Traccia d’esame - Progetto gestione digitale delle carriere musicali Music Career

Daniele Visone 0334000130

Il progetto Music Career ha come obiettivo la realizzazione di un sistema informativo per la gestione delle carriere musicali di artisti e manager. L’applicativo web è stato sviluppato utilizzando Django, un framework web basato su Python, che consente un’implementazione sicura, scalabile e facilmente mantenibile di sistemi informativi relazionali. Il sistema consente di: gestire rapporti contrattuali tra manager e artisti, registrare e visualizzare brani pubblicati e concerti organizzati, per i manager la possibilità di monitorare i propri artisti e, se necessario, azzerare le statistiche.

L’applicazione è stata sviluppata utilizzando per tutto il ciclo di vita della progettazione Django, un framework web basato su Python, che consente in modo sicuro di implementare sistemi informativi relazionali, nel nostro caso il database a cui fa affidamento Django è SQLite3. Il front-end dell’applicazione è realizzato con HTML5 e CSS.

Analisi e progettazione concettuale

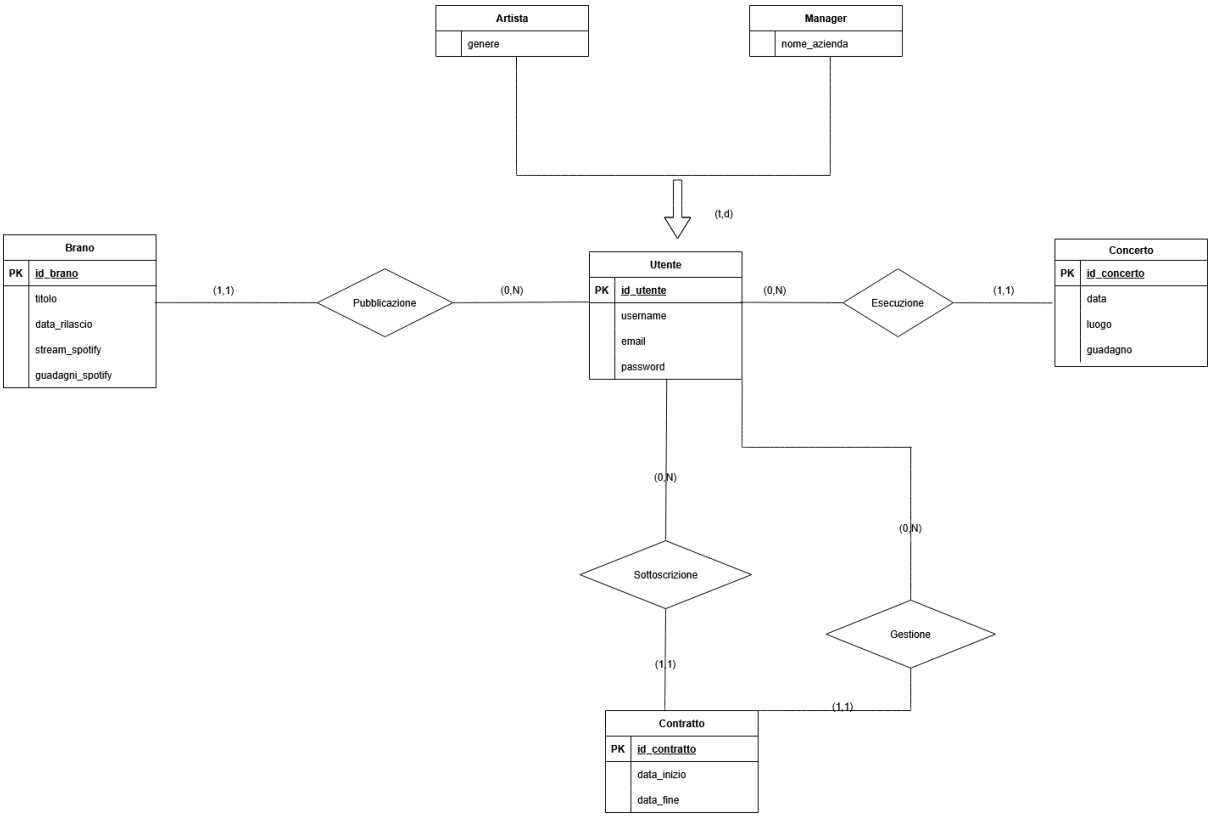
Il dominio applicativo riguarda la gestione delle carriere artistiche musicali. Ogni Utente può registrarsi come Artista o Manager. Gli artisti pubblicano brani e partecipano a concerti, mentre i manager stipulano contratti di collaborazione con gli artisti per gestirne le loro attività.

Entità individuabili nel progetto

1)**Utente**: gestisce l’autenticazione e i dati base dell’account (username, password ed email).
2)**Artista**: contiene le informazioni aggiuntive sull’artista. È collegato a un utente e include genere musicale, guadagni totali e riferimenti ai brani e ai concerti.

3)**Manager**: Rappresenta l’utente supervisore dell’artista che gestisce contratti e artisti.
4)**Contratto:** Relazione tra artista e un manager. Ha come attributi la data di inizio e fine del rapporto lavorativo.
5)**Traccia:** Rappresenta un brano pubblicato da un artista. Contiene informazioni su titolo, data di pubblicazione, genere musicale, numero di stream Spotify (concettuale, non implementato con API) e ricavi derivanti.
6)**Concerto:** Evento dal vivo organizzato per un artista. Contiene data, luogo e guadagno ottenuto.

Modello E-R (entità relazioni)



Il sistema è incentrato sulla gestione delle carriere musicali e delle interazioni tra artisti e manager. È stato realizzato un modello ER che prevede un'entità padre (Utente) e due entità figlie (Artista e Manager), collegata da una relazione di generalizzazione.

Relazione Utente -> Artista e Manager

- Vincolo di copertura: Totale (nel contesto dell'applicazione, ogni profilo operativo deve essere o un artista o un manager).
- Vincolo di disgiunzione: Disgiunta (un singolo profilo utente è specializzato come Artista o come Manager).

  La specializzazione dell'entità Utente è stata così definita:
  - Attributi comuni (Utente): id_utente, username, password, email (gestiti dal modello User di Django).
  - Artista (ArtistProfile): genere musicale.
  - Manager (ManagerProfile): nome azienda.

    Soluzioni possibili alla generalizzazione
    Per la gestione delle entità Artista e Manager, che derivano dall'entità generale Utente, sono state valutate due strategie implementative per il passaggio dal modello concettuale al modello logico relazionale.

1. Soluzione 1: Accorpamento nell'entità padre
   In quest'approccio, si sarebbe creata un'unica tabella Utente contenente sia gli attributi comuni (username, email) sia tutti gli attributi specifici delle sottoclassi (genere, nome_azienda). Un attributo discriminante (esempio ruolo) avrebbe distinto il tipo di utente.
   Vantaggi:

- Gestione di una sola tabella fisica.
- Query molto semplici e veloci per recuperare tutti gli utenti del sistema, senza necessità di JOIN.

Svantaggi:

- Spreco di memoria e valori NULL: I campi specifici dell'Artista (es. genere) rimarrebbero vuoti (NULL) per tutti i Manager, e viceversa.
- Minore integrità semantica: Non si possono imporre vincoli NOT NULL sugli attributi specifici a livello di database, delegando il controllo interamente all'applicazione.
- Complessità delle relazioni: La tabella Utente diventerebbe un'entità accentratrice con troppe relazioni in uscita (es. verso Brano, Concerto, Contratto), rendendo lo schema poco chiaro.

2. Soluzione 2: Accorpamento nelle figlie / Tabelle Separate
   Questo è l'approccio che prevede il mantenimento delle tabelle fisiche distinte per le sottoclassi (ArtistProfile, ManagerProfile), collegate alla tabella padre (User), tramite una relazione (1,1).
   Vantaggi:
   - Nessun valore NULL ingiustificato: Ogni tabella contiene solo le colonne pertinenti al proprio ruolo.
   - Integrità referenziale: È possibile definire vincoli specifici e relazioni pulite (es. solo la tabella ArtistProfile ha la relazione con Brano).
   - Scalabilità: È possibile aggiungere nuovi attributi complessi per gli Artisti in futuro senza impattare la struttura dati dei Manager.

   Svantaggi:
   - Richiede operazioni di JOIN per recuperare i dati completi di un utente (es. username e il genere), il che può essere leggermente più oneroso a livello di elaborazione.

Motivazione scelta progettuale

Per il sistema informativo in oggetto è stata adottata la Soluzione 2 (implementata in Django tramite OneToOneField).

La decisione è motivata dalla differenza operativa tra le due ruoli:

1. Gli Artisti interagiscono con entità quali Brani e Concerti.
2. I Manager gestiscono principalmente Contratti.

Accorpare tutto in un'unica tabella, come prevista dalla soluzione 1, avrebbe creato confusione nelle relazioni, permettendo potenzialmente ad un Manager di avere dei Brani, violando la logica di business. La separazione in tabelle distinte garantisce che ogni ruolo acceda esclusivamente alle funzionalità e ai dati di sua competenza, privilegiando la correttezza e la pulizia del dato rispetto alla pura semplicità delle query.

Progettazione Logica

Di seguito è riportato lo schema logico relazionale dal modello concettuale, con una descrizione delle tabelle e dei rispettivi attributi

- Utente (id_utente, username, email, password). Questa tabella rappresenta l'entità principale per la gestione degli accessi al sistema. Ogni utente è identificato univocamente da un campo id autoincrementale. Le credenziali di accesso sono costituite dallo username (che deve essere univoco) e dalla password (memorizzata da Django sottoforma di bcrypt). L'attributo email è necessario per le comunicazioni di sistema e recupero credenziali. La tabella Utente costituisce la superclasse da cui derivano i profili specifici.
- Artista (id_artista, genere). Questa tabella rappresenta la specializzazione dell'utente con ruolo artistico. È collegata alla tabella tramite l'id (relazione 1,1). Oltre ai dati ereditati, l'Artista è caratterizzato dall'attributo genere, che specifica lo stile musicale di appartenenza. La figura artista è abilitata a pubblicare brani ed eseguire concerti.
- Manager (id_manager, nome_azienda). Questa tabella rappresenta la specializzazione dell'utente con ruolo gestionale. Anch'essa è collegata alla tabella Utente tramite l'id. Il Manager si distingue per l'attributo nome_azienda, che indica l'etichetta o l'agenzia di management per cui opera. Il compito principale del Manager è la stipula dei contratti con gli Artisti.
- Brano (id_brano, titolo, data_rilascio, stream_spotify, guadagni_spotify). Questa tabella rappresenta le singole tracce musicali presenti nel sistema. Ogni brano è identificato da un id e possiede un titolo e una data_rilascio. Sono inoltre presenti attributi necessari per il calcolo dei compensi stream_spotify e guadagni_spotify. Ogni brano è associato a un Artista tramite una chiave esterna.
- Concerto (id_concerto, data, luogo, biglietti_venduti, ricavo). Questa tabella rappresenta gli eventi dal vivo. Ogni concerto è un'entità distinta identificata da un id, che si svolge in una determinata data e in un luogo. Per monitorare le entrate, vengono

registrati biglietti_venduti e il guadagno totale dell'evento. Anche qui, ogni concerto è legato all'Artista che lo esegue.

- Contratto (id_contratto, data_inizio, data_fine). Questa tabella formalizza la relazione lavorativa tra Artista e Manager. Oltre all'identificativo id, il contratto è definito temporalmente da una data_inizio e da una data_fine. Questa entità funge da collegamento N,N logico tra le due figure professionali, regolamentando la loro collaborazione.

Di seguito sono riportate le tabelle presenti nel database

Tabella Utente

Nome campo	Tipo di dato	Chiave	Vincoli
id_utente	INT	PK	Autoincrement
username	VARCHAR(150)		UNIQUE, NOT NULL
password	VARCHAR(128)		UNIQUE, NOT NULL
email	VARCHAR(254)		NOT NULL

Tabella Artista

Nome campo	Tipo di dato	Chiave	Vincoli
id_artista	INT	PK	Autoincrement
id_utente	INT	FK	Utente(id), UNIQUE (1,1)
genere	VARCHAR(100)		opzionale

Tabella Manager

Nome campo	Tipo di dato	Chiave	Vincoli
id_manager	INT	PK	Autoincrement
id_utente	INT	FK	Utente(id), UNIQUE (1,1)
nome_azienda	VARCHAR(200)		opzionale

Tabella Brano

Nome campo	Tipo di dato	Chiavi	Vincoli
id_brano	INT	PK	Autoincrement
id_artista	INT	FK	Artista(id)
titolo	VARCHAR(200)	FK	NOT NULL
data_rilascio	DATE		opzionale
stream_spotify	INT		Default 0 positivo
guadagni_spotify	Decimal(10,2)		Default 0.00

Tabella Concerto

Nome campo	Tipo di dato	Chiave	Vincoli
id_brano	INT	PK	Autoincrement
id_artista	INT	FK	Artista(id)
data	DATE		NOT NULL
luogo	VARCHAR(200)		NOT NULL
biglietti_venduti	INT		Default 0 positivo
guadagno	Decimal(12,2)		Default 0.00

Tabella Contratto

Nome campo	Tipo di dato	Chiave	Vincoli
id_contratto	INT	PK	Autoincrement
id_artista	INT	FK	Artista(id)
id_manager	INT	FK	Manager(id)
data_inizio	DATE		NOT NULL

data_fine	**DATE**		**opzionale**

Vincoli interrelazionali e di integrità

Per garantire la consistenza e la correttezza dei dati memorizzati nel database, il sistema implementa una serie di vincoli logici e strutturali.

Vincolo di integrità referenziale

Il sistema adotta una politica di cancellazione a cascata per prevenire la presenza di dati orfani, e lo fa nel seguente modo:

- Eliminazione Utente: Se un record nella tabella padre Utente viene eliminato, il sistema elimina automaticamente il corrispondente profilo (Artista o Manager) collegato tramite relazione (1,1).
- Eliminazione Artista: Se un profilo Artista viene rimosso, vengono eliminati a cascata tutti i Brani, Concerti e Contratti a esso associati.
- Eliminazione Manager: Se un profilo Manager viene rimosso, vengono eliminati a cascata tutti i Contratti da esso gestiti.

Vincolo di unicità e cardinalità

La relazione tra la tabella Utente e le tabelle di profilo (Artista, Manager) è vincolata da una cardinalità (1,1).

- Un Utente può essere associato a un solo record nella tabella Artista oppure a un solo record nella tabella Manager.
- Non è possibile che più utenti puntino allo stesso profilo, nè che un profilo esista senza che un utente sia associato. Tale vincolo è garantito a livello di database tramite chiave esterna con proprietà UNIQUE.

Vincoli di Dominio

Per assicurare la validità semantica dei dati numerici, sono stati applicati vincoli sui valori accettabili:

- Stream e Biglietti: I campi stream_spotify (tabella Brano) e biglietti_venduti (tabella Concerto) sono definiti come interi positivi (PositiveIntegerField). Il sistema rifuta l’inserimento di valori negativi poichè non avrebbe senso logico avere un numero di ascolti o vendite inferiore a 0.
- Valori di valuta: I campi relativi ai guadagni (guadagni_spotify, guadagno) utilizzano il tipo Decimal per garantire la precisione contabile, evitando errori di approssimazione tipici dei numeri in virgola mobile.

Vincoli di validità temporale (per i Contratti)

Nella tabella Contratto, la validità logica della relazione lavorativa è subordinata alla coerenza delle date. Nonostante il database permetta l'inserimento tecnico, l'applicativo impone che la data_fine, se presente, debba essere cronologicamente successiva alla data_inizio. Un contratto senza data di fine è considerato "a tempo indeterminato" oppure attivo fino alla sua revoca.

Implementazione del sistema informativo

Il sistema è sviluppato con Django,che è formato come segue:
1)paradigma MVT che viene utilizzato in quasi tutti i progetti
2)gestione integrata di utenti, sessioni e autenticazione (hashing delle password bcrypt integrato)
3)interfaccia amministrativa per la gestione dei dati.

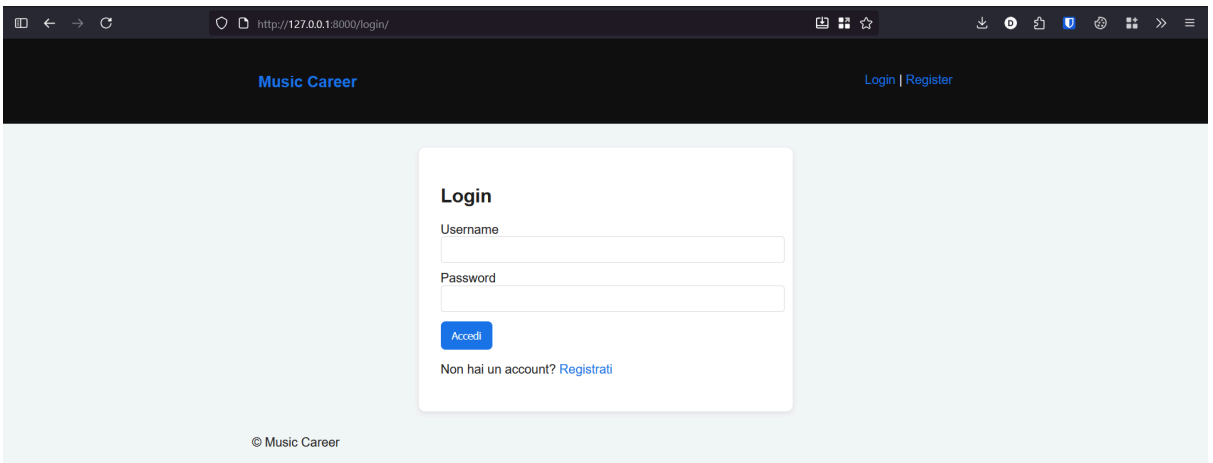
In particolare l'artista può compiere le seguente funzionalità

- Registrazione tramite form dedicato
- Accesso alla dashboard personale
- Visualizzazione di: contratti attivi (data inizio e fine), brani pubblicati e ricavi per ciascuno, concerti organizzati e relativi guadagni, somma totale dei guadagni derivanti da streaming e concerti.

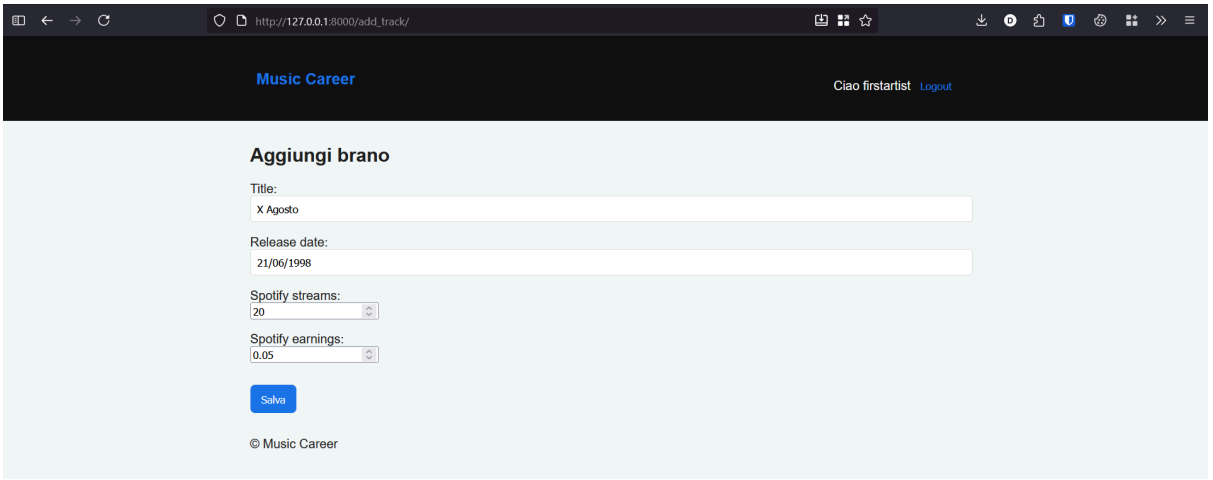
Per quanto riguarda il manager quest'ultimo può:

- Visualizzare la lista di artisti gestiti.
- Accesso ai dati relativi ai contratti, brani e concerti.
- Possibilità di azzerare le statistiche di un artista (ricavi, concerti, stream ecc)
- Per quanto riguarda l'affidabilità dell'applicativo web

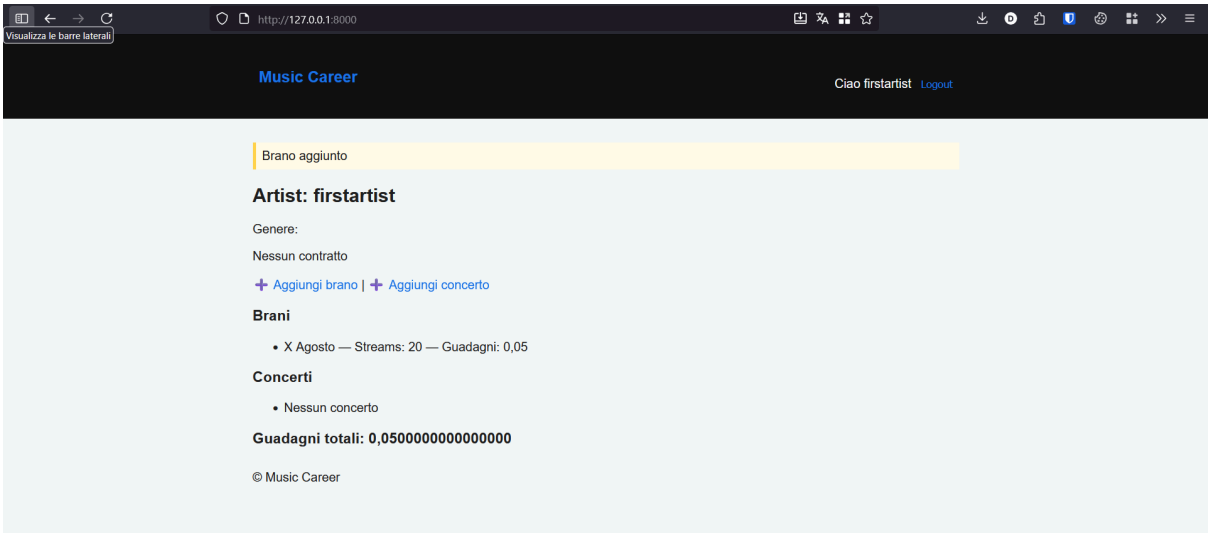
In allegato alcune foto dell'applicativo:



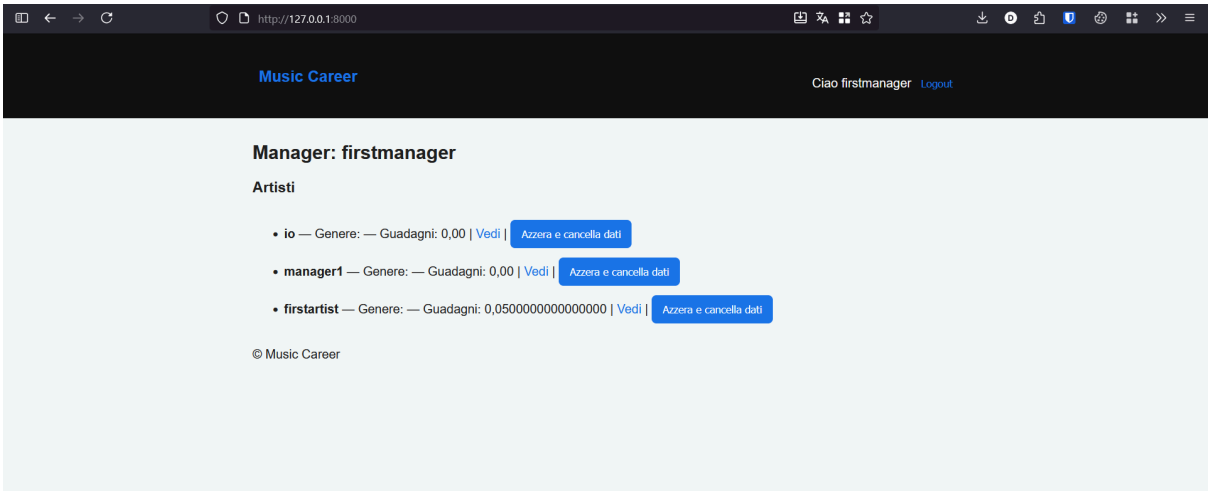
Landing page del sito, in questa schermata è possibile accedere con le proprie credenziali o creare un account.



Inserimento dati della carriera artista.



Dashboard artista aggiornata con l’inserimento dei dati.



Dashboard manager con la facoltà di visualizzare le statistiche di tutti i suoi artisti e di azzerare tutte le statistiche a essi correlate.

Possibili sviluppi futuri

Integrazione API Spotify per ottenere in tempo reale statistiche sugli stream, gestione pagamenti e royalties e dashboard analitiche con grafici di perfomance per artista.
Sicurezza aggiuntiva: Implementare autenticazione a due fattori per proteggere i dati sensibili di contratti e guadagni, oltre all’attuale hashing delle password.