizzata.

/pubplaylist : operazioni sulle playlist pubbliche

GET: unico metodo previsto, permette di ottenere un elenco di playlist pubbliche, filtrando eventualmente per nome, tag e brani inclusi nelle playlist. Questi parametri opzionali, come secondo la teoria di REST, sono eventualmente passati come parametri in query nell'URL.

5.5 Gestione condivisioni

Una volta che una playlist è stata resa pubblica, un utente può ricercarla tramite apposita sezione. Si è deciso di mostrare tutte le playlist pubbliche, incluse le playlist pubbliche di cui l'utente loggato è autore. L'aggiunta o rimozione di un follow consiste nell'aggiunta o rimozione dell'id della playlist nell'array di playlist seguite dell'autore. Al fine di preservare l'integrità referenziale, ci si è occupati di adeguare i riferimenti nella base di dati nei seguenti scenari:

- Modifica visibilità della playlist: qualora una playlist pubblica dovesse divenire privata, ogni riferimento a questa all'interno delle playlist pubbliche seguite da altri utenti viene rimosso.
- Eliminazione playlist: ogni riferimento alla playlist nelle playlist pubbliche seguite da altri utenti viene rimossa
- Eliminazione dell'intero utente: medesima operazione svolta per l'eliminazione della singola playlist, ma utilizzando l'**array di id di playlist di cui l'utente è autore**

Queste operazioni non nascono dalla necessita di mostrare un "elenco di playlist seguite" corretto, ma bensì per evitare di possedere un elenco di playlist seguite dove alcune di queste non sono più esistenti o accessibili perché diventate private. In ogni caso, anche se queste ulteriori operazioni dovessero non essere presenti nel codice, un utente non sarebbe in grado di accedere a playlist private altrui o visualizzare playlist seguite ed ora divenute private.

5.6 Spotify token

Un'altra problematica è stata identificata nella gestione dei token Spotify. L'API di Spotify richiede l'inserimento di un token all'interno dell'header della richiesta come forma di autenticazione. Il token viene restituito in risposta ad una richiesta contenente al suo interno dei codici identificativi dell'utente richiedente e ha validità di un'ora. Proprio per questo, si è optato per definire nel codice JS backend una funzione che richiede il token e lo assegna ad una variabile globale, nonché per impostare un intervallo di un'ora dopo il quale viene effettuata una callback alla medesima funzione.

5.7 Pacchetti utilizzati

Tra i pacchetti npm utilizzati per lo sviluppo di TuneMate, jsonwebtoken per la gestione dei token JWT; validator, per sanificare gli input dell'utente e verificarne alcune proprietà, ad esempio per definire se una password è sufficientemente forte o se l'input è un indirizzo email valido.

Le informazioni sensibili come il clientId di Spotify o le credenziali per l'accesso al DBMS di MongoDB sono memorizzate in un file *.env*, a cui si accede tramite il pacchetto npm dotenv. Si sottolinea nuovamente come si sia deciso di spostare le richieste all'API di Spotify dal frontend al backend, al fine di impedire agli utenti della piattaforma di accedere a queste credenziali semplicemente navigando tra i file JS trasmessi al client al caricamento delle pagine. Inoltre il file *.env* include anche il secret utilizzato per firmare i token JWT. Affinché l'uso di token JWT sia sensato, il secret **deve** risiedere lato server e non essere mai trasmesso al client. Il file *.env* seguirà quindi la seguente struttura e attraverso il pacchetto il suo contenuto sarà automaticamente importato e accessibile attraverso *process.env.VARNAME*:

```
JWTSECRET = XXX
CLIENTID = XXX
CLIENTSECRET = XXX
MONGOPWD = XXX
```

Per quanto riguarda la sanificazione dell'input al fine di mitigare NoSQL injection, si è fatto uso del pacchetto express-mongo-sanitize, che analizza il contenuto di *req.body*, *req.query* e *req.params* e rimuove eventuali elementi che potrebbero sottoindentere una injection.

Vi sono poi i pacchetti swagger-ui-express per la generazione della pagina contenente le informazioni dello swagger (utilizzando un file *.json* precedentemente generato utilizzato il pacchetto swagger-autogen) e crypto-js per la generazione di digest **SHA-256**.

Vi sono poi chiaramente Express e mongodb per l'erogazione del RESTful service e web server e per l'interazione con il DBMS.

Tra le dipendenze previste, le uniche indicate nel *package.json* come *DevDependencies* sono *nodemon* e *swagger-autogen*, necessarie solamente in fase di sviluppo.

5.8 Edge cases

Sono stati considerati i diversi scenari in cui il backend dovesse erogare una determinata risorsa a fronte di input parziale o mal formato rispetto a quello previsto. Sono stati riconosciuti gli scenari ed in questi casi il backend risponde, in base alla risorsa in questione, con diversi codici di stato HTTP. Tra i più utilizzati si trova 400, per le richieste che non includono tutte le informazioni obbligatorie per processare la richiesta, oppure un campo non sia del tipo previsto ma bensì un oggetto o un array, ad esempio.
Dal momento in cui un eventuale codice relativo alla sicurezza e posto frontend potrebbe in ogni modo essere bypassato, anche i controlli sulla correttezza del formato dei dati ricevuti in input non vengono posti frontend, ma il processo è gestito a backend. Il frontend si occupa dunque di raccogliere i dati e inviarli. Eventuali controlli posti frontend hanno un'esclusiva utilità grafica. Sarà poi il backend a rispondere adeguatamente alla luce dei dati ricevuti.

In linea generale, molte risorse che interagiscono con il DBMS potrebbero restituire uno status code **500** durante la generazione di ObjectId necessari a realizzare le query. Ciò si verifica a fronte di un input manipolato in cui l'*_id* passato non rispetta le specifiche affinché sia possibile creare un nuovo ObjectId.

Per migliore l'usabilità si applica una funzione *sanitize* agli input stringa dell'utente, per rimuovere eventuali spazi in eccesso e ottenere ad esempio risultati più uniformi nelle ricerche.

Per i dati memorizzati a DB si è optato anche per eliminare dall'input alcuni caratteri speciali, che una volta inseriti all'interno della pagina potrebbero causare un comportamento anomalo, come ad esempio $<, >$.

6 Prove di funzionamento

Seguono una sequenza di immagini volte a descrivere l'utilizzo della piattaforma:

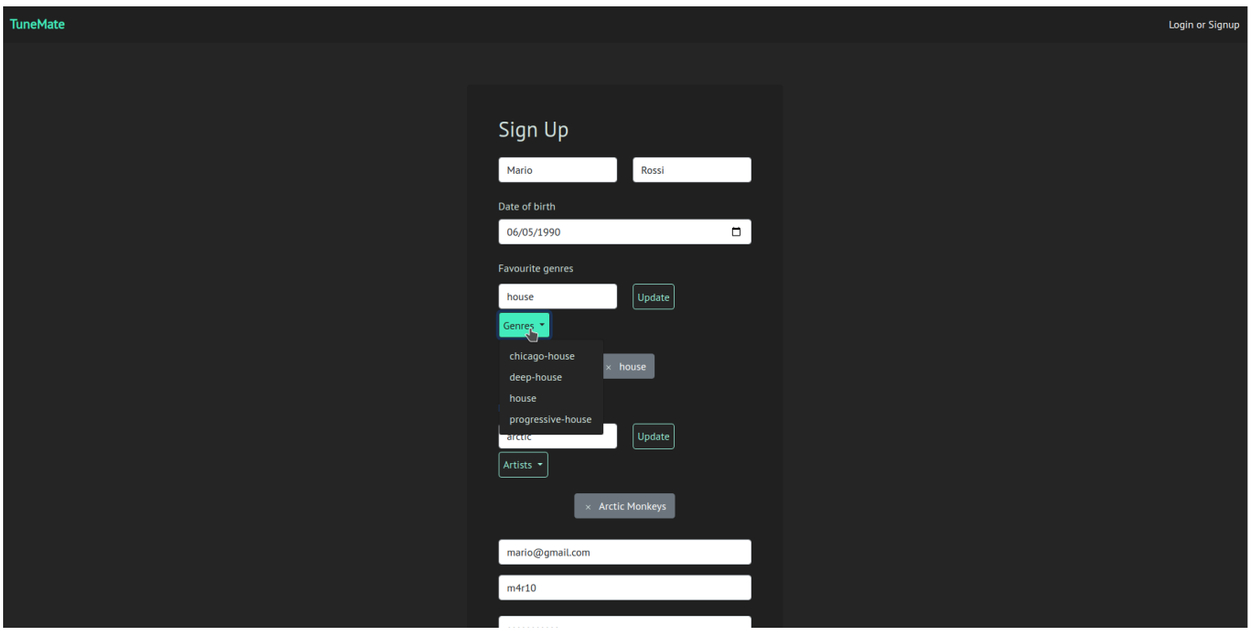

Figure 9: Registrazione dell'utente

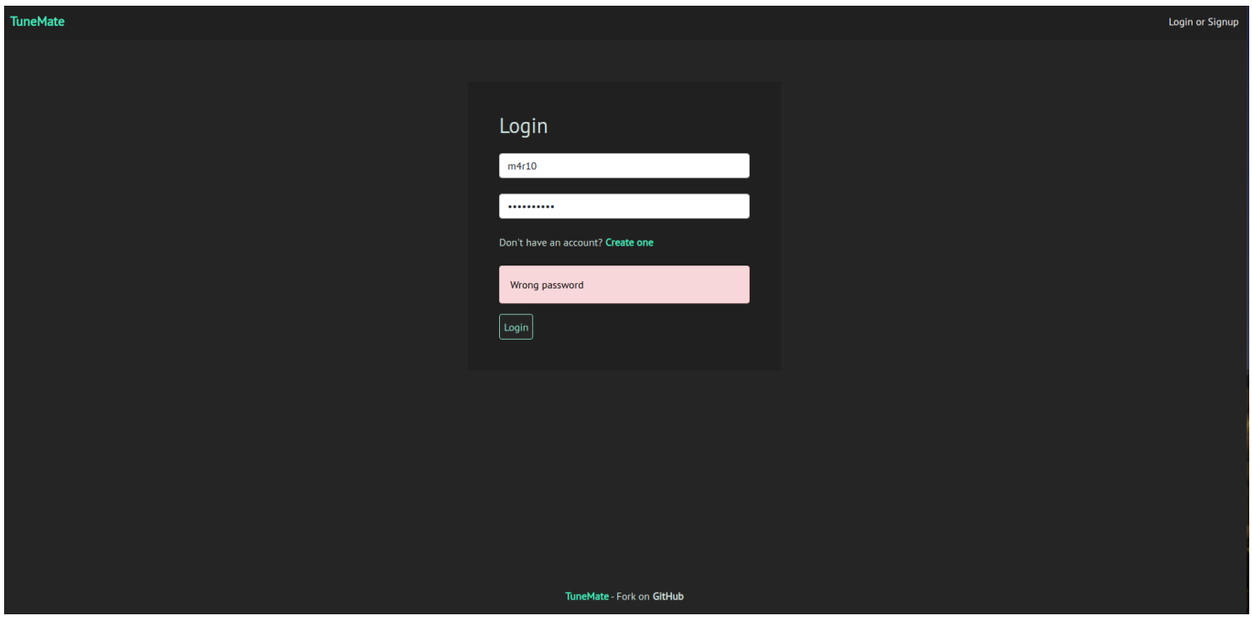

Figure 10: Errore login per credenziali errate

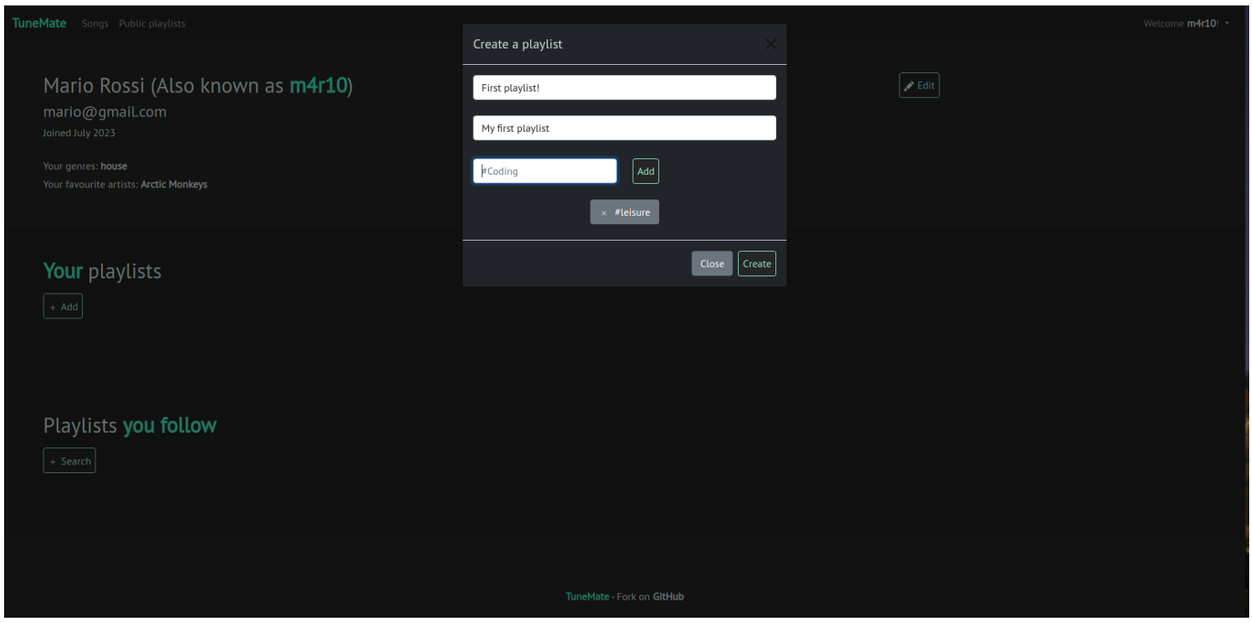

Figure 11: Creazione di una playlist

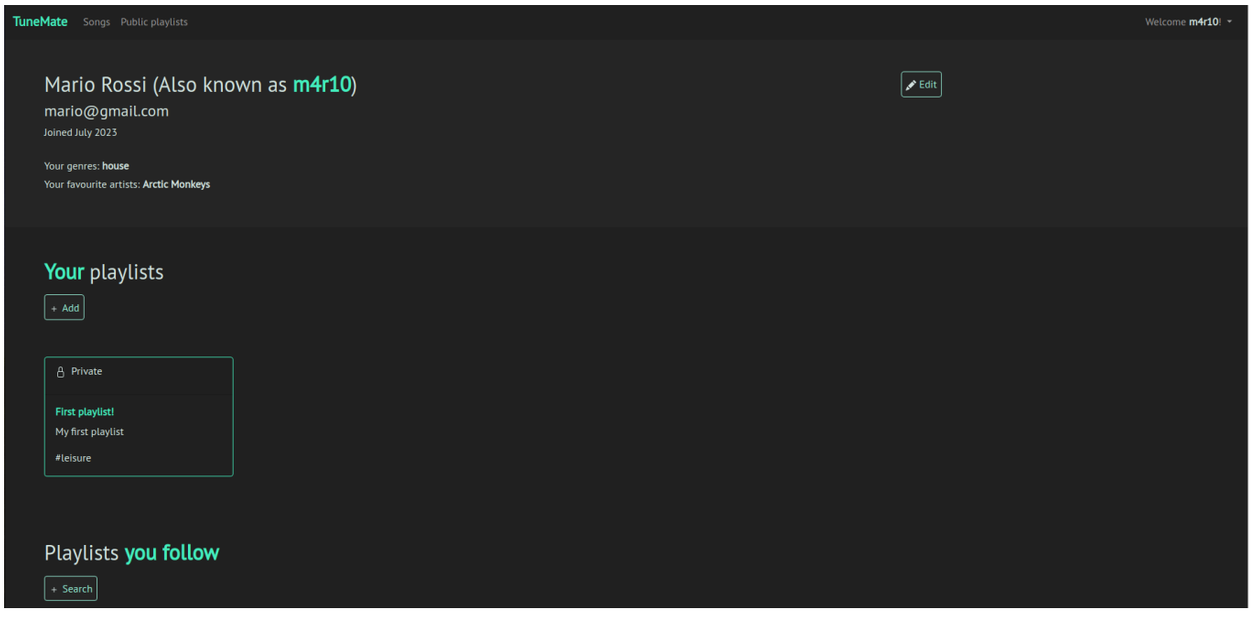

Figure 12: Profilo personale

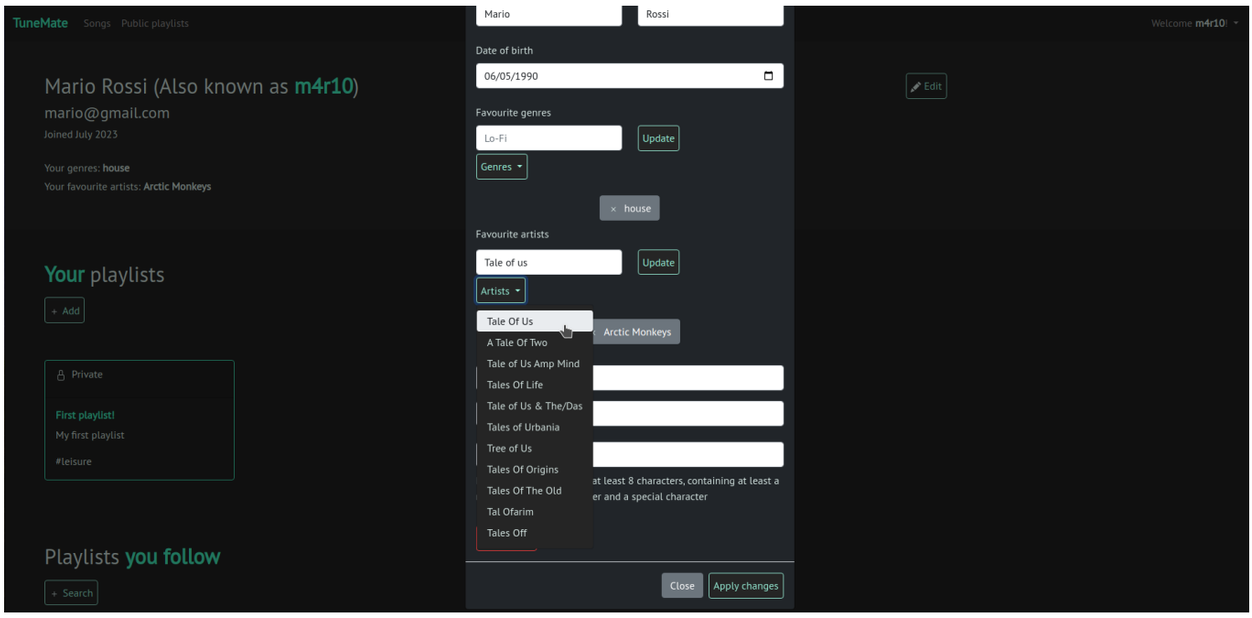

Figure 13: Modifica del profilo

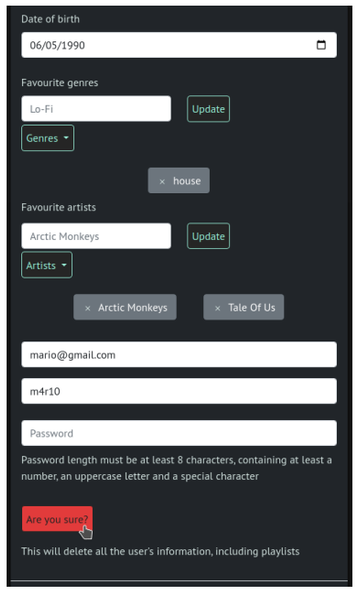

Figure 14: Eliminazione del profilo

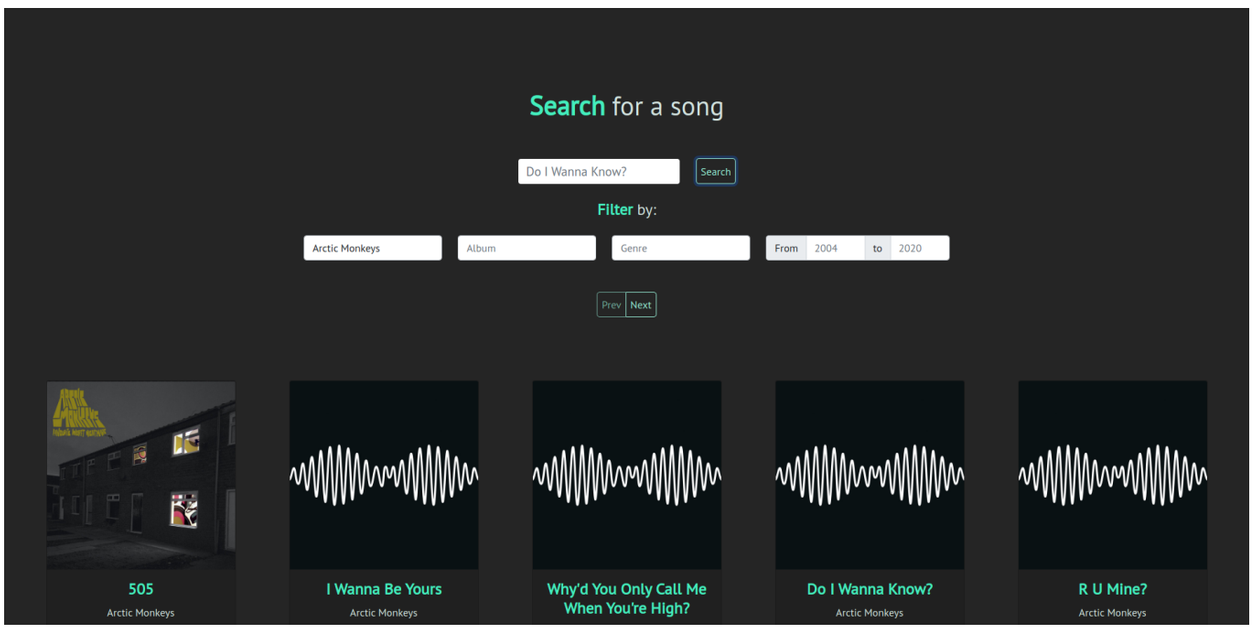

Figure 15: Ricerca canzoni

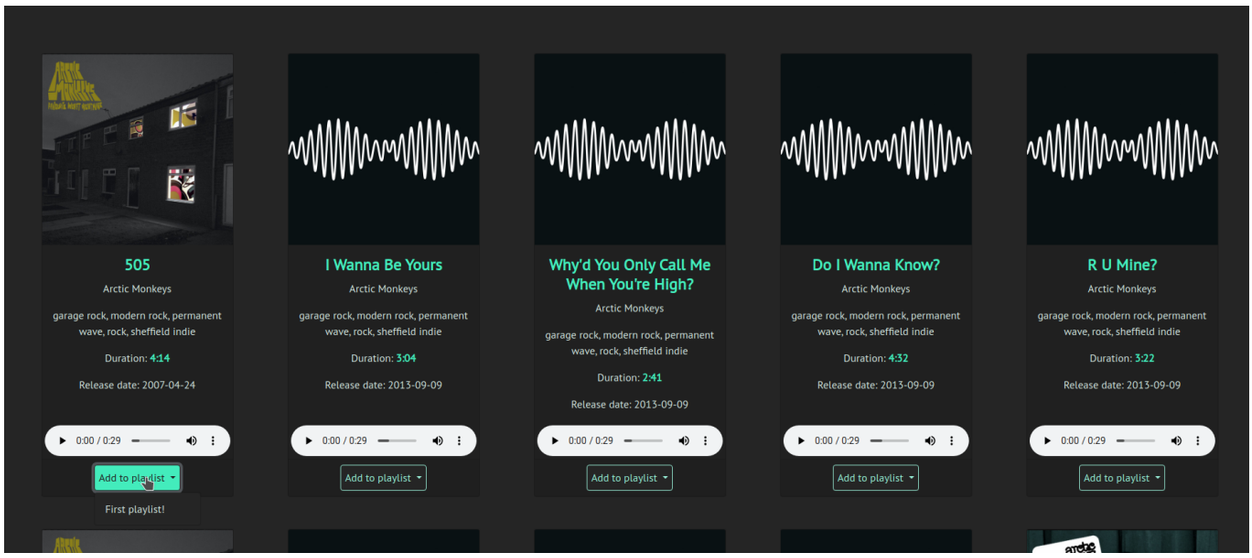

Figure 16: Inserimento canzone in playlist

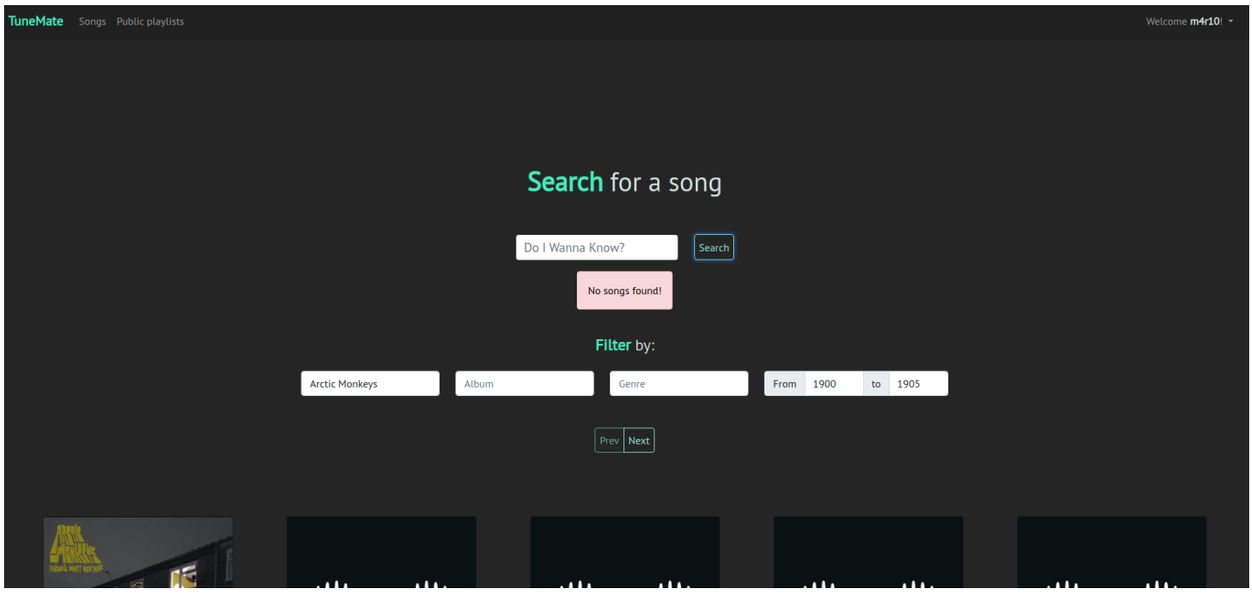

Figure 17: Ricerca canzone (senza risultati)

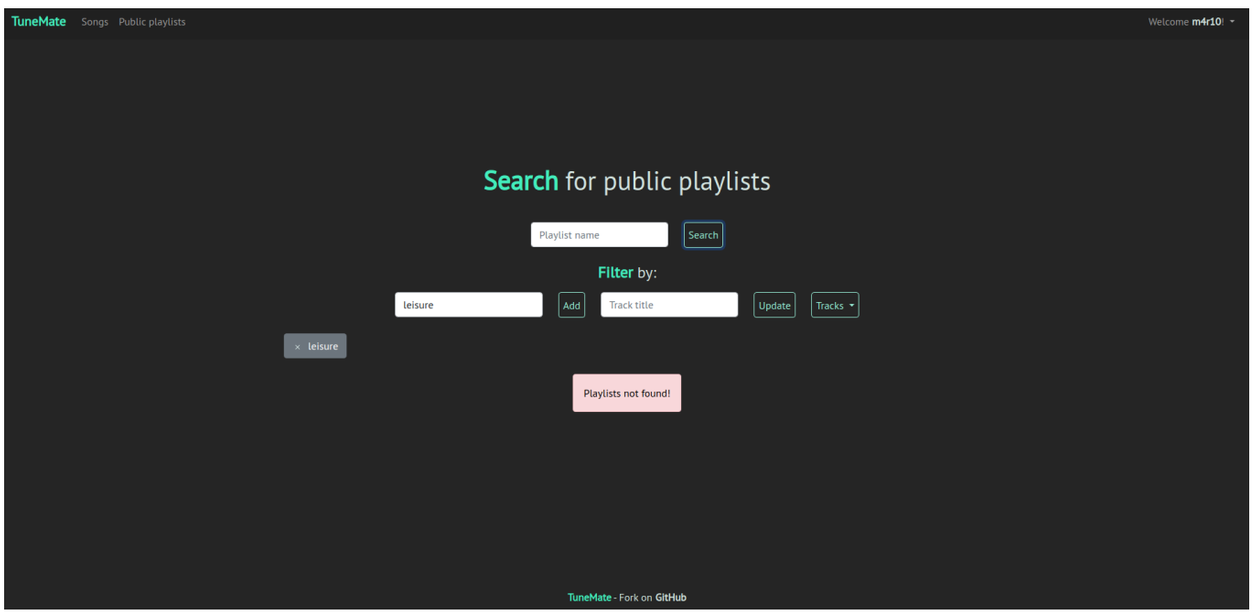

Figure 18: Ricerca playlist pubbliche (senza risultati)

Figure 19: Dettaglio playlist personale

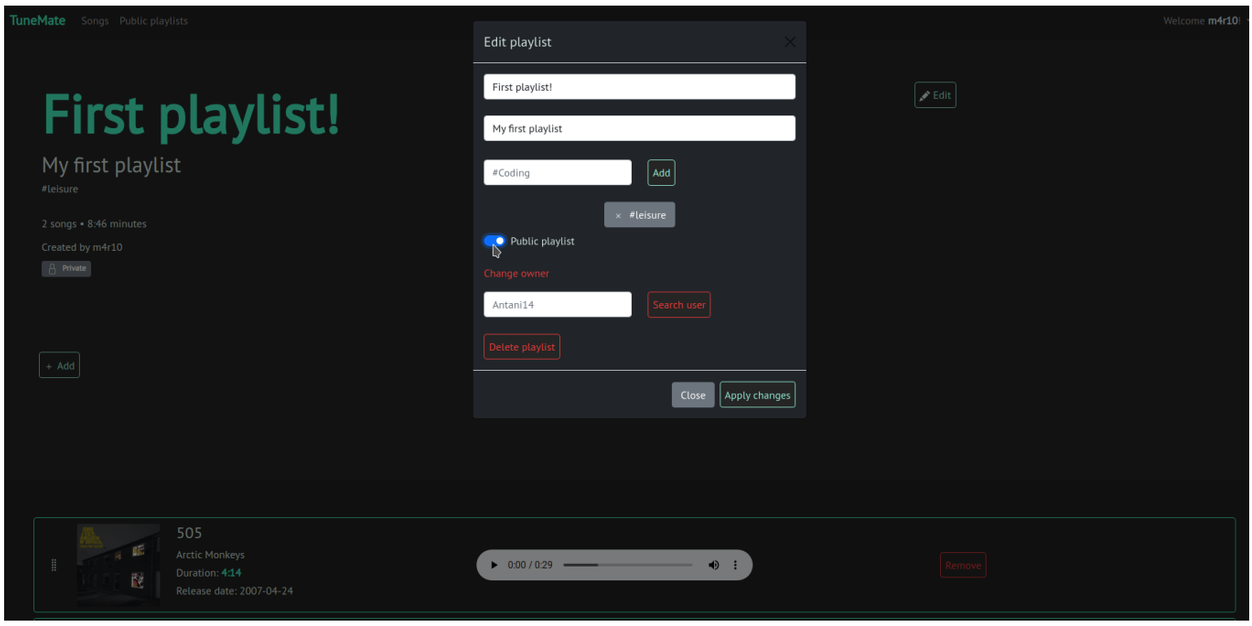

Figure 20: Modifica playlist

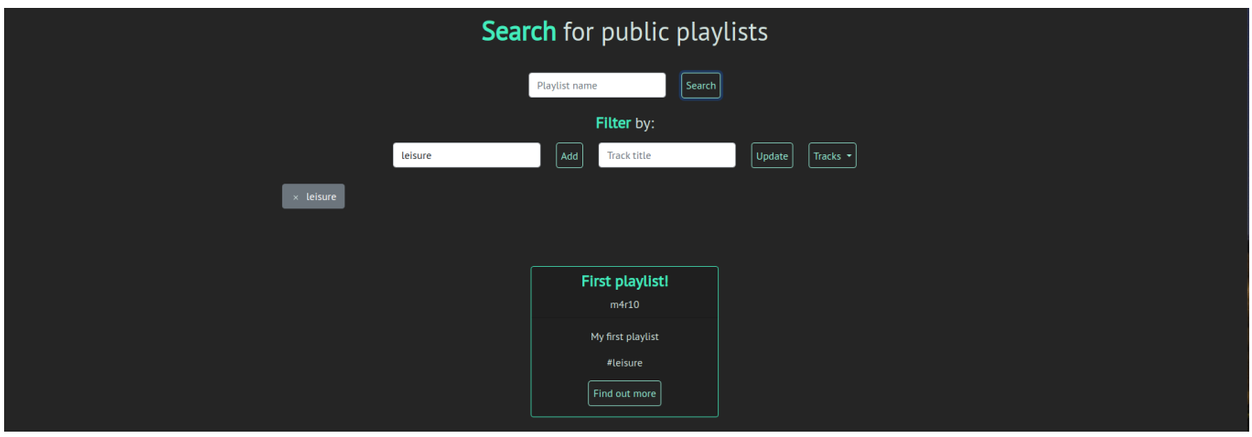

Figure 21: Ricerca playlist pubbliche (con risultato)

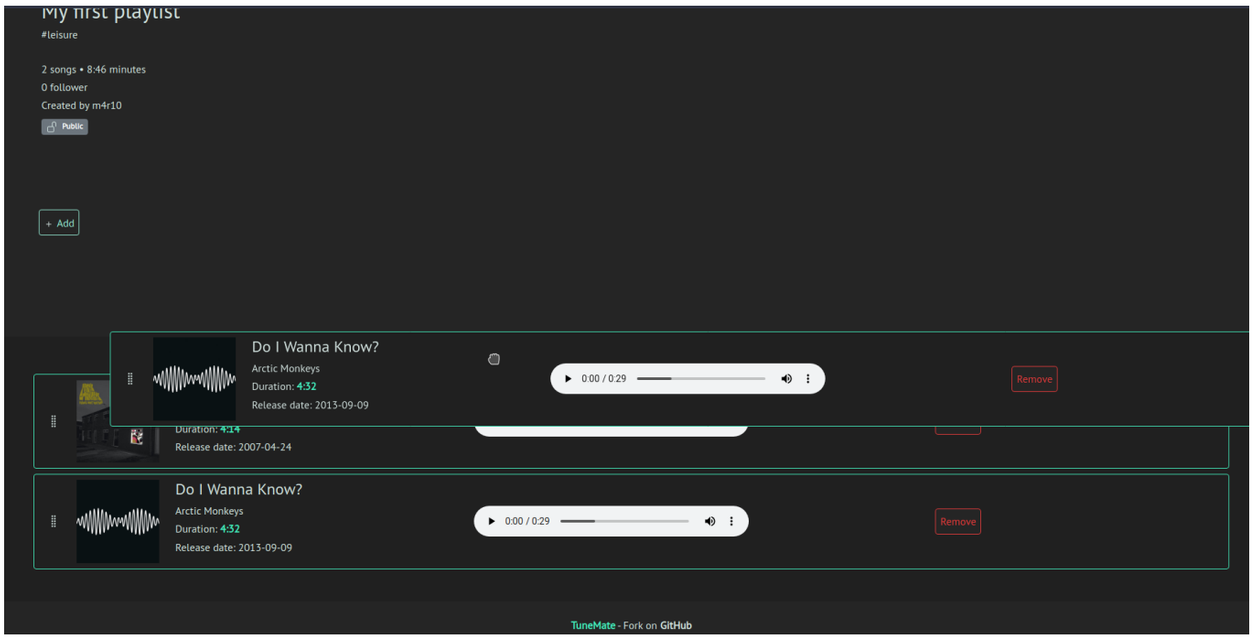

Figure 22: Modifica ordine brani in playlist

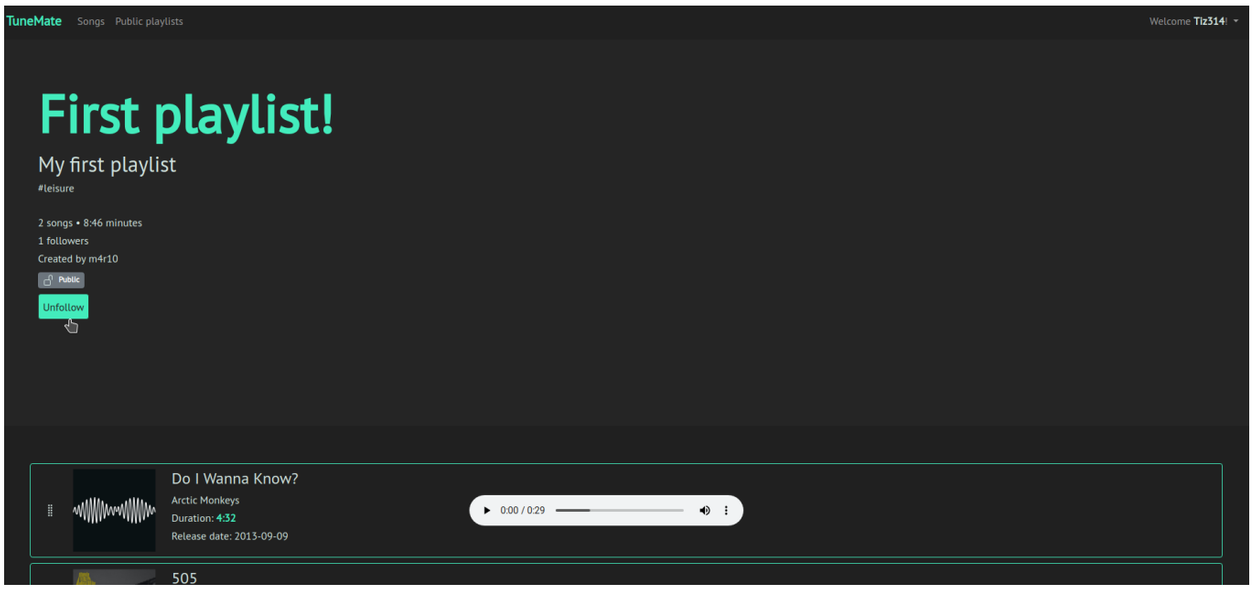

Figure 23: Follow di playlist pubbliche altrui

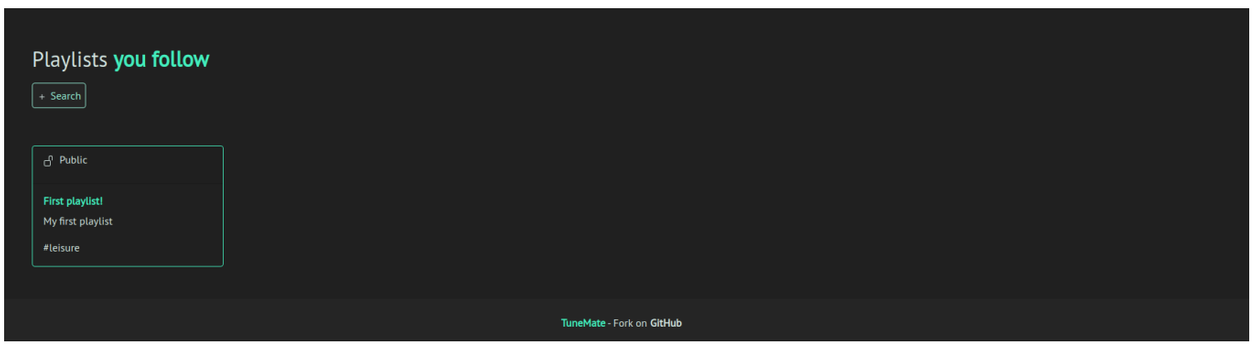

Figure 24: Resoconto playlist pubbliche seguite

7 Conclusioni e sviluppi futuri

Si conclude l'analisi di sviluppo di TuneMate, Social Network for Music.

Gli sviluppi futuri possono prevedere l'implementazione del precedentemente citato utente amministratore, al fine di fornire un ulteriore strumento per chi amministra la piattaforma e moderarne il contenuto.

Si può immaginare anche il caso delle comunità musicali, create da un utente e nelle quali questo può condividere playlist, private o pubbliche (di cui è autore o meno) con altri utenti che partecipano alla comunità.
Si potrebbe ulteriormente immaginare un maggiore sviluppo per quanto riguarda l'interazione tra gli utenti, con una ricerca di utenti e l'aggiunta di following di utenti e non più solamente di playlist pubbliche.