SN4M - Social Network for Music

Relazione tecnica del progetto

Giulia Martina Colombo (42601A)
Applicazioni Web e Cloud - A.A. 2024/2025

INTRODUZIONE

Il progetto **Social Network for Music** (**SN4M**) ha come obiettivo lo sviluppo di un'applicazione web per la gestione di playlist musicali e l'organizzazione di community virtuali dedicate agli appassionati di musica.
L'applicazione consente agli utenti di creare, organizzare e condividere contenuti musicali, favorendo l'interazione tra persone accomunate da interessi e gusti musicali simili.

ANALISI DEI REQUISITI

L'obiettivo del progetto è lo sviluppo di una web application denominata **Social Network for Music (SN4M)**, finalizzata alla **gestione di playlist musicali** e alla **creazione di comunità virtuali** per utenti appassionati di musica.
L'applicazione consente agli utenti di **organizzare**, **condividere** e **ricercare** playlist musicali, sfruttando informazioni provenienti da sorgenti esterne tramite **API REST**, in particolare le **API di Spotify**.
.
Il sistema prevede come unico attore principale l'**utente registrato**, il quale può accedere alla piattaforma tramite un processo di **registrazione** e **autenticazione**. Durante la fase di registrazione vengono raccolte informazioni personali quali nome utente, indirizzo email, password e preferenze musicali, che devono poter essere modificate o cancellate dall'utente stesso.

Dal punto di vista funzionale, l'applicazione deve supportare tre macro-scenari principali:

• Il primo riguarda la **gestione delle playlist**, permettendo agli utenti di creare, modificare ed eliminare playlist musicali private. Ogni playlist è composta da un insieme di canzoni, le cui informazioni principali (titolo, artista, genere, durata e anno di pubblicazione) vengono recuperate tramite le API REST di Spotify. Per ogni playlist l'utente deve poter inserire una descrizione testuale e uno o più tag descrittivi.

• Il secondo macro-scenario riguarda la **gestione delle comunità musicali**. Gli utenti devono poter creare nuove comunità, specificandone titolo, descrizione e tag, oppure ricercare comunità esistenti e decidere di unirsi ad esse.

• Il terzo macro-scenario concerne la **gestione delle condivisioni**. Gli utenti possono condividere le proprie playlist all'interno delle comunità di cui fanno parte. Le playlist condivise risultano visibili esclusivamente agli altri membri della comunità, che possono visualizzarne le informazioni principali ed eventualmente importarle nel proprio profilo. È inoltre richiesta la ricerca delle playlist condivise sulla base dei tag associati o delle canzoni contenute.

Dal punto di vista non funzionale, l'applicazione deve essere sviluppata utilizzando **HTML5**, **CSS3** e **JavaScript**, rispettando il principio di separazione tra struttura e presentazione delle pagine web. I dati devono essere memorizzati localmente tramite Web Storage del browser, in formato **JSON** o **XML**, senza l'utilizzo di un **database server-side**.
Tali requisiti costituiscono la base per le successive fasi di progettazione e implementazione del sistema.

Quindi gli obiettivi sono:

- Sviluppare un'applicazione web per la **gestione di playlist** musicali
- Consentire la **creazione** e la **gestione di community** musicali tematiche
- Permettere la **condivisione di playlist** all'interno delle community
- Offrire **strumenti di ricerca** e **visualizzazione** dei contenuti musicali
- Garantire una **navigazione semplice** e **intuitiva** per l'utente

FUNZIONALITA' DA SVILUPPARE

GESTIONE UTENTI

III sistema prevede come unico attore principale l'utente registrato, che può:

- **registrarsi alla piattaforma;**
- **autenticarsi tramite login;**
- **gestire i propri dati personali;**
- **interagire con playlist e comunità musicali.**

Durante la fase di registrazione vengono raccolte informazioni personali, tra cui:

- **nome utente;**
- **indirizzo email;**
- **password;**
- **preferenze musicali**

Tali informazioni devono poter essere modificate o cancellate dall'utente.

GESTIONE PLAYLIST

Il sistema deve consentire agli utenti di:

- **creare playlist musicali private;**
- **modificare ed eliminare playlist esistenti;**
- **aggiungere canzoni alle playlist tramite le API REST di Spotify;**
- **visualizzare per ogni canzone le informazioni principali**, quali:
  - **titolo**,
  - **artista**,
  - **genere**,
  - **durata**,
  - **anno di pubblicazione**;
- **associare a ogni playlist una descrizione testuale e uno o più tag descrittivi.**

GESTIONE COMMUNITY

Il sistema deve permettere agli utenti di:

- **creare comunità musicali, specificando:**
  - **titolo,**
  - **descrizione,**
  - **tag associati;**
- **ricercare comunità esistenti;**
- **unirsi a una o più comunità.**

CONDIVISIONE

Il sistema deve consentire:

- **la condivisione delle playlist all'interno delle comunità di cui l'utente fa parte;**
- **la visualizzazione delle playlist condivise dagli altri membri;**
- **l'accesso alle informazioni principali delle playlist condivise;**
- **la ricerca delle playlist condivise in base a:**
  - **tag associati,**
  - **canzoni contenute;**
- **l'eventuale importazione delle playlist nel proprio profilo.**

VINCOLI

Dal punto di vista tecnologico e progettuale:

- l'applicazione deve essere sviluppata utilizzando **HTML5**, **CSS3** e **JavaScript**;
- deve essere rispettata la **separazione** tra struttura (HTML) e presentazione (CSS);
- i dati devono essere memorizzati localmente tramite **Web Storage** del browser;
- il formato di memorizzazione deve essere **JSON** o **XML**;
- non è previsto l'uso di un database server-side.

PROGETTAZIONE STRUTTURA

STRUTTURA GENERALE DEL SITO

L'applicazione web **SN4M (Social Network for Music)** è organizzata in un insieme di pagine distinte, ognuna progettata per supportare una specifica fase dell'interazione dell'utente con il sistema. Questa suddivisione consente una chiara separazione delle funzionalità e favorisce una navigazione semplice e intuitiva.

Le principali pagine che compongono il sito sono le seguenti:

- **Landing page**, che rappresenta il punto di ingresso dell'applicazione e fornisce una presentazione generale del servizio, invitando l'utente a effettuare il login o la registrazione;
- **Pagina di registrazione**, dedicata alla creazione di un nuovo account utente tramite l'inserimento delle informazioni richieste;
- **Pagina di login**, che consente agli utenti registrati di autenticarsi e accedere all'applicazione;
- **Pagina di recupero della password**, che permette all'utente di avviare la procedura di recupero delle credenziali in caso di password dimenticata;
- **Home page**, che funge da pagina principale dell'applicazione dopo l'accesso e offre una panoramica delle funzionalità disponibili, permettendo un accesso rapido alle sezioni principali;
- **Pagina delle playlist**, dedicata alla gestione delle playlist musicali dell'utente, incluse le operazioni di creazione, visualizzazione, modifica ed eliminazione;
- **Pagina delle community**, che consente la ricerca, la creazione e la gestione delle comunità musicali, oltre alla visualizzazione delle playlist condivise dagli utenti appartenenti alla stessa community;
- **Pagina del profilo**, che permette all'utente di visualizzare e modificare le proprie informazioni personali e le preferenze musicali, nonché di gestire le impostazioni dell'account.

Questa organizzazione modulare delle pagine rende l'applicazione facilmente estendibile e migliora l'esperienza d'uso, garantendo coerenza funzionale e chiarezza nella navigazione tra le diverse sezioni del sito.

SCELTE ARTISTICHE E VISIVE

Le scelte grafiche del sito web sono state orientate a trasmettere un'**identità visiva coerente con il tema musicale** e con l'idea di community alla base di SN4M.
È stato adottato uno stile visivo **moderno**, caratterizzato da **colori scuri** e **accenti cromatici vivaci**, in modo da evocare l'estetica delle piattaforme musicali digitali senza risultare eccessivamente decorativo.

In fase di progettazione visiva, sono state prese come riferimento alcune delle principali piattaforme di streaming musicale, come Spotify e Apple Music, che fanno largo uso di palette cromatiche scure abbinate a colori di accento ben riconoscibili.
L'analisi di queste interfacce ha permesso di **individuare pattern visivi efficaci**, in particolare l'utilizzo del colore in modo mirato per guidare l'attenzione dell'utente e valorizzare gli elementi interattivi, mantenendo al contempo un'elevata leggibilità dei contenuti. L'uso del colore è stato limitato a pochi elementi chiave, così da:

- mantenere un **buon contrasto** tra testo e sfondo;
- guidare l'attenzione dell'utente verso le **call-to-action**;
- garantire una **lettura confortevole** anche su schermi di dimensioni ridotte.

Gli elementi decorativi presenti nella sezione principale non hanno una funzione puramente estetica, ma contribuiscono a rafforzare l'identità del progetto e a rendere la pagina riconoscibile, senza compromettere la chiarezza dei contenuti.
La scelta di animazioni leggere e continue è stata calibrata per aggiungere dinamismo all'interfaccia, evitando però distrazioni o effetti invasivi, e mantenendo l'attenzione focalizzata sulle funzionalità principali dell'applicazione.

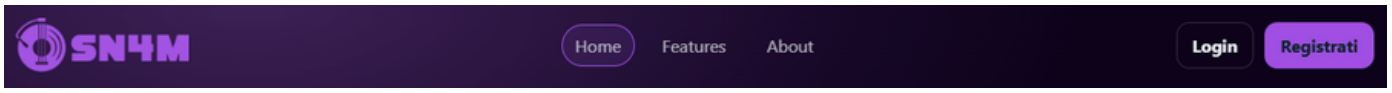
SPOTIFY | **APPLE MUSIC** | **SOUND CLOUD** | **SN4M**

STRUTTURA DELLE SINGOLE PAGINE

La landing page è progettata come pagina introduttiva dell'applicazione ed è rivolta agli utenti non autenticati.
La sua struttura è composta da:

- **Header superiore**, contenente il logo dell'applicazione e i collegamenti alle pagine di login e registrazione;
- **Sezione hero**, che presenta il messaggio principale del servizio, accompagnato da elementi grafici a tema musicale e da call-to-action ben visibili;
- **Sezione delle funzionalità principali**, organizzata in card informative che introducono le caratteristiche fondamentali della piattaforma.
- **Footer**, che conclude la pagina fornendo informazioni di contesto e riferimenti al progetto.

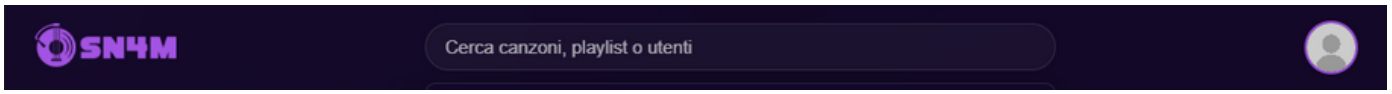
Questa struttura ha l'obiettivo di comunicare in modo immediato l'identità del progetto e incentivare l'utente all'accesso o alla registrazione.

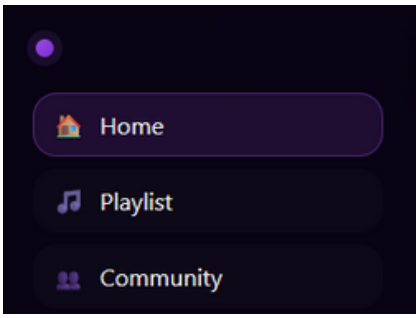


Le pagine accessibili dopo l'autenticazione (home page, playlist, community e profilo) condividono una struttura comune, al fine di garantire coerenza visiva e facilità di navigazione.
In particolare, esse presentano:

- **Sidebar laterale**, utilizzata per la navigazione tra le principali sezioni dell'applicazione;
- **Header superiore**, che include la barra di ricerca globale e l'accesso al profilo utente;
- **Area centrale dei contenuti**, il cui contenuto varia in base alla pagina visualizzata (panoramica, gestione playlist, community, informazioni del profilo).

L'adozione di una struttura condivisa consente all'utente di orientarsi facilmente all'interno dell'applicazione e di riconoscere rapidamente le funzionalità disponibili.





IMPLEMENTAZIONE DELL'APPLICAZIONE

LANDING PAGE

INTRODUZIONE

La **landing page** del progetto **SN4M – Social Network for Music** rappresenta il primo punto di contatto tra l'utente e l'applicazione web. Questo tipo di pagina non svolge un ruolo esclusivamente estetico, ma assume una funzione centrale dal punto di vista comunicativo, funzionale e progettuale. Attraverso la landing page l'utente deve essere in grado di **comprendere rapidamente lo scopo della piattaforma**, le **principali funzionalità** offerte e le **azioni** che può intraprendere, come la registrazione o l'accesso al sistema.

STRUTTURA GENERALE DEL DOCUMENTO HTML

Il file **landing_page.html** definisce la struttura complessiva della pagina. La prima scelta progettuale rilevante riguarda l'organizzazione gerarchica dei contenuti, che sono racchiusi all'interno di due contenitori principali:

```
<body>
  <div class="page">
    <div class="shell">

      <header class="nav">...</header>
      <main id="top" class="hero">...</main>
      <section id="features" class="section">...</section>
      <section id="about" class="section">...</section>
      <footer class="footer">...</footer>

    </div>
  </div>
</body>
```

Il contenitore **.page** ha il compito di garantire che la pagina occupi sempre almeno l'intera altezza della viewport, evitando spazi vuoti su schermi di grandi dimensioni. Il contenitore **.shell**, invece, svolge un ruolo più grafico e strutturale: al suo interno vengono applicati lo sfondo con gradienti, i padding interni e gli effetti visivi che caratterizzano l'intera landing page. Questa separazione consente di isolare le responsabilità dei vari elementi e rende il layout più facilmente estendibile in futuro.

L'utilizzo di tag semantici come **<header>**, **<main>**, **<section>** e **<footer>** non è casuale, ma risponde all'esigenza di migliorare l'accessibilità e la leggibilità del codice.

GESTIONE DELLO STILE GLOBALE E DELLE VARIABILI CSS

Nel file **landing_page.css**, uno degli aspetti centrali è l'uso delle variabili CSS, definite all'interno del selettore **:root:**

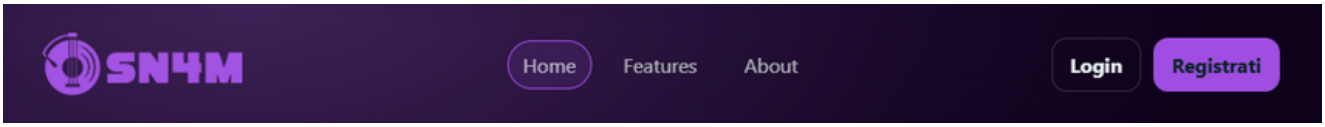
```
:root{
  --card: #0e031b;
  --card-2: #0e0118;

  --text: #e9f1f1;
  --muted: rgba(233,241,241,.72);

  --accent: #a04fe2;
  --accent-2: #6c27a5;

  --radius: 28px;
}
```

Questa scelta permette di **centralizzare la gestione dei colori** e delle **principali caratteristiche visive dell'interfaccia**. In questo modo, un'eventuale modifica al tema grafico della pagina può essere effettuata intervenendo in un unico punto del codice, senza dover cercare e sostituire manualmente i valori all'interno di tutto il foglio di stile. Inoltre, l'uso delle **variabili** migliora la coerenza visiva e riduce la probabilità di introdurre incongruenze cromatiche.

BARRA DI NAVIGAZIONE: STRUTTURA E LAYOUT

La barra di navigazione è definita all'interno di un elemento **<header>** ed è strutturata in tre parti principali:

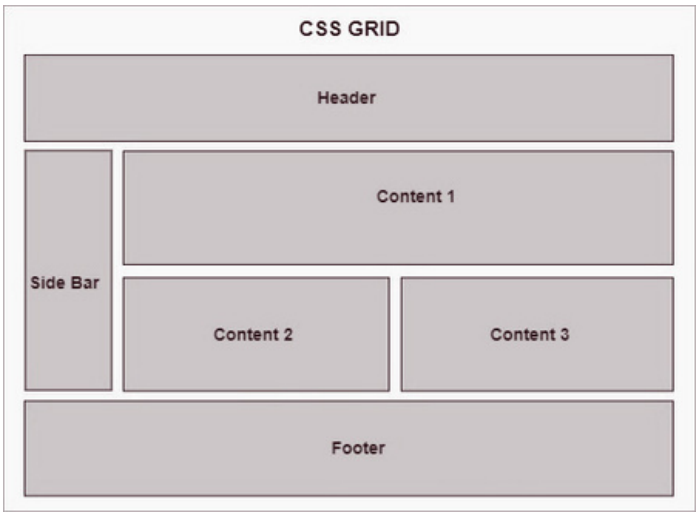
- il **logo** del progetto
- il **menù** di navigazione interna
- l'area delle azioni principali (**login** e **registrazione**).




```
<header class="nav">
  <a class="brand" href="#top">
    <img src="media/logo.png" class="brand-logo" />
  </a>

  <nav class="menu">
    <a class="navlink is-active" href="#top">Home</a>
    <a class="navlink" href="#features">Features</a>
    <a class="navlink" href="#about">About</a>
  </nav>

  <div class="actions">
    <a class="btn btn-ghost"
href="login.html">Login</a>
    <a class="btn btn-primary"
href="register.html">Registrati</a>
  </div>
</header>
```

Dal punto di vista del layout, la navbar utilizza **CSS Grid**, una scelta che consente di ottenere un allineamento preciso e stabile degli elementi:

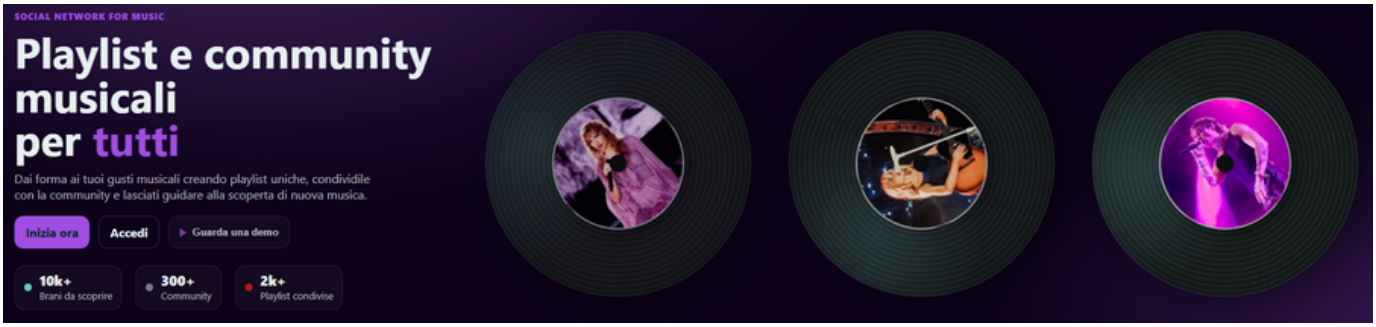
```
.nav{
  display: grid;
  grid-template-columns: 1fr auto 1fr;
  align-items: center;
}
```



Questa configurazione permette di mantenere il **menu perfettamente centrato**, indipendentemente dalla larghezza del logo o dei pulsanti laterali. Rispetto a una soluzione basata esclusivamente su Flexbox, CSS Grid offre un controllo maggiore sulla distribuzione dello spazio orizzontale e garantisce un comportamento più prevedibile anche in fase di responsive design.

HERO SECTION E ORGANIZZAZIONE DEL CONTENUTO

La **hero section** è contenuta all'interno del tag **<main>** ed è suddivisa in due colonne: una dedicata al contenuto testuale e alle call to action, l'altra a un elemento grafico decorativo.



Il layout è gestito tramite una **griglia a due colonne**:

```
<main id="top" class="hero">
  <section class="hero-left">...</section>
  <section class="hero-right">...
</section>
</main>
```

```
.hero{
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 26px;
  align-items: center;
}
```

Questa struttura consente di bilanciare testo e grafica, evitando che la pagina risulti eccessivamente densa di contenuti testuali o, al contrario, troppo povera di informazioni.

TITOLO PRINCIPALE E ADATTAMENTO RESPONSIVO

Il titolo principale utilizza un tag **<h1>** con una dimensione del font definita tramite la funzione **clamp()**:

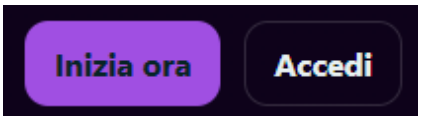
```
h1{
  font-size: clamp(32px, 4vw, 56px);
  line-height: 1.03;
}
```

L'uso di **clamp()** permette di definire una dimensione minima, una massima e un valore dinamico intermedio, garantendo così una buona leggibilità su dispositivi mobili e un forte impatto visivo su schermi di grandi dimensioni. Questa scelta evita la necessità di definire molte media query dedicate esclusivamente alla dimensione del testo.

CALL TO ACTION E GERARCHIA VISIVA

Le **call to action** sono state progettate per guidare l'utente verso l'azione principale, ovvero la **registrazione** alla piattaforma, senza però nascondere le azioni secondarie come il **login**.

```
<div class="cta">
  <a class="btn btn-primary"
href="register.html">Inizia ora</a>
  <a class="btn btn-ghost"
href="login.html">Accedi</a>
</div>
```



VINILI ANIMATI: COSTRUZIONE GRAFICA E ANIMAZIONE

Uno degli elementi più complessi e caratterizzanti della landing page è la rappresentazione dei **vinili animati**, realizzati interamente tramite CSS.

```
<div class="vinyl-row">
  <div class="vinyl"></div>
  <div class="vinyl"></div>
  <div class="vinyl"></div>
</div>
```

Ogni vinile è un elemento circolare la cui superficie è costruita attraverso l'uso di gradienti radiali:

```
.vinyl{
  width: 350px;
  height: 350px;
  border-radius: 50%;

  background:
    repeating-radial-gradient(
      circle,
      rgba(255,255,255,.10) 0 1px,
      rgba(255,255,255,0) 1px 7px
    ),
    radial-gradient(circle at center, #161c1c 0 60%, #0a0f10 100%);

  animation: spin 3.6s linear infinite;
}
```

Le scanalature del disco sono simulate tramite **repeating-radial-gradient**, evitando l'uso di immagini **raster** e migliorando sia le prestazioni sia la scalabilità grafica. L'animazione di rotazione è definita tramite una semplice **keyframe**:

```
@keyframes spin{
  to{
    transform: rotate(360deg);
  }
}
```

Velocità di rotazione differenti per ciascun vinile contribuiscono a rendere l'animazione più naturale e meno artificiale.

RESPONSIVE DESIGN

L'adattamento ai diversi dispositivi è ottenuto tramite l'uso di media query, che modificano il layout della navbar, delle griglie e dei vinili su schermi di dimensioni ridotte:

```
(@media (max-width: 980px){
  .vinyl-row{
    display: grid;
  }
}
```

Questo approccio consente di mantenere un'esperienza utente coerente e funzionale sia su desktop sia su dispositivi mobili.

PAGINA DI REGISTRAZIONE

INTRODUZIONE

La **pagina di registrazione** rappresenta uno degli snodi funzionali più importanti dell'intera applicazione SN4M, in quanto costituisce il primo vero punto di interazione strutturata tra l'utente e il sistema. A differenza della landing page, che ha principalmente un ruolo informativo e promozionale, la pagina di registrazione è una pagina operativa, progettata per raccogliere dati, validarli e inizializzare lo stato dell'utente all'interno dell'applicazione.

Dal punto di vista tecnico, la pagina è implementata nei file **register.html**, **register.css** e **register.js** . Anche in questo caso viene mantenuta una netta separazione tra struttura, stile e comportamento, in linea con le buone pratiche di sviluppo web.

STRUTTURA HTML GENERALE DELLA PAGINA

La struttura della pagina di registrazione è definita nel file **register.html**. A livello concettuale, la pagina è costruita come una card centrale, inserita all'interno di un contenitore che simula l'aspetto di uno smartphone. Questa scelta non è puramente estetica, ma risponde all'idea che la registrazione sia spesso effettuata da dispositivi mobili.

```
<div class="page">
  <div class="phone">
    <div class="card">
      ...
    </div>
  </div>
</div>
```

Il contenitore **.page** ha il compito di occupare l'intera viewport e di applicare lo sfondo a gradiente, mentre **.phone** funge da mockup grafico di un dispositivo mobile. All'interno di .card sono invece contenuti tutti gli elementi funzionali della pagina.

NAVIGAZIONE E CONTINUITA' DELL'ESPERIENZA UTENTE

All'interno della card è presente un **pulsante di ritorno** alla landing page, implementato come semplice link HTML:

```
<a href="landing_page.html" class="back">‹</a>
```

Questa soluzione garantisce continuità nella navigazione e permette all'utente di tornare indietro senza perdere il contesto. Dal punto di vista semantico, l'uso di un **<a>** invece di un <button> è coerente, in quanto si tratta di una vera e propria navigazione tra pagine.

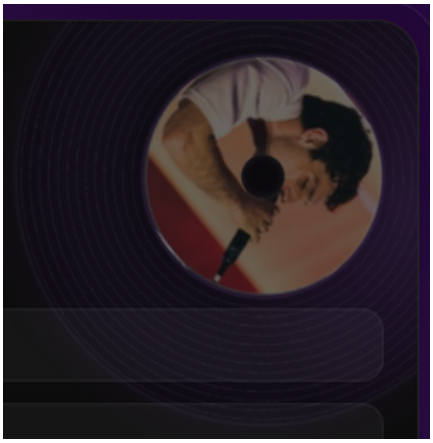
VINILI DECORATIVI E COERENZA VISIVA

Anche nella pagina di registrazione sono presenti elementi grafici a forma di **vinile**, in continuità con la landing page. Tuttavia, qui il loro ruolo è esclusivamente decorativo e non centrale nella composizione.

```
<div class="vinyl vinyl-top" aria-hidden="true"></div>
<div class="vinyl vinyl-bottom" aria-hidden="true"></div>
```

L'attributo **aria-hidden="true"** indica esplicitamente che questi elementi non hanno significato informativo e devono essere ignorati dalle tecnologie assistive, migliorando l'accessibilità della pagina.

Dal punto di vista CSS, i vinili sono posizionati in modo assoluto e animati lentamente tramite una rotazione continua, così da non distrarre l'utente durante la compilazione del form.



STRUTTURA DEL FORM DI REGISTRAZIONE

Il cuore della pagina è rappresentato dal form di registrazione, identificato tramite l'id **registerForm**:

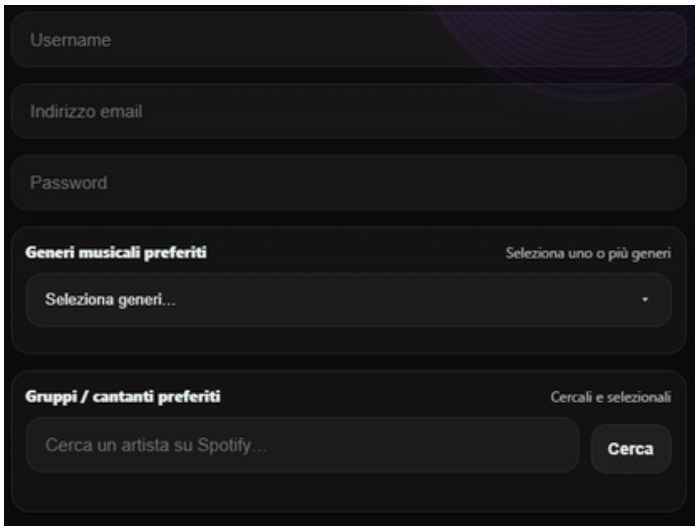
```
<form class="form" id="registerForm" autocomplete="on">
```

Il form raccoglie inizialmente tre informazioni fondamentali:

- **username;**
- **indirizzo email;**
- **password.**

Questi campi sono definiti tramite input HTML standard, sfruttando le funzionalità native del browser per la validazione di base:

```
<input type="text" name="username" required minlength="3" maxlength="20" />
<input type="email" name="email" required />
<input type="password" name="password" required minlength="8" />
```



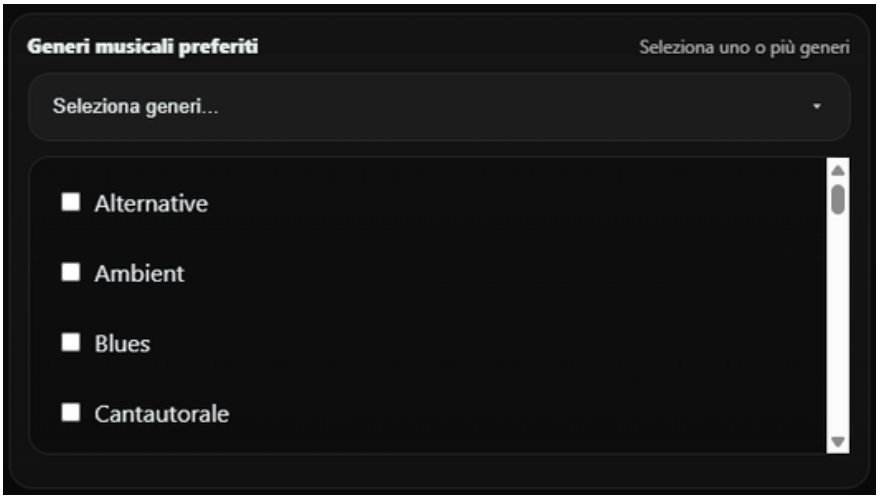
L'uso di attributi come **required**, **minlength** e **type="email"** consente una prima validazione lato client senza ricorrere immediatamente a JavaScript, riducendo il carico di codice e migliorando l'esperienza utente.

SELEZIONE DEI GENERI MUSICALI: MULTI-SELECT PERSONALIZZATO

Una delle parti più interessanti della pagina di registrazione è la selezione dei generi musicali preferiti, implementata tramite **un multi-select personalizzato**, anziché con un semplice <select> HTML.

```
<div class="multi" id="genreMulti">
  <button type="button" class="multi-btn">
    Seleziona generi…
  </button>

  <div class="multi-panel">
    <label class="opt">
      <input type="checkbox" value="Rock">
      <span>Rock</span>
    </label>
    …
  </div>

  <div class="multi-tags"></div>
  <input type="hidden" name="genres" id="genresHidden" />
</div>
```

Il componente è costruito combinando **checkbox standard** con un pannello a comparsa. Le selezioni effettuate dall'utente vengono mostrate sotto forma di tag visivi, mentre i valori effettivi sono memorizzati in un input hidden (**genresHidden**). Questa soluzione permette di mantenere il form compatibile con una normale sottomissione dei dati, pur offrendo un'interfaccia più moderna e intuitiva.



LOGICA JAVASCRIPT PER LA GESTIONE DEI GENERI

Nel file **register.js**, la logica di gestione del **multi-select** è interamente implementata in JavaScript.

```
function getCheckedGenreValues() {
  return Array.from(
    panel.querySelectorAll('input[type="checkbox"]:checked')
  ).map(i => i.value);
}
```

Questa funzione estrae i valori dei **checkbox selezionati** e li restituisce come array di stringhe. Successivamente, tali valori vengono utilizzati per aggiornare sia la visualizzazione dei tag sia il campo hidden associato:

```
function syncGenres() {
  const checked = getCheckedGenreValues();
  renderGenreTags(checked);
  hiddenGenres.value = checked.join(",");
}
```

In questo modo si mantiene una sincronizzazione costante tra:

- **stato dell'interfaccia;**
- **valori del form;**
- **dati che verranno effettivamente salvati.**

RICERCA E SELEZIONE DEGLI ARTISTI TRAMITE SPOTIFY API

Un altro elemento avanzato della pagina di registrazione è la possibilità di selezionare artisti musicali preferiti tramite una ricerca basata su **Spotify Web API**.

L'interfaccia HTML prevede un campo di ricerca, un pulsante e un'area per i risultati:

```
<input type="text" id="artistQuery" placeholder="Cerca un artista su Spotify..." />
<button type="button" id="artistSearchBtn">Cerca</button>
<div id="artistResults"></div>
<div id="artistTags"></div>
<input type="hidden" name="artists" id="artistsHidden" />
```

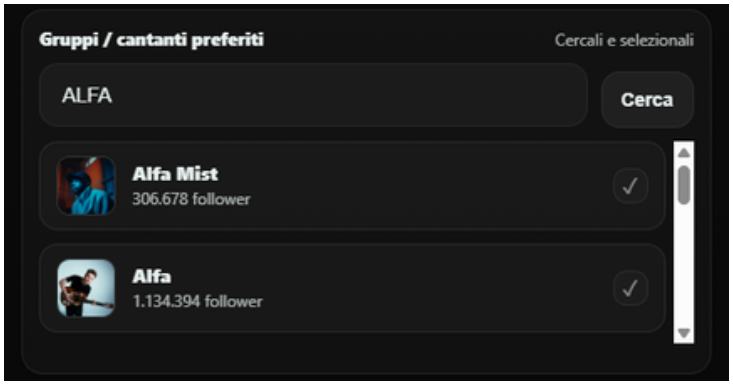
Dal punto di vista JavaScript, la ricerca avviene in più fasi. In primo luogo viene ottenuto un token OAuth tramite le credenziali dell'applicazione:

```
async function fetchSpotifyToken() {
  const res = await fetch("https://accounts.spotify.com/api/token", {
    method: "POST",
    headers: {
      Authorization: "Basic " + btoa(clientId + ":" + clientSecret),
      "Content-Type": "application/x-www-form-urlencoded"
    },
    body: new URLSearchParams({ grant_type: "client_credentials" })
  });
  ...
}
```

Successivamente, il token viene utilizzato per interrogare l'endpoint di ricerca degli artisti:

```
async function searchArtists(q) {
  const token = await getSpotifyToken();
  const res = await fetch(
    `https://api.spotify.com/v1/search?type=artist&q=${encodeURIComponent(q)}`,
    { headers: { Authorization: "Bearer " + token } }
  );
  ...
}
```

I risultati vengono poi renderizzati dinamicamente nella pagina e possono essere selezionati dall'utente. Gli artisti selezionati vengono mostrati sotto forma di tag e salvati in un input hidden, in modo analogo ai generi musicali.



VALIDAZIONE FINALE E SALVATAGGIO DELL'UTENTE

Al submit del form, la logica JavaScript intercetta l'evento per eseguire una validazione completa:

```
form.addEventListener("submit", (e) => {
  e.preventDefault();
  ...
});
```

Vengono controllati:

- **presenza di almeno un genere musicale;**
- **presenza di almeno un artista selezionato;**
- **unicità di username ed email.**

I dati dell'utente vengono infine salvati in localStorage:

```
users.push(newUser);
localStorage.setItem("users", JSON.stringify(users));
localStorage.setItem("currentUser", JSON.stringify(newUser));
```

Questa soluzione consente di simulare un sistema di registrazione completo anche in assenza di un backend, rendendo il progetto pienamente funzionante in ambiente client-side.

PAGINA DI LOGIN

INTRODUZIONE

La **pagina di login** rappresenta il punto di accesso per gli utenti già registrati al sistema SN4M. Dal punto di vista concettuale, essa svolge una funzione complementare rispetto alla pagina di registrazione: mentre quest'ultima si occupa di raccogliere e inizializzare i dati dell'utente, la pagina di login ha il compito di verificare l'identità dell'utente e ripristinare il suo stato all'interno dell'applicazione.

Anche per questa pagina è stata mantenuta una forte coerenza visiva e strutturale con la pagina di registrazione e con la landing page, sia per garantire continuità nell'esperienza utente, sia per rafforzare l'identità visiva dell'applicazione. I file coinvolti sono login.html, login.css e login.js .

STRUTTURA HTML DELLA PAGINA DI LOGIN

La struttura generale della pagina di login ricalca volutamente quella della pagina di registrazione. Anche in questo caso, l'interfaccia è racchiusa all'interno di una **card centrale**, inserita in un contenitore che simula un dispositivo mobile.

```
<div class="page">
  <div class="phone">
    <div class="card">
      ...
    </div>
  </div>
</div>
```

Questa scelta progettuale ha un duplice obiettivo. Da un lato, garantisce una forte coerenza visiva tra le due pagine; dall'altro, suggerisce implicitamente che l'applicazione è pensata per essere utilizzata frequentemente anche da smartphone, contesto in cui le operazioni di login sono molto comuni.

All'interno della card è nuovamente presente un pulsante di ritorno alla landing page:

```
<a href="landing_page.html" class="back">‹</a>
```

La presenza di questo elemento consente all'utente di interrompere il processo di autenticazione e tornare alla pagina principale senza dover ricorrere ai controlli del browser, migliorando l'usabilità complessiva.

ELEMENTI DECORATIVI E CONTINUITÀ VISIVA

Analogamente alla pagina di registrazione, anche nella pagina di login sono presenti due **vinili decorativi** posizionati nella parte superiore e inferiore della card:

```
<div class="vinyl vinyl-top"></div>
<div class="vinyl vinyl-bottom"></div>
```

Questi elementi non hanno alcuna funzione informativa o interattiva, ma contribuiscono a mantenere una forte continuità estetica tra le diverse sezioni dell'applicazione. Dal punto di vista CSS, i vinili sono posizionati in modo assoluto e animati lentamente tramite una rotazione continua, così da risultare visivamente presenti ma non invasivi durante la compilazione del form.

CONTENUTO PRINCIPALE E MESSAGGIO ALL'UTENTE

Il contenuto centrale della pagina è racchiuso nel contenitore **.content** e si apre con un messaggio di benvenuto rivolto all'utente:

Questa scelta testuale ha una funzione importante: a differenza della registrazione, che introduce l'utente al sistema, il login presuppone una familiarità preesistente. Il messaggio **"Bentornato/a"** rafforza quindi l'idea di un ritorno in un ambiente già conosciuto, rendendo l'esperienza più personale e accogliente.

STRUTTURA DEL FORM DI LOGIN

Il form di login è volutamente essenziale e raccoglie solo le informazioni strettamente necessarie all'autenticazione:

```
<form class="form" id="loginForm">
  <input type="text" name="username" required minlength="3" maxlength="20" />
  <input type="password" name="password" required />
  ...
</form>
```

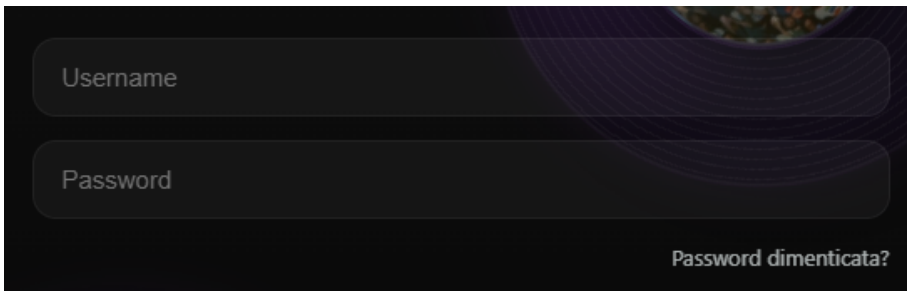
La scelta di richiedere lo **username** anziché l'email mantiene coerenza con la logica adottata nella fase di registrazione, dove lo username è un identificativo univoco dell'utente. Anche in questo caso vengono sfruttati gli attributi HTML nativi (required, minlength) per una prima validazione lato client, riducendo la necessità di controlli immediati in JavaScript.

All'interno del form è inoltre presente un link per il **recupero della password**:

```
<a href="forgot_password.html" class="forgot">Password dimenticata?</a>
```

La scelta di richiedere lo **username** anziché l'email mantiene coerenza con la logica adottata nella fase di registrazione, dove lo username è un identificativo univoco dell'utente. Anche in questo caso vengono sfruttati gli attributi HTML nativi (required, minlength) per una prima validazione lato client, riducendo la necessità di controlli immediati in JavaScript.

All'interno del form è inoltre presente un link per il **recupero della password**:



GESTIONE DELLO STILE DEL FORM

Dal punto di vista grafico, il form utilizza le stesse scelte stilistiche della pagina di registrazione, in particolare per quanto riguarda:

- **campi di input con sfondo semitrasparente;**
- **bordi arrotondati;**
- **evidenziazione del campo attivo tramite cambio di colore del bordo.**

```
input{
  padding:14px 16px;
  border-radius:14px;
  border:1px solid var(--stroke-2);
  background: rgba(255,255,255,.04);
}
```

Questa coerenza visiva riduce il carico cognitivo dell'utente, che ritrova un'interfaccia già familiare.

LOGICA JAVASCRIPT PER L'AUTENTICAZIONE

La gestione del login è implementata nel file **login.js** ed è racchiusa, come nelle altre pagine, all'interno di una IIFE (**Immediately Invoked Function Expression**), per evitare l'inquinamento dello scope globale.

```
(() => {
  const form = document.getElementById("loginForm");
  if (!form) return;
  ...
})();
```

Il primo passo consiste nel recupero della lista degli utenti registrati dal localStorage:

```
function getUsers() {
  try {
    const users = JSON.parse(localStorage.getItem("users"));
    return Array.isArray(users) ? users : [];
  } catch {
    return [];
  }
}
```

Questa funzione è volutamente robusta: utilizza un blocco **try/catch** per evitare che eventuali dati corrotti nel localStorage causino errori bloccanti nell'esecuzione dello script.

VALIDAZIONE DELLE CREDENZIALI

Al submit del form, l'evento viene intercettato per eseguire la validazione manuale delle credenziali:

```
form.addEventListener("submit", (e) => {
  e.preventDefault();
  ...
});
```

Vengono quindi confrontati username e password inseriti dall'utente con quelli salvati localmente:

```
const user = users.find(
  (u) =>
    (u.username || "").toLowerCase() === username.toLowerCase() &&
    u.password === password
);
```

Il confronto dello username avviene in modo case-insensitive, migliorando l'usabilità e riducendo errori dovuti a differenze di maiuscole e minuscole. La password, invece, viene confrontata in modo diretto, in quanto in questa fase del progetto non è prevista una gestione avanzata della cifratura (che sarebbe invece demandata a un backend reale).

Nel caso in cui le credenziali non siano valide, viene mostrato un messaggio di errore:

```
showError("Username o password non corretti.");
```

PAGINA DI PASSWORD DIMENTICATA

INTRODUZIONE

La pagina "Password dimenticata" completa il flusso di autenticazione dell'applicazione SN4M, offrendo all'utente una modalità di recupero dell'accesso in caso di smarrimento delle credenziali. Dal punto di vista progettuale, essa si colloca come estensione naturale della pagina di login e ne mantiene coerenza visiva e strutturale.

I file coinvolti sono **forgot_password.html**, **forgot_password.css** e **forgot_password.js** .

STRUTTURA HTML E ORGANIZZAZIONE DEGLI STEP

La pagina è costruita secondo lo stesso schema delle pagine di login e registrazione: una card centrale all'interno di un mockup mobile, con sfondo a gradiente e vinili decorativi.
All'interno del contenuto principale sono presenti due form distinti, che rappresentano due step logici del recupero password:

- **Step 1**: inserimento dell'indirizzo email;
- **Step 2**: inserimento della nuova password (attualmente simulato).

```
<form class="form" id="requestForm">
  <input type="email" name="email" required />
  <p class="error" id="errorBox"></p>
  <button type="submit">Continua</button>
</form>

<form class="form hidden" id="resetForm">
  ...
</form>
```

Il secondo form è inizialmente nascosto tramite la classe **.hidden** e può essere mostrato dal JavaScript. Questa struttura consente di simulare un flusso multi-step senza cambiare pagina, migliorando la chiarezza dell'interazione.

GESTIONE DELLO STILE E COERENZA VISIVA

Dal punto di vista CSS, la pagina riutilizza quasi interamente le scelte stilistiche delle altre pagine di autenticazione: stessi colori, stessi vinili decorativi, stessa tipografia e stessi componenti di input.

Questo approccio riduce la duplicazione concettuale e rafforza l'identità visiva dell'applicazione, facendo percepire le diverse pagine come parti di un unico sistema coerente.

LOGICA JAVASCRIPT PER IL RECUPERO PASSWORD

Il comportamento della pagina è gestito nel file **forgot_password.js**, anch'esso racchiuso in una IIFE per evitare variabili globali indesiderate.

```
(() => {
  const USERS_KEY = "users";
  ...
})();
```

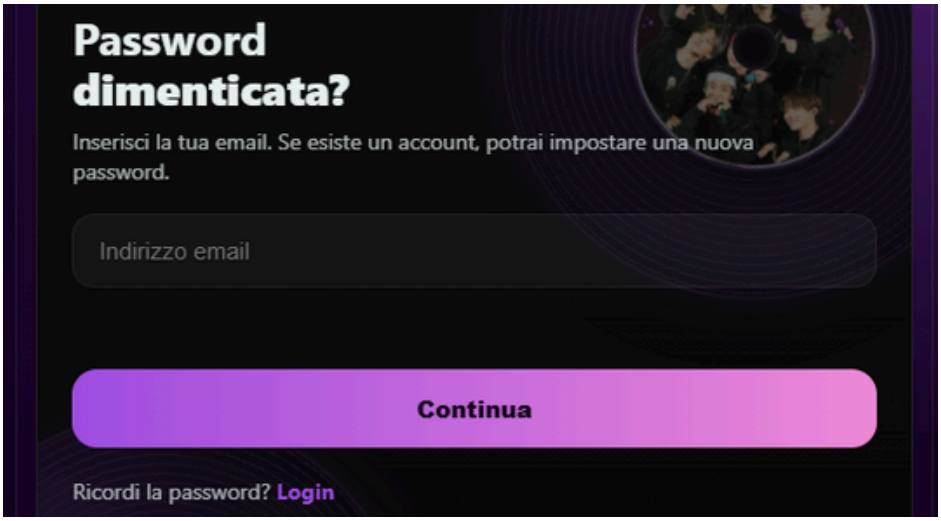
Alla sottomissione del primo form, lo script:

1. **valida il formato dell'email inserita;**
2. **carica la lista degli utenti dal localStorage;**
3. **verifica l'esistenza di un account associato all'email.**

```
const userExists = users.some(
  u => normalizeEmail(u.email) === email
);
```

Se l'email non è associata a nessun account, viene mostrato un messaggio di errore; in caso contrario, viene simulato l'invio di un'email di recupero password tramite un messaggio informativo.

Questa simulazione è coerente con l'architettura client-side del progetto: in un'applicazione reale, questa logica sarebbe demandata a un backend, mentre qui viene riprodotta per completezza funzionale.



HOMEPAGE

INTRODUZIONE

La **homepage** rappresenta il cuore funzionale dell'applicazione SN4M, ovvero la prima pagina visualizzata dall'utente dopo l'autenticazione. A differenza delle pagine precedenti, dedicate all'accesso, questa vista è pensata per fornire una panoramica immediata dello stato dell'utente, delle sue playlist, delle community a cui partecipa e dei contenuti condivisi.

I file coinvolti sono **homepage.html**, **homepage.css** e **homepage.js** .

STRUTTURA GENERALE E LAYOUT DELL'INTERFACCIA

La homepage è organizzata secondo una classica architettura sidebar + contenuto principale, tipica delle web app moderne.
Nel file HTML, l'intera applicazione è racchiusa nel contenitore .app, che include:

- **una sidebar laterale per la navigazione principale;**
- **un'area .main contenente header e contenuto dinamico.**

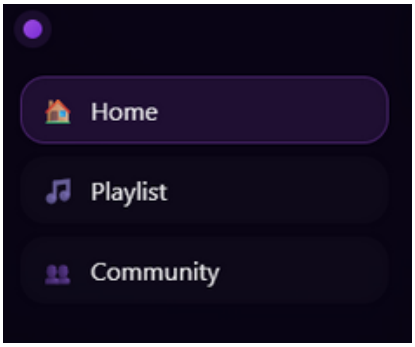
```
<div class="app">
  <aside class="sidebar">...</aside>
  <div class="main">...</div>
</div>
```

SIDEBAR DI NAVIGAZIONE

La **sidebar** contiene i link principali dell'applicazione (**Home**, **Playlist**, **Community**) ed è progettata per essere semplice e immediata. L'elemento attivo è evidenziato tramite la classe isActive, così da fornire un feedback visivo chiaro all'utente.

Dal punto di vista CSS, la sidebar è resa sticky, in modo da rimanere visibile anche durante lo scroll della pagina:

```
.sidebar{
  position: sticky;
  top: 0;
  height: 100vh;
}
```



HEADER SUPERIORE E RICERCA GLOBALE

L'header superiore contiene tre elementi fondamentali:

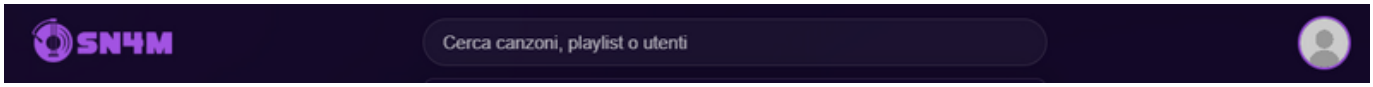
- **logo cliccabile per tornare alla landing page;**
- **campo di ricerca globale;**
- **avatar dell'utente autenticato.**

```
<input id="globalSearch" type="text" placeholder="Cerca canzoni, playlist o utenti">
```

La ricerca globale è implementata in JavaScript tramite integrazione con **Spotify Web API** e utilizza una funzione di debounce per evitare richieste eccessive durante la digitazione:

```
function debounce(fn, delay = 300){
 let t;
 return (...args) => {
  clearTimeout(t);
  t = setTimeout(() => fn(...args), delay);
 };
}
```

I risultati vengono mostrati in un dropdown e, al click su un brano, viene aperto un modal informativo con i dettagli del brano selezionato.



HERO CARD E KPI INIZIALI

La parte superiore della homepage presenta una hero card che accoglie l'utente e riassume la sua attività tramite indicatori numerici (**KPI**), come numero di playlist, community e condivisioni.

```
<div class="heroKpi__n" id="heroPlCount">—</div>
```

Questi valori vengono calcolati dinamicamente in JavaScript leggendo i dati dal localStorage:

```
function updateHeroKpis(){
 const pls = readPlaylists();
 heroPlCount.textContent = String(pls.length);
}
```

Questo approccio rende la homepage immediatamente personalizzata in base ai dati dell'utente.

SEZIONE "LE TUE PLAYLIST" E COMMUNITY DELL'UTENTE

La homepage mostra solo le **ultime 4 playlist create**, per evitare sovraccarico visivo e mantenere la pagina ordinata. Ogni playlist è visualizzata in modalità read-only e, al click, apre un modal di dettaglio che mostra informazioni e brani senza cambiare pagina. Questa scelta migliora la fluidità dell'esperienza utente.

La **sezione Community** mostra solo le community a cui l'utente partecipa. I dati vengono recuperati dal localStorage e filtrati in base all'identificativo dell'utente corrente. In assenza di community, viene mostrato un messaggio informativo, evitando sezioni vuote poco chiare.

Una delle parti più avanzate della homepage è la sezione dedicata alle playlist condivise nelle community dell'utente. Qui è possibile:

- **filtrare per tag;**
- **filtrare per canzone o artista;**
- **importare playlist nel proprio profilo.**

Questa funzionalità rafforza il concetto di social network musicale, permettendo la scoperta di nuovi contenuti all'interno delle community.

PLAYLIST E COMMUNITY

INTRODUZIONE

Le sezioni **Playlist** e **Community** costituiscono il nucleo centrale dell'esperienza SN4M, in quanto permettono all'utente non solo di creare e organizzare contenuti musicali, ma anche di condividerli e scoprirne di nuovi all'interno di gruppi tematici. Dal punto di vista progettuale, queste due pagine sono state pensate come **fortemente interconnesse** e sono state implementate riutilizzando la stessa architettura di layout della homepage, basata su sidebar laterale e contenuto principale.

GESTIONE DELLE PLAYLIST

La **pagina delle playlist** consente all'utente di visualizzare, creare e gestire le proprie playlist musicali. L'interfaccia utilizza una griglia di tile, ciascuna rappresentante una playlist.

```
<div class="playlistGrid playlistGrid--all playlistGrid--tiles" id="allPlaylistGrid"></div>
```

Ogni playlist è rappresentata come una card cliccabile che apre un modal di dettaglio, evitando il cambio di pagina e mantenendo il contesto dell'utente.

Dal punto di vista dei dati, le playlist sono salvate nel **localStorage** come unica fonte di verità. La lettura e scrittura avvengono tramite funzioni dedicate:

```
function readPlaylists(){
 try{
  const raw = localStorage.getItem(PLAYLISTS_STORAGE_KEY);
  const list = raw ? JSON.parse(raw) : [];
  return Array.isArray(list) ? list : [];
 }catch{
  return [];
 }
}

function writePlaylists(list){
 localStorage.setItem(PLAYLISTS_STORAGE_KEY, JSON.stringify(list));
 window.dispatchEvent(new CustomEvent("sn4m:playlists-changed"));
}
```

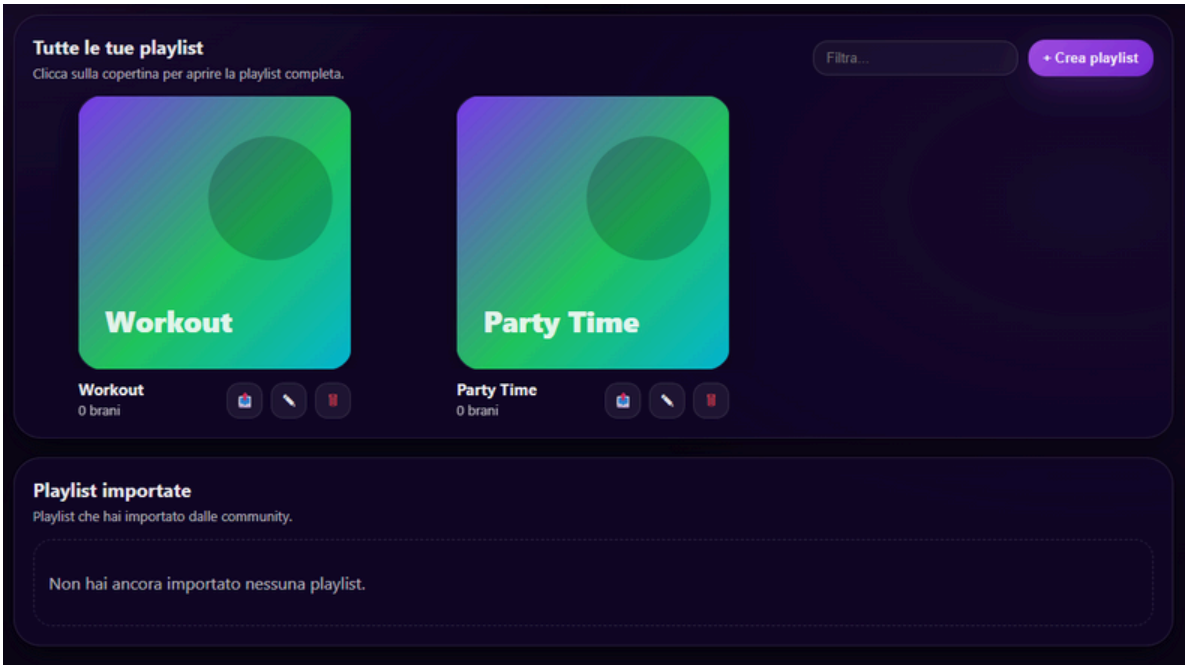
L'emissione dell'evento custom **sn4m:playlists-changed** consente di aggiornare automaticamente l'interfaccia anche nella stessa tab del browser, simulando un comportamento reattivo tipico delle Single Page Application.

CREAZIONE E MODIFICA DELLE PLAYLIST

La **creazione** (e modifica) di una playlist avviene tramite un modal che raccoglie titolo, descrizione, tag e brani. La validazione dei campi principali è gestita in JavaScript prima del salvataggio:

```
function validateCoreFields({ title, desc, tags }) {
  if (!title) return "Inserisci un titolo.";
  if (!desc) return "Inserisci una descrizione.";
  if (!tags || tags.length === 0) return "Inserisci almeno un tag.";
  return "";
}
```

Questo controllo garantisce la coerenza dei dati salvati e migliora l'esperienza utente, fornendo feedback immediato in caso di errore.



PAGINA COMMUNITY

La pagina **Community** permette all'utente di esplorare e creare community musicali. Anche qui l'interfaccia è basata su una griglia di tile, coerente con quella delle playlist.

```
<div class="playlistGrid playlistGrid--all playlistGrid--tiles" id="communityGrid"></div>
```

Le community sono suddivise logicamente in:

- **tutte le community disponibili;**
- **community di cui l'utente fa parte.**

Il filtraggio avviene lato client confrontando il testo inserito con titolo e tag della community.

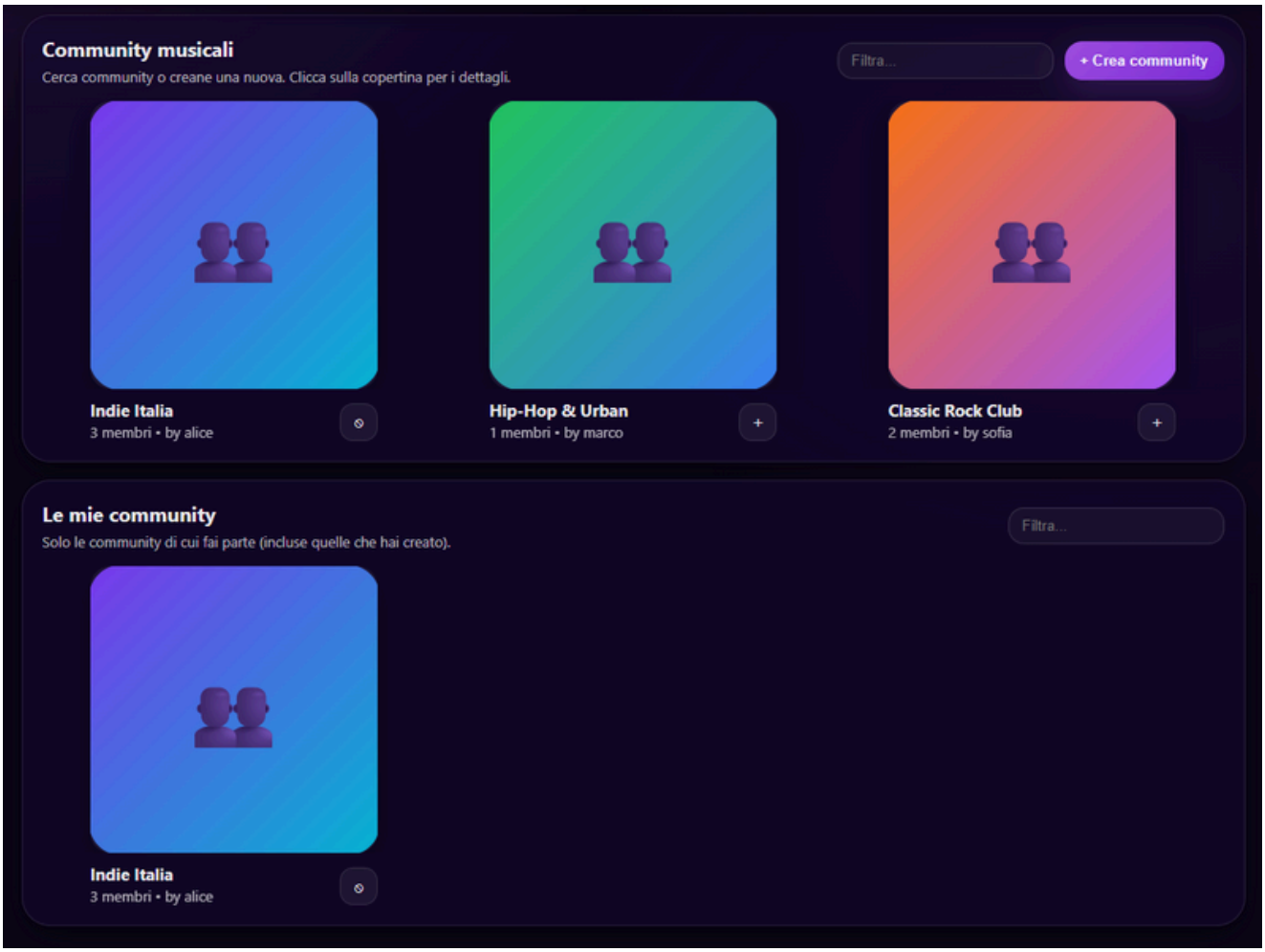
CREAZIONE DELLE COMMUNITY

La creazione di una **community** avviene tramite un modal dedicato, che raccoglie titolo, descrizione e tag:

```
<form class="modal__form" id="communityForm">
  <input id="coTitle" type="text" required />
  <textarea id="coDesc" required></textarea>
  <input id="coTags" type="text" required />
</form>
```

I tag vengono poi normalizzati in JavaScript per essere salvati come array:

```
function parseTags(txt) {
  return String(txt || "")
    .split(",")
    .map(s => s.trim())
    .filter(Boolean);
}
```



COLLEGAMENTO TRA PLAYLIST E COMMUNITY

Il **collegamento** tra playlist e community è uno degli aspetti più rilevanti del progetto. Ogni playlist può essere condivisa in una o più community, mantenendo traccia degli identificativi delle community associate.

```
sharedCommunityIds: Array.isArray(p.sharedCommunityIds)
 ? p.sharedCommunityIds.map(String)
 : []
```

Nel dettaglio di una community, vengono mostrate le playlist condivise e i relativi brani, permettendo all'utente di esplorare contenuti musicali creati da altri membri e, se desiderato, importarli nel proprio profilo.

PROFILO UTENTE

INTRODUZIONE

La pagina **Profilo utente** consente all'utente autenticato di gestire i propri dati personali e le preferenze musicali, rappresentando un punto di controllo centrale sull'identità dell'utente all'interno dell'applicazione. Dal punto di vista funzionale, questa pagina permette la modifica delle informazioni di base (username ed email), la gestione dei generi musicali preferiti e degli artisti seguiti, nonché l'eliminazione definitiva dell'account.

I file coinvolti sono **profile.html**, **profile.css** e **profile.js**.

STRUTTURA HTML DELLA PAGINA

La struttura della pagina è basata su una card centrale, coerente con le altre sezioni dell'applicazione, e include una top bar con logo e avatar dell'utente.

```
<header class="topBar">
 <button id="logoBtn" class="logoBtn">
  <img src="media/logo.png" class="logoImg" />
 </button>

 <div class="profile">
  <img id="profileImg" />
 </div>
</header>
```

Il contenuto principale ospita un form unico (**profileForm**) che raccoglie tutte le informazioni modificabili. Questa scelta evita la frammentazione dell'interfaccia e rende il salvataggio dei dati più intuitivo.

CARICAMENTO E GESTIONE DELL'UTENTE CORRENTE

All'apertura della pagina, il JavaScript verifica la presenza di un utente **autenticato**. In caso contrario, l'utente viene reindirizzato alla landing page:

```
function getCurrentUser() {
  try {
    return JSON.parse(localStorage.getItem("currentUser") || "null");
  } catch {
    return null;
  }
}

if (!getCurrentUser()) {
  window.location.href = "landing_page.html";
  return;
}
```

Questa logica impedisce l'accesso diretto alla pagina profilo senza autenticazione, simulando un controllo di accesso tipico delle applicazioni web con backend.

MODIFICA DEI DATI PERSONALI

Username ed **email** vengono caricati automaticamente nel form e possono essere modificati dall'utente:

```
function fillFormFromUser(u) {
  inputUsername.value = u.username || "";
  inputEmail.value = u.email || "";
}
```

Al salvataggio, i dati vengono aggiornati sia nel **currentUser** sia nella lista globale degli utenti salvata nel **localStorage**, garantendo coerenza tra stato attivo e dati persistenti.

SELEZIONE DEI GENERI MUSICALI

La selezione dei generi musicali è implementata tramite un **multi-select personalizzato**, analogo a quello utilizzato nella registrazione. I generi disponibili sono definiti in un array statico e renderizzati dinamicamente:

```
const GENRE_OPTIONS = ["Pop","Rock","Rap","Jazz","EDM","Metal", ...];

function renderGenresOptions() {
  GENRE_OPTIONS.forEach(g => {
    // checkbox + label
  });
}
```

La scelta di un componente custom consente un'interazione più chiara rispetto a un **<select>** HTML standard, mantenendo comunque il controllo completo dello stato tramite JavaScript.

GESTIONE DEGLI ARTISTI PREFERITI (SPOTIFY API)

La pagina profilo permette di **aggiornare gli artisti preferiti** tramite ricerca su Spotify Web API, riutilizzando la stessa logica della registrazione. È imposto un limite massimo di artisti selezionabili per evitare un'eccessiva dispersione dei dati:

```
const MAX_ARTISTS = 8;

function toggleArtist(name) {
  if (selectedArtists.length >= MAX_ARTISTS) {
    alert("Puoi selezionare al massimo 8 artisti.");
    return;
  }
  ...
}
```

Gli artisti selezionati vengono mostrati visivamente e salvati come array all'interno del profilo utente.

ELIMINAZIONE DELL'ACCOUNT

Una delle funzionalità più rilevanti della pagina profilo è la possibilità di **eliminare definitivamente l'account**. L'azione è protetta da un modal di conferma, per evitare cancellazioni accidentali.

```
confirmDeleteBtn.addEventListener("click", () => {
  const users = JSON.parse(localStorage.getItem("users") || "[]");
  const updatedUsers = users.filter(u => u.email !== currentUser.email);
  localStorage.setItem("users", JSON.stringify(updatedUsers));
  localStorage.removeItem("currentUser");
  window.location.href = "landing_page.html";
});
```

Questa operazione rimuove l'utente dal sistema e cancella anche i dati associati (playlist e token Spotify), simulando una vera eliminazione dell'account lato server.